Sylius

Contents

Sylius is a modern e-commerce solution for PHP, based on Symfony Framework.

Note: This documentation assumes you have a working knowledge of the Symfony Framework. If you're not familiar with Symfony, please start with reading the Quick Tour from the Symfony documentation.

Tip: **The Book, Customization Guide, REST API Reference, Cookbook, Contribution Guide and Behat Guide** are chapters describing the usage of **the whole Sylius platform**, on the examples for Sylius-Standard distribution.

For tips on using only some bundles of Sylius head to Bundles and Components docs.
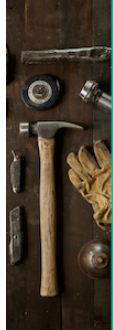

The Book

The Developer's guide to leveraging the flexibility of Sylius. Here you will find all the concepts used in the Sylius platform. *The Book* helps to understand how Sylius works.

introduction

What is Sylius about?

What are the
environments?

installation

How to install Sylius?

Sylius installation
via Vagrant

What are the
requirements?

How to
upgrade Sylius?

architecture

Resources,
State Machines,
Events,
Translations,
Emails,
Contact,
Fixtures,
Events

configuration

Sylius basic concepts
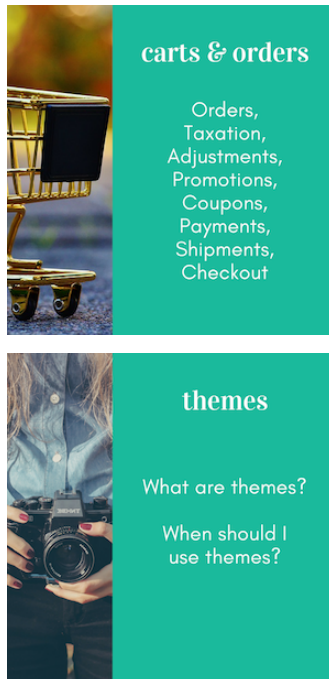configuration:
Channels,
Locales,
Currencies

customers

Customers
ShopUsers
AdminUsers
Addresses

products

Taxons,
Attributes,
Associations,
Reviews,
Inventory,
Pricing,
Search

1.1 The Book

The Developer's guide to leveraging the flexibility of Sylius. Here you will find all the concepts used in Sylius. The Books helps to understand how Sylius works.

1.1.1 Introduction

Introduction aims to describe the philosophy of Sylius. It will also teach you about environments before you start installing it.

Introduction

This is the beginning of the journey with Sylius. We will start with a basic insight into terms that we use in Sylius Documentation.

Introduction to Sylius

Sylius is a game-changing e-commerce solution for PHP, based on the Symfony framework.

Philosophy

Sylius is completely open source (MIT license) and free, maintained by a diverse and creative community of developers and companies.

What are our core values and what makes us different from other solutions?

- Components based approach

- Unlimited flexibility and simple customization

- Developer-friendly, using latest technologies

- Developed using best practices and BDD approach

- Highest quality of code

And much more, but we will let you discover it yourself.

The Three Natures of Sylius

Sylius is constructed from fully decoupled and flexible e-commerce components for PHP. It is also a set of Symfony bundles, which integrate the components into the full-stack framework. On top of that, Sylius is also a complete e-commerce platform crafted from all these building blocks.

It is your choice how to use Sylius, you can benefit from the components with any framework, integrate selected bundles into existing or new Symfony app or built your application on top of Sylius platform.

Sylius Platform

This book is about our **full-stack e-commerce platform**, which is a standard Symfony application providing the most common webshop and a foundation for custom systems.

Leveraging Symfony Bundles

If you prefer to build your very custom system step by step and from scratch, you can integrate the standalone Symfony bundles. For the installation instructions, please refer to the appropriate bundle documentation.

E-Commerce Components for PHP

If you use a different framework than Symfony, you are welcome to use Sylius components, which will make it much easier to implement a webshop with any PHP application and project. They provide you with default models, services and logic for all aspects of e-commerce, completely separated and ready to use.

Final Thoughts

Depending on how you want to use Sylius, continue reading The Book, which covers the usage of the full stack solution, browse the Bundles Reference or learn about The Components.

Understanding Environments

Every Sylius application is the combination of code and a set of configuration that dictates how that code should function. The configuration may define the database being used, whether or not something should be cached, or how verbose logging should be. In Symfony, the idea of "environments" is the idea that the same codebase can be run using multiple different configurations. For example, the dev environment should use configuration that makes development easy and friendly, while the prod environment should use a set of configuration optimized for speed.

Development

Development environment or `dev`, as the name suggests, should be used for development purposes. It is much slower than production, because it uses much less aggressive caching and does a lot of processing on every request. However, it allows you to add new features or fix bugs quickly, without worrying about clearing the cache after every change.

Sylius console runs in `dev` environment by default. You can access the website in dev mode via the `/app_dev.php` file in the `web/` directory. (under your website root)

Production

Production environment or `prod` is your live website environment. It uses proper caching and is much faster than other environments. It uses live APIs and sends out all e-mails.

To run Sylius console in `prod` environment, add the following parameters to every command call:

```
$ bin/console --env=prod --no-debug cache:clear
```

You can access the website in production mode via the `/app.php` file in your website root (`web/`) or just `/` path. (on Apache)

Staging

Staging environment or `staging` is the last line before the shop will go to the production. Here you should test all new features to ensure that everything works as expected. It's almost an exact copy of production environment but with different database and turned off e-mails.

To run Sylius console in `staging` environment, add the following parameters to every command call:

```
$ bin/console --env=staging --no-debug cache:clear
```

You can access the website in staging mode via the `/app_staging.php` file in your website root (`web/`) or just `/` path. (on Apache)

Test

Test environment or `test` is used for automated testing. Most of the time you will not access it directly.

To run Sylius console in `test` environment, add the following parameters to every command call:

```
$ bin/console --env=test cache:clear
```

Final Thoughts

You can read more about Symfony environments in this cookbook article.

- *Introduction to Sylius*
- *Understanding Environments*
- *Introduction to Sylius*
- *Understanding Environments*

1.1.2 Installation

The installation chapter is of course a comprehensive guide to installing Sylius on your machine, but it also provides a general instruction on upgrading Sylius in your project.

Installation

The process of installing Sylius together with the requirements to run it efficiently.

System Requirements

Here you will find the list of system requirements that have to be adhered to be able to use **Sylius**. First of all have a look at the requirements for running Symfony.

Read about the LAMP stack and the MAMP stack.

Operating Systems

The recommended operating systems for running Sylius are the Unix systems - **Linux, MacOS**.

Web server and configuration

In the production environment we do recommend using Apache web server 2.2.

While developing the recommended way to work with your Symfony application is to use PHP's built-in web server.

Go there to see the full reference to the web server configuration.

PHP required modules and configuration

PHP version:

PHP	^7.1

PHP extensions:

gd	No specific configuration
exif	No specific configuration
fileinfo	No specific configuration
intl	No specific configuration

PHP configuration settings:

memory_limit	1024M
date.timezone	Europe/Warsaw

> **Warning:**   Use your local timezone, for example America/Los_Angeles or Europe/Berlin. See http://php.net/manual/en/timezones.php for the list of all available timezones.

Database

By default, the database connection is pre-configured to work with a following MySQL configuration:

MySQL	5.x

Note: Of course you may use any other RDBMS for instance PostgreSQL.

Access rights

Most of the application folders and files require only read access, but a few folders need also the write access for the Apache/Nginx user:

- var/cache
- var/logs
- web/media

You can read how to set these permissions in the Symfony - setting up permissions section.

Installation

The Sylius main application can serve as an end-user app, as well as a foundation for your custom e-commerce application.

Warning: This article assumes you're familiar with Composer, a dependency manager for PHP. It also assumes you have Composer installed globally.

Note: If you downloaded the Composer phar archive, you should use `php composer.phar` where this guide uses `composer`.

Note: In order to inform you about newest Sylius releases and be aware of shops based on Sylius, the Core Team uses an internal statistical service called GUS. The only data that is collected and stored in its database are hostname, user agent, locale, environment (test, dev or prod), current Sylius version and the date of last contact. If you do not want your shop to send requests to GUS, please visit *this guide* for further instructions.

Initiating A New Sylius Project

To create a new project using Sylius Standard Edition, run this command:

```
$ composer create-project sylius/sylius-standard acme
```

Note: Make sure to use PHP ^7.1. Using an older PHP version will result in installing an older version of Sylius.

This will create a new Symfony project in `acme` directory. When all the dependencies are installed, you'll be asked to fill the `parameters.yml` file via interactive script. Please follow the steps. After everything is in place, run the following commands:

```
$ cd acme # Move to the newly created directory
$ php bin/console sylius:install
```

This package has the whole `sylius/sylius` package in vendors, so you can easily update it and focus on your custom development.

Warning: During the `sylius:install` command you will be asked to provide important information, but also its execution ensures that the default **currency** (USD) and the default **locale** (English - US) are set they can be later on changed in the `parameters.yml` file. From now on all the prices will be stored in the database in USD as integers, and all the products will have to be added with a base american english name translation.

Installing assets

In order to see a fully functional frontend you will need to install its assets.

Sylius already has a `gulpfile.babel.js`, therefore you just need to get Gulp using Yarn.

Note: We recommend using stable versions (*^1.0.0*) of Yarn.

Having Yarn installed go to your project directory and run:

```
$ yarn install
```

And now you can use gulp for installing views, by just running a simple command:

```
$ yarn build
```

Accessing the Shop

Tip: We strongly recommend using the Symfony built-in web server by running the `php bin/console server:start --docroot=web 127.0.0.1:8000` command and then accessing `http://127.0.0.1:8000` in your web browser to see the shop.

Note: The localhost's 8000 port may be already occupied by some other process. If so you should try other ports, like for instance: `php bin/console server:start --docroot=web 127.0.0.1:8081` Want to know more about using a built-in server, see here.

You can log in as an administrator, with the credentials you have provided during the installation process. Since now you can play with your clean Sylius installation.

Accessing the Administration Panel

Note: Have a look at the /admin url, where you will find the administration panel. Remember that you have to be logged in as an administrator using the credentials provided while installing Sylius.

How to start developing? - Project Structure

After you have successfully gone through the installation process of **Sylius-Standard** you are probably going to start developing within the framework of Sylius.

In the root directory of your project you will find these important subdirectories:

- `app/config/` - here you will be adding the yaml configuration files including routing, security, state machines configurations etc.
- `var/logs/` - these are the logs of your application
- `var/cache/` - this is the cache of you project
- `src/` - this is where you will be adding all you custom logic in the `AppBundle`
- `web/` - there you will be placing assets of your project

Tip: As it was mentioned before we are basing on Symfony, that is why we've adopted its approach to architecture. Read more in the Symfony documentation. Read also about the best practices while structuring your project.

Contributing

Tip: If you would like to contribute to Sylius - please go to the *Contribution Guide*

Upgrading

Sylius is releasing new versions from time to time. Each release is supported with an UPGRADE file, which is meant to help in the upgrading process, especially for the major versions, which can break the backwards compatibility.

Update the Sylius library version constraint by modifying the `composer.json` **file:**

```json
{
    ...

    "require": {
        "...": "...",

        "sylius/sylius": "^1.0@beta",

        "...": "...",
    },

    ...
}
```

Then run `composer update` **command:**

```
$ composer update sylius/sylius
```

If this results in a dependency error, it may mean that other Sylius dependencies also have to be upgraded. Using this command may help you upgrade Sylius dependencies.

```
$ composer update sylius/sylius --with-dependencies
```

If this does not help, it is a matter of debugging the conflicting versions and working out how your `composer.json` should look after the upgrade.

Finally to make everything work check the UPGRADE file of Sylius for instructions.

One more important thing is running the database migrations:

```
$ bin/console doctrine:migrations:migrate
```

Tip: Check if the migrations (more than 1) are in your `app/migrations` directory. If not, then replace the contents of this directory with the migrations from the `vendor/sylius/sylius/app/migrations/` directory.

After fixing the project according to the upgrade and having run the migrations you are done!

- *System Requirements*

- *Installation*

- *Upgrading*

- *System Requirements*

- *Installation*

- *Upgrading*

1.1.3 Architecture

The key to understanding principles of Sylius internal organization. Here you will learn about the Resource layer, state machines, events and general non e-commerce concepts adopted in the platform, like E-mails or Fixtures.

Architecture

Before we dive separately into every Sylius concept, you need to have an overview of how our main application is structured. In this chapter we will sketch this architecture and our basic, cornerstone concepts, but also some supportive approaches, that you need to notice.

Architecture Overview

Before we dive separately into every Sylius concept, you need to have an overview of how our main application is structured. You already know that Sylius is built from components and Symfony bundles, which are integration layers with the framework.

All bundles share the same conventions for naming things and the way of data persistence. Sylius, by default, uses the Doctrine ORM for managing all entities.

For deeper understanding of how Doctrine works, please refer to the excellent documentation on their official website.

Fullstack Symfony

Sylius is based on Symfony, which is a leading PHP framework to create web applications. Using Symfony allows developers to work better and faster by providing them with certainty of developing an application that is fully compatible with the business rules, that is structured, maintainable and upgradable, but also it allows to save time by providing generic re-usable modules.

Learn more about Symfony.

Doctrine

Doctrine is a family of PHP libraries focused on providing data persistence layer. The most important are the object-relational mapper (ORM) and the database abstraction layer (DBAL). One of Doctrine's key features is the possibility to write database queries in Doctrine Query Language (DQL) - an object-oriented dialect of SQL.

To learn more about Doctrine - see their documentation.

Twig

Twig is a modern template engine for PHP that is really fast, secure and flexible. Twig is being used by Symfony.

To read more about Twig, go here.

Architecture

On the below image you can see the symbolic representation of Sylius architecture.
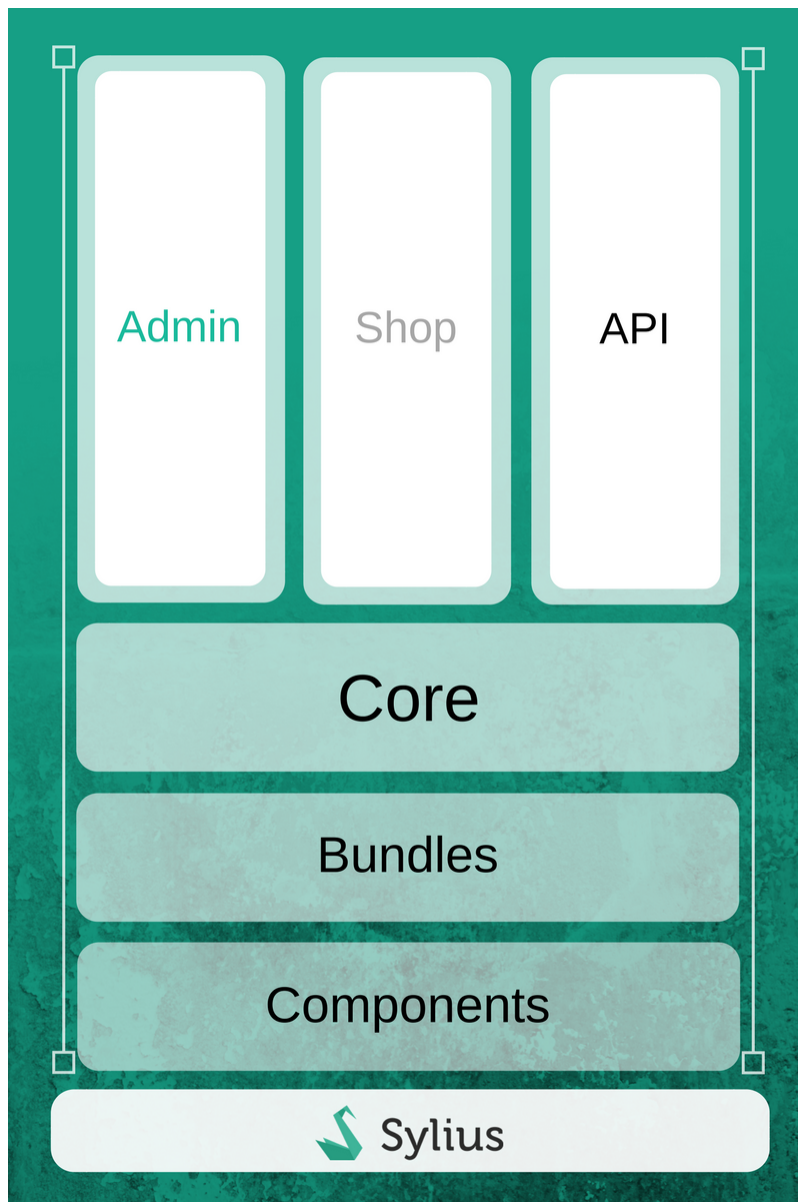
Keep on reading this chapter to learn more about each of its parts: Shop, Admin, API, Core, Components and Bundles.

Division into Components, Bundles, Platform

Components

Every single component of Sylius can be used standalone. Taking the `Taxation` component as an example, its only responsibility is to calculate taxes, it does not matter whether these will be taxes for products or anything else, it is fully decoupled. In order to let the Taxation component operate on your objects you need to have them implementing the `TaxableInterface`. Since then they can have taxes calculated. Such approach is true for every component of

Sylius. Besides components that are strictly connected to the e-commerce needs, we have plenty of components that are more general. For instance Attribute, Mailer, Locale etc.

All the components are packages available via Packagist.

Read more about the Components.

Bundles

These are the Symfony Bundles - therefore if you are a Symfony Developer, and you would like to use the Taxation component in your system, but you do not want to spend time on configuring forms or services in the container. You can include the `TaxationBundle` in your application with minimal or even no configuration to have access to all the services, models, configure tax rates, tax categories and use that for any taxes you will need.

Read more about the Bundles.

Platform

This is a fullstack Symfony Application, based on Symfony Standard. Sylius Platform gives you the classic, quite feature rich webshop. Before you start using Sylius you will need to decide whether you will need a full platform with all the features we provide, or maybe you will use decoupled bundles and components to build something very custom, maybe smaller, with different features. But of course the platform itself is highly flexible and can be easily customized to meet all business requirements you may have.

Division into Core, Admin, Shop, Api

Core

The Core is another component that integrates all the other components. This is the place where for example the `ProductVariant` finally learns that it has a `TaxCategory`. The Core component is where the `ProductVariant` implements the `TaxableInterface` and other interfaces that are useful for its operation. Sylius has here a fully integrated concept of everything that is needed to run a webshop. To get to know more about concepts applied in Sylius - keep on reading *The Book*.

Admin

In every system with the security layer the functionalities of system administration need to be restricted to only some users with a certain role - Administrator. This is the responsibility of our `AdminBundle` although if you do not need it, you can turn it off. Views have been built using the SemanticUI.

Shop

Our `ShopBundle` is basically a standard B2C interface for everything that happens in the system. It is made mainly of yaml configurations and templates. Also here views have been built using the SemanticUI.

Api

Our API uses the REST approach. Since our controllers are format agnostic they have become reusable in the API. Therefore if you request products in the shop frontend you are using exactly the same action as when you are placing the api request. Read more about our API in the *Sylius API Guide*.

Third Party Libraries

Sylius uses a lot of libraries for various tasks:

- Payum for payments

- KnpMenu - for shop and admin menus

- Gaufrette for filesystem abstraction (store images locally, Amazon S3 or external server)

- Imagine for images processing, generating thumbnails and cropping

- Pagerfanta for pagination

- Winzou State Machine - for the state machines handling

Resource Layer

We created an abstraction on top of Doctrine, in order to have a consistent and flexible way to manage all the resources. By "resource" we understand every model in the application. Simplest examples of Sylius resources are "product", "order", "tax_category", "promotion", "user", "shipping_method" and so on. . .

There are two types of resources in **Sylius**:

- registered by default - their names begin with `sylius.*` for example: `sylius.product`

- custom resources, from your application which have a separate convention. We place them under `sylius_resource: resource_name:` in the `config.yml`. For these we recommend using the naming convention of `app.*` for instance `app.my_entity`.

Sylius resource management system lives in the **SyliusResourceBundle** and can be used in any Symfony project.

Services

For every resource you have four essential services available:

- Factory

- Manager

- Repository

- Controller

Let us take the "product" resource as an example. By default, it is represented by an object of a class that implements the `Sylius\Component\Core\Model\ProductInterface`.

Factory

The factory service gives you an ability to create new default objects. It can be accessed via the *sylius.factory.product* id (for the Product resource of course).

```php
<?php

public function myAction()
{
    $factory = $this->container->get('sylius.factory.product');

    /** @var ProductInterface $product **/
    $product = $factory->createNew();
}
```

Note: Creating resources via this factory method makes the code more testable, and allows you to change the model class easily.

Manager

The manager service is just an alias to appropriate Doctrine's ObjectManager and can be accessed via the *sylius.manager.product* id. API is exactly the same and you are probably already familiar with it:

```php
<?php

public function myAction()
{
    $manager = $this->container->get('sylius.manager.product');

    // Assuming that the $product1 exists in the database we can perform such
    ↪operations:
    $manager->remove($product1);

    // If we have created the $product2 using a factory, we can persist it in the
    ↪database.
    $manager->persist($product2);

    // Before performing a flush, the changes we have made, are not saved. There is
    ↪only the $product1 in the database.
    $manager->flush(); // Saves changes in the database.

    //After these operations we have only $product2 in the database. The $product1
    ↪has been removed.
}
```

Repository

Repository is defined as a service for every resource and shares the API with standard Doctrine *ObjectRepository*. It contains two additional methods for creating a new object instance and a paginator provider.

The repository service is available via the *sylius.repository.product* id and can be used like all the repositories you have seen before.

```php
<?php

public function myAction()
{
```

```php
    $repository = $this->container->get('sylius.repository.product');

    $product = $repository->find(4); // Get product with id 4, returns null if not
↪found.
    $product = $repository->findOneBy(['slug' => 'my-super-product']); // Get one
↪product by defined criteria.

    $products = $repository->findAll(); // Load all the products!
    $products = $repository->findBy(['special' => true]); // Find products matching
↪some custom criteria.
}
```

Tip: An important feature of the repositories are the `add($resource)` and `remove($resource)` methods, which take a resource as an argument and perform the adding/removing action with a flush inside.

These actions can be used when the performance of operations may be neglected. If you are willing to perform operations on sets of data we are suggesting to use the manager instead.

Every Sylius repository supports paginating resources. To create a Pagerfanta instance use the `createPaginator` method:

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');

    $products = $repository->createPaginator();
    $products->setMaxPerPage(3);
    $products->setCurrentPage($request->query->get('page', 1));

    // Now you can return products to template and iterate over it to get products
↪from current page.
}
```

Paginator can be created for a specific criteria and with desired sorting:

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');

    $products = $repository->createPaginator(['foo' => true], ['createdAt' => 'desc
↪']);
    $products->setMaxPerPage(3);
    $products->setCurrentPage($request->query->get('page', 1));
}
```

Controller

This service is the most important for every resource and provides a format agnostic CRUD controller with the following actions:

- [GET] showAction() for getting a single resource

- [GET] indexAction() for retrieving a collection of resources

- [GET/POST] createAction() for creating new resource

- [GET/PUT] updateAction() for updating an existing resource

- [DELETE] deleteAction() for removing an existing resource

As you see, these actions match the common operations in any REST API and yes, they are format agnostic. This means, all Sylius controllers can serve HTML, JSON or XML, depending on what you request.

Additionally, all these actions are very flexible and allow you to use different templates, forms, repository methods per route. The bundle is very powerful and allows you to register your own resources as well. To give you some idea of what is possible, here are some examples!

Displaying a resource with a custom template and repository methods:

```yaml
# routing.yml
app_product_show:
    path: /products/{slug}
    methods: [GET]
    defaults:
        _controller: sylius.controller.product:showAction
        _sylius:
            template: AppStoreBundle:Product:show.html.twig # Use a custom template.
            repository:
                method: findForStore # Use a custom repository method.
                arguments: [$slug] # Pass the slug from the url to the repository.
```

Creating a product using custom form and a redirection method:

```yaml
# routing.yml
app_product_create:
    path: /my-stores/{store}/products/new
    methods: [GET, POST]
    defaults:
        _controller: sylius.controller.product:createAction
        _sylius:
            form: AppStoreBundle/Form/Type/CustomFormType # Use this form type!
            template: AppStoreBundle:Product:create.html.twig # Use a custom template.
            factory:
                method: createForStore # Use a custom factory method to create a
→product.
                arguments: [$store] # Pass the store name from the url.
            redirect:
                route: app_product_index # Redirect the user to their products.
                parameters: [$store]
```

All other methods have the same level of flexibility and are documented in the *Resource Bundle Guide*.

State Machine

In **Sylius** we are using the Winzou StateMachine Bundle. State Machines are an approach to handling changes occurring in the system frequently, that is extremely flexible and very well organised.

Every state machine will have a predefined set of states, that will be stored on an entity that is being controlled by it. These states will have a set of defined transitions between them, and a set of callbacks - a kind of events, that will happen on defined transitions.

States

States of a state machine are defined as constants on the model of an entity that the state machine is controlling.

How to configure states? Let's see on the example from **Checkout** state machine.

```
# CoreBundle/Resources/config/app/state_machine/sylius_order_checkout.yml
winzou_state_machine:
    sylius_order_checkout:
        # list of all possible states:
        states:
            cart: ~
            addressed: ~
            shipping_selected: ~
            payment_selected: ~
            completed: ~
```

Transitions

On the graph it would be the connection between two states, defining that you can move from one state to another subsequently.

How to configure transitions? Let's see on the example of our **Checkout** state machine. Having states configured we can have a transition between the cart state to the addressed state.

```
# CoreBundle/Resources/config/app/state_machine/sylius_order_checkout.yml
winzou_state_machine:
    sylius_order_checkout:
        transitions:
            address:
                from: [cart, addressed, shipping_selected, payment_selected]  # here
→you specify which state is the initial
                to: addressed                                                  # there
→you specify which state is final for that transition
```

Callbacks

Callbacks are used to execute some code before or after applying transitions. Winzou StateMachineBundle adds the ability to use Symfony services in the callbacks.

How to configure callbacks? Having a configured transition, you can attach a callback to it either before or after the transition. Callback is simply a method of a service you want to be executed.

```
# CoreBundle/Resources/config/app/state_machine/sylius_order_checkout.yml
winzou_state_machine:
    sylius_order_checkout:
        callbacks:
            # callbacks may be called before or after specified transitions, in
→the checkout state machine we've got callbacks only after transitions
            after:
                sylius_process_cart:
                    on: ["address", "select_shipping", "select_payment"]
                    do: ["@sylius.order_processing.order_processor", "process"]
                    args: ["object"]
```

Configuration

In order to use a state machine, you have to define a graph beforehand. A graph is a definition of states, transitions and optionally callbacks - all attached on an object from your domain. Multiple graphs may be attached to the same object.

In **Sylius** the best example of a state machine is the one from checkout. It has five states available: `cart`, `addressed`, `shipping_selected`, `payment_selected` and `completed` - which can be achieved by applying some transitions to the entity. For example, when selecting a shipping method during the shipping step of checkout we should apply the `select_shipping` transition, and after that the state would become `shipping_selected`.

```yaml
# CoreBundle/Resources/config/app/state_machine/sylius_order_checkout.yml
winzou_state_machine:
    sylius_order_checkout:
        class: "%sylius.model.order.class%" # class of the domain object - in our
↪case Order
        property_path: checkoutState
        graph: sylius_order_checkout
        state_machine_class: "%sylius.state_machine.class%"
        # list of all possible states:
        states:
            cart: ~
            addressed: ~
            shipping_selected: ~
            payment_selected: ~
            completed: ~
        # list of all possible transitions:
        transitions:
            address:
                from: [cart, addressed, shipping_selected, payment_selected] # here
↪you specify which state is the initial
                to: addressed                                                # there
↪you specify which state is final for that transition
            select_shipping:
                from: [addressed, shipping_selected, payment_selected]
                to: shipping_selected
            select_payment:
                from: [payment_selected, shipping_selected]
                to: payment_selected
            complete:
                from: [payment_selected]
                to: completed
        # list of all callbacks:
        callbacks:
        # callbacks may be called before or after specified transitions, in the
↪checkout state machine we've got callbacks only after transitions
            after:
                sylius_process_cart:
                    on: ["address", "select_shipping", "select_payment"]
                    do: ["@sylius.order_processing.order_processor", "process"]
                    args: ["object"]
                sylius_create_order:
                    on: ["complete"]
                    do: ["@sm.callback.cascade_transition", "apply"]
                    args: ["object", "event", "'create'", "'sylius_order'"]
                sylius_hold_inventory:
                    on: ["complete"]
```

(continues on next page)

```yaml
                do: ["@sylius.inventory.order_inventory_operator", "hold"]
                args: ["object"]
            sylius_assign_token:
                on: ["complete"]
                do: ["@sylius.unique_id_based_order_token_assigner",
→"assignTokenValueIfNotSet"]
                args: ["object"]
```

Learn more

- Winzou StateMachine Bundle
- *Customization guide: State machines*

Translations

Sylius uses the approach of personal translations - where each entity is bound with a translation entity, that has it's own table (instead of keeping all translations in one table for the whole system). This results in having the `ProductTranslation` class and `sylius_product_translation` table for the `Product` entity.

The logic of handling translations in Sylius is in the **ResourceBundle**

The fields of an entity that are meant to be translatable are saved on the translation entity, only their getters and setters are also on the original model.

Let's see an example:

Assuming that we would like to have a translatable model of a `Supplier`, we need a Supplier class and a Supplier-Translation class.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\AbstractTranslation;

class SupplierTranslation extends AbstractTranslation
{
    /**
     * @var string
     */
    protected $name;

    /**
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * @param string $name
     */
    public function setName($name)
```

```
    {
        $this->name = $name;
    }
}
```

The actual entity has access to its translation by using the `TranslatableTrait` which provides the `getTranslation()` method.

> **Warning:** Remember that the **Translations collection** of the entity (from the TranslatableTrait) has to be initialized in the constructor!

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\TranslatableInterface;
use Sylius\Component\Resource\Model\TranslatableTrait;

class Supplier implements TranslatableInterface
{
    use TranslatableTrait {
        __construct as private initializeTranslationsCollection;
    }

    public function __construct()
    {
        $this->initializeTranslationsCollection();
    }

    /**
     * @return string
     */
    public function getName()
    {
        return $this->getTranslation()->getName();
    }

    /**
     * @param string $name
     */
    public function setName($name)
    {
        $this->getTranslation()->setName($name);
    }
}
```

Fallback Translations

The `getTranslation()` method gets a translation for the current locale, while we are in the shop, but we can also manually impose the locale - `getTranslation('pl_PL')` will return a polish translation **if there is a translation in this locale**.

But when the translation for the chosen locale is unavailable, instead the translation for the **fallback locale** (the one

that was either set in `config.yml` or using the `setFallbackLocale()` method from the TranslatableTrait on the entity) is used.

How to add a new translation programmatically?

You can programmatically add a translation to any of the translatable resources in Sylius. Let's see how to do it on the example of a ProductTranslation.

```php
// Find a product to add a translation to it

/** @var ProductInterface $product */
$product = $this->container->get('sylius.repository.product')->findOneBy(['code' =>
→'radiohead-mug-code']);

// Create a new translation of product, give it a translated name and slug in the
→chosen locale

/** @var ProductTranslation $translation */
$translation = new ProductTranslation();

$translation->setLocale('pl_PL');
$translation->setName('Kubek Radiohead');
$translation->setSlug('kubek-radiohead');

// Add the translation to your product
$product->addTranslation($translation);

// Remember to save the product after adding the translation
$this->container->get('sylius.manager.product')->flush($product);
```

Learn more

- *Resource - translations documentation*
- *Locales - concept documentation*

E-Mails

Sylius is sending various e-mails and this chapter is a reference about all of them. Continue reading to learn what e-mails are sent, when and how to customize the templates. To understand how e-mail sending works internally, please refer to *SyliusMailerBundle documentation*. And to learn more about mailer services configuration, read *the dedicated cookbook*.

User Confirmation

Every time a customer registers via the registration form, a user registration e-mail is sent to them.

Code: `user_registration`

The default template: `SyliusShopBundle:Email:userRegistration.html.twig`

You also have the following parameters available:

- `user`: Instance of the user model

Email Verification

When a customer registers via the registration form, besides the User Confirmation an Email Verification is sent.

Code: `verification_token`

The default template: `SyliusShopBundle:Email:verification.html.twig`

You also have the following parameters available:

- `user`: Instance of the user model

Password Reset

This e-mail is used when the user requests to reset their password in the login form.

Code: `reset_password_token`

The default template: `SyliusShopBundle:Email:passwordReset.html.twig`

You also have the following parameters available:

- `user`: Instance of the user model

Order Confirmation

This e-mail is sent when order is placed.

Code: `order_confirmation`

The default template: `SyliusShopBundle:Email:orderConfirmation.html.twig`

You also have the following parameters available:

- `order`: Instance of the order, with all its data

Shipment Confirmation

This e-mail is sent when the order's shipping process has started.

Code: `shipment_confirmation`

The default template: `SyliusAdminBundle:Email:shipmentConfirmation.html.twig`

You have the following parameters available:

- `shipment`: Shipment instance
- `order`: Instance of the order, with all its data

How to send an Email programmatically?

For sending emails **Sylius** is using a dedicated service - **Sender**. Additionally we have **EmailManagers** for Order Confirmation(OrderEmailManager) and for Shipment Confirmation(ShipmentEmailManager).

Tip: While using **Sender** you have the available emails of Sylius available under constants in:

- Core - Emails

- User - Emails

Example using **Sender**:

```
/** @var SenderInterface $sender */
$sender = $this->container->get('sylius.email_sender');

$sender->send(\Sylius\Bundle\UserBundle\Mailer\Emails::EMAIL_VERIFICATION_TOKEN, [
↪'bannanowa@gmail.com'], ['user' => $user]);
```

Example using **EmailManager**:

```
/** @var OrderEmailManagerInterface $sender */
$orderEmailManager = $this->container->get('sylius.email_manager.order');

$orderEmailManager->sendConfirmationEmail($order);
```

Learn more

- *Mailer - Component Documentation*
- *Mailer - Bundle Documentation*

Contact

The functionality of contacting the shop support/admin is in Sylius very basic. Each **Channel** of your shop may have a `contactEmail` configured on it. This will be the email address to support.

Contact form

The contact form can be found on the `/contact` route.

Note: When the `contactEmail` is not configured on the channel, the customer will see the following flash message:
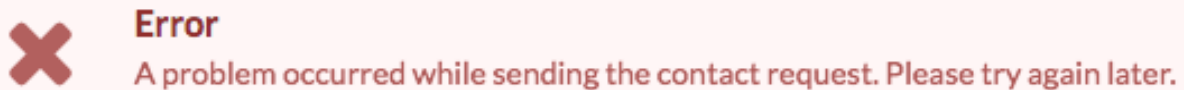
The form itself has only two fields `email` (which will be filled automatically for the logged in users) and `message`.

ContactEmailManager

The **ContactEmailManager** service is responsible for the sending of a contact request email. It can be found under the `sylius.email_manager.contact` service id.

ContactController

The controller responsible for the request action handling is the **ContactController**. It has the `sylius.controller.shop.contact` service id.

Configuration

The routing for contact can be found in the `Sylius/Bundle/ShopBundle/Resources/config/routing/contact.yml` file. By overriding that routing you will be able to customize **redirect url, error flash, success flash, form and its template**.

You can also change the template of the email that is being sent by simply overriding it in your project in the `app/Resources/SyliusShopBundle/views/Email/contactRequest.html.twig` file.

Learn more

- *Emails - Documentation*

Fixtures

Fixtures are used mainly for testing, but also for having your shop in a certain state, having defined data - they ensure that there is a fixed environment in which your application is working.

Note: They way Fixtures are designed in Sylius is well described in the *FixturesBundle documentation*.

What are the available fixtures in Sylius?

To check what fixtures are defined in Sylius run:

```
$ php bin/console sylius:fixtures:list
```

How to load Sylius fixtures?

The recommended way to load the predefined set of Sylius fixtures is here:

```
$ php bin/console sylius:fixtures:load
```

What data is loaded by fixtures in Sylius?

All files that serve for loading fixtures of Sylius are placed in the `Sylius/Bundle/CoreBundle/Fixture/*` directory.

And the specified data for fixtures is stored in the Sylius/Bundle/CoreBundle/Resources/config/app/fixtures.yml file.

Learn more

- *FixturesBundle documentation*

Events

Tip: You can learn more about events in general in the Symfony documentation.

What is the naming convention of Sylius events?

The events that are designed for the entities have a general naming convention: `sylius.entity_name.event_name`.

The examples of such events are: `sylius.product.pre_update`, `sylius.shop_user.post_create`, `sylius.taxon.pre_create`.

Events reference

All Sylius bundles are using *SyliusResourceBundle*, which has some built-in events.

Event	Description
sylius.<resource>.pre_create	Before persist
sylius.<resource>.post_create	After flush
sylius.<resource>.pre_update	Before flush
sylius.<resource>.post_update	After flush
sylius.<resource>.pre_delete	Before remove
sylius.<resource>.post_delete	After flush
sylius.<resource>.initialize_create	Before creating view
sylius.<resource>.initialize_update	Before creating view

CRUD events rules

As you should already know, every resource controller is represented by the `sylius.controller.<resource_name>` service. Several useful events are dispatched during execution of every default action of this controller. When creating a new resource via the `createAction` method, 2 events occur.

First, before the `persist()` is called on the resource, the `sylius.<resource_name>.pre_create` event is dispatched.

And after the data storage is updated, `sylius.<resource_name>.post_create` is triggered.

The same set of events is available for the `update` and `delete` operations. All the dispatches are using the `GenericEvent` class and return the resource object by the `getSubject` method.

What events are already used in Sylius?

Even though Sylius has events as entry points to each resource only some of these points are already used in our usecases.

The events already used in Sylius are described in the Book alongside the concepts they concern.

Tip: What is more you can easily check all the Sylius events in your application by using this command:

```
$ php bin/console debug:event-dispatcher | grep sylius
```

Customizations

Note: Customizing logic via Events vs. State Machines

The logic in which Sylius operates can be customized in two ways. First of them is using the state machines: what is really useful when you need to modify business logic for instance modify the flow of the checkout, and the second is listening on the kernel events related to the entities, which is helpful for modifying the HTTP responses visible directly to the user, like displaying notifications, sending emails.

Learn more

- *Sylius Documentation: The Book*
- *Architecture Overview*
- *Resource Layer*
- *State Machine*
- *Translations*
- *E-Mails*
- *Contact*
- *Fixtures*
- *Events*
- *Architecture Overview*
- *Resource Layer*
- *State Machine*
- *Translations*
- *E-Mails*
- *Contact*
- *Fixtures*
- *Events*

1.1.4 Configuration

Having knowledge about basics of our architecture we will introduce the three most important concepts - Channels, Locales and Currencies. These things have to be configured before you will have a Sylius application up and running.

Configuration

Having knowledge about basics of our *architecture* we will introduce the three most important concepts - Channels, Locales and Currencies. These things have to be configured before you will have a Sylius application up and running.

Channels

In the modern world of e-commerce your website is no longer the only point of sale for your goods.

Channel model represents a single sales channel, which can be one of the following things:

- Webstore
- Mobile application
- Cashier in your physical store

Or pretty much any other channel type you can imagine.

What may differ between channels? Particularly anything from your shop configuration:

- products,
- currencies,
- locales (language),
- themes,
- hostnames,
- taxes,
- payment and shipping methods.

A **Channel** has a `code`, a `name` and a `color`.

In order to make the system more convenient for the administrator - there is just one, shared admin panel. Also users are shared among the channels.

Tip: In the dev environment you can easily check what channel you are currently on in the Symfony debug toolbar.
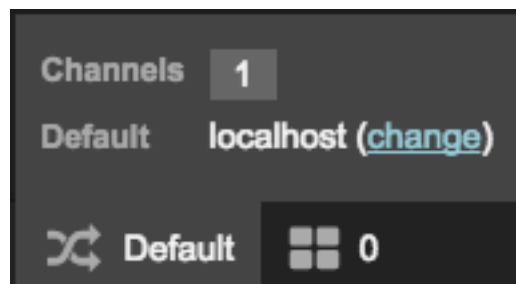
How to get the current channel?

You can get the current channel from the channel context.

```
$channel = $this->container->get('sylius.context.channel')->getChannel();
```

> **Warning:** Beware! When using multiple channels, remember to configure `hostname` for **each** of them. If missing, default context would not be able to provide appropriate channel and it will result in an error.

> **Note:** The channel is by default determined basing on the hostname, but you can customize that behaviour. To do that you have to implement the `Sylius\Component\Channel\Context\ChannelContextInterface` and register it as a service under the `sylius.context.channel` tag. Optionally you can add a `priority="-64"` since the default ChannelContext has a `priority="-128"`, and by default a `priority="0"` is assigned.

> **Note:** Moreover if the channel depends mainly on the request you can implement the `Sylius\Component\Channel\Context\RequestBased\RequestResolverInterface` with its `findChannel(Request $request)` method and register it under the `sylius.context.channel.request_based.resolver` tag.

Learn more

- *Channel - Component Documentation*.

> **Note:** In order to add a new locale to your store you have to assign it to a channel.

Locales

To support multiple languages we are using **Locales** in **Sylius**. Locales are language codes standardized by the ISO 15897.

> **Tip:** In the dev environment you can easily check what locale you are currently using in the Symfony debug toolbar:
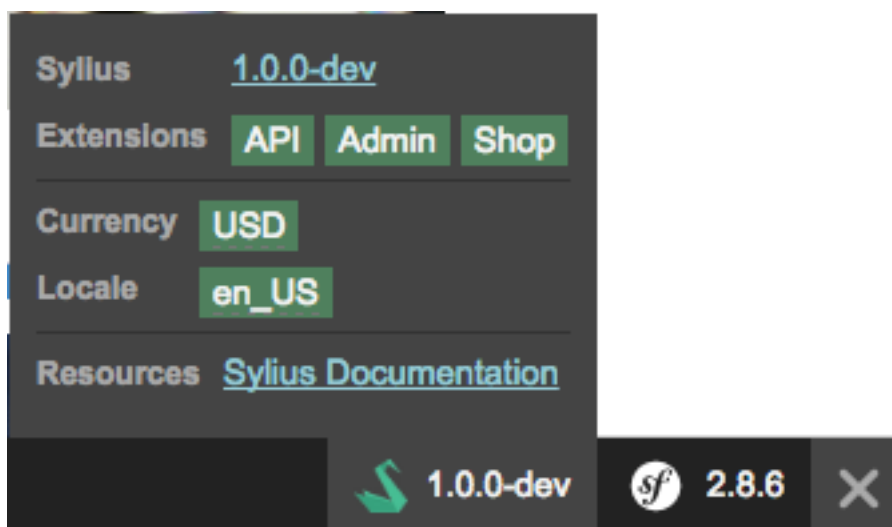
Base Locale

During the *installation* you provided a default base locale. This is the language in which everything in your system will be saved in the database - all the product names, texts on website, e-mails etc.

Locale Context

To manage the currently used language, we use the **LocaleContext**. You can always access it with the ID `sylius.context.locale` in the container.

```php
<?php

public function fooAction()
{
    $locale = $this->get('sylius.context.locale')->getLocaleCode();
}
```

The locale context can be injected into any of your services and give you access to the currently used locale.

Available Locales Provider

The Locale Provider service (`sylius.locale_provider`) is responsible for returning all languages available for the current user. By default, returns all configured locales. You can easily modify this logic by overriding this service.

```php
<?php

public function fooAction()
{
    $locales = $this->get('sylius.locale_provider')->getAvailableLocalesCodes();

    foreach ($locales as $locale) {
        echo $locale;
    }
}
```

To get all languages configured in the store, regardless of your availability logic, use the locales repository:

```php
<?php

$locales = $this->get('sylius.repository.locale')->findAll();
```

Learn more

- *Locale - Component Documentation*.

Currencies

Sylius supports multiple currencies per store and makes it very easy to manage them.

There are several approaches to processing several currencies, but we decided to use the simplest solution we are storing all money values in the **base currency per channel** and convert them to other currencies with exchange rates.

Note: The **base currency** to the first channel is set during the installation of Sylius and it has the **exchange rate** equal to "1.000".

Tip: In the dev environment you can easily check the base currency in the Symfony debug toolbar:
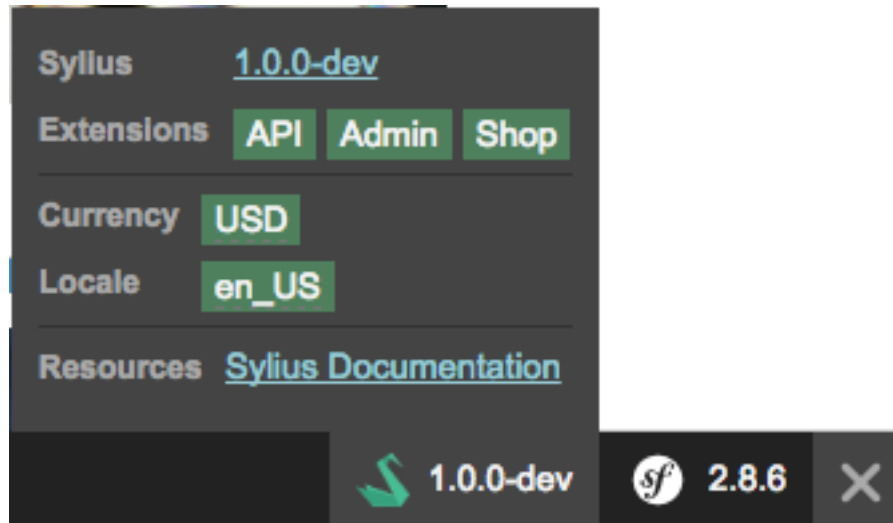
Currency Context

By default, user can switch the current currency in the frontend of the store.

To manage the currently used currency, we use the **CurrencyContext**. You can always access it through the `sylius.context.currency` id.

```php
<?php

public function fooAction()
{
    $currency = $this->get('sylius.context.currency')->getCurrency();
}
```

Currency Converter

The `Sylius\Component\Currency\Converter\CurrencyConverter` is a service available under the `sylius.currency_converter` id.

It allows you to convert money values from one currency to another.

This solution is used for displaying an *approximate* value of price when the desired currency is different from the base currency of the current channel.

Available Currencies Provider

The default menu for selecting currency is using a service - **CurrencyProvider** - with the `sylius.currency_provider` id, which returns all enabled currencies. This is your entry point if you would like override this logic and return different currencies for various scenarios.

```php
<?php

public function fooAction()
{
    $currencies = $this->get('sylius.currency_provider')->getAvailableCurrencies();
}
```

Switching Currency of a Channel

We may of course change the currency used by a channel. For that we have the `sylius.storage.currency` service, which implements the `Sylius\Component\Core\Currency\CurrencyStorageInterface` with methods `->set(ChannelInterface $channel, $currencyCode)` and `->get(ChannelInterface $channel)`.

```php
$container->get('sylius.storage.currency')->set($channel, 'PLN');
```

Displaying Currencies in the templates

There are some useful helpers for rendering money values in the front end. Simply import the money macros of the `ShopBundle` in your twig template and use the functions to display the value:

```twig
..
{% import "@SyliusShop/Common/Macro/money.html.twig" as money %}
..

<span class="price">{{ money.format(price, 'EUR') }}</span>
```

Sylius provides you with some handy *Global Twig variables* to facilitate displaying money values even more.

Learn more

- *Currency - Component Documentation*
- *Pricing Concept Documentation*
- *Channels*
- *Locales*
- *Currencies*
- *Channels*
- *Locales*
- *Currencies*

1.1.5 Customers

This chapter will tell you more about the way Sylius handles users, customers and admins. There is also a subchapter dedicated to addresses of your customers.

Customers

Customer and ShopUser

For handling customers of your system **Sylius** is using a combination of two entities - **Customer** and **ShopUser**. The difference between these two entities is simple: the **Customer** is a guest in your shop and the **ShopUser** is a user registered in the system - they have an account.

Customer

The Customer entity was created to collect data about non-registered guests of the system - ones that has been buying without having an account or that have somehow provided us their e-mail.

How to create a Customer programmatically?

As usual, use a factory. The only required field for the Customer entity is `email`, provide it before adding it to the repository.

```
/** @var CustomerInterface $customer */
$customer = $this->container->get('sylius.factory.customer')->createNew();

$customer->setEmail('customer@test.com');

$this->container->get('sylius.repository.customer')->add($customer);
```

The Customer entity can of course hold other information besides an email, it can be for instance `firstName`, `lastName` or `birthday`.

Note: The relation between the Customer and ShopUser is bidirectional. Both entities hold a reference to each other.

ShopUser

ShopUser entity is designed for customers that have registered in the system - they have an account with both e-mail and password. They can visit and modify their account.

While creating new account the existence of the provided email in the system is checked - if the email was present - it will already have a Customer therefore the existing one will be assigned to the newly created ShopUser, if not - a new Customer will be created together with the ShopUser.

How to create a ShopUser programmatically?

Assuming that you have a Customer (either retrieved from the repository or a newly created one) - use a factory to create a new ShopUser, assign the existing Customer and a password via the `setPlainPassword()` method.

```
/** @var ShopUserInterface $user */
$user = $this->container->get('sylius.factory.shop_user')->createNew();

// Now let's find a Customer by their e-mail:
/** @var CustomerInterface $customer */
$customer = $this->container->get('sylius.repository.customer')->findOneBy(['email' =>
↪ 'customer@test.com']);

// and assign it to the ShopUser
$user->setCustomer($customer);
$user->setPlainPassword('pswd');

$this->container->get('sylius.repository.shop_user')->add($user);
```

Changing the ShopUser password

The already set password of a **ShopUser** can be easily changed via the `setPlainPassword()` method.

```
$user->getPassword(); // returns encrypted password - 'pswd'

$user->setPlainPassword('resu1');
// the password will now be 'resu1' and will become encrypted while saving the user
↪in the database
```

Customer related events

Event id	Description
`sylius.customer.post_register`	dispatched when a new Customer is registered
`sylius.customer.pre_update`	dispatched when a Customer is updated
`sylius.oauth_user.post_create`	dispatched when an OAuthUser is created
`sylius.oauth_user.post_update`	dispatched when an OAuthUser is updated
`sylius.shop_user.post_create`	dispatched when a User is created
`sylius.shop_user.post_update`	dispatched when a User is updated
`sylius.shop_user.pre_delete`	dispatched before a User is deleted
`sylius.user.email_verification.token`	dispatched when a verification token is requested
`sylius.user.password_reset.request.token`	dispatched when a reset password token is requested
`sylius.user.pre_password_change`	dispatched before user password is changed
`sylius.user.pre_password_reset`	dispatched before user password is reset
`sylius.user.security.implicit_login`	dispatched when an implicit login is done
`security.interactive_login`	dispatched when an interactive login is done

Learn more

- *User - Component Documentation*

AdminUser

The **AdminUser** entity extends the **User** entity. It is created to enable administrator accounts that have access to the administration panel.

How to create an AdminUser programmatically?

The **AdminUser** is created just like every other entity, it has its own factory. By default it will have an administration **role** assigned.

```php
/** @var AdminUserInterface $admin */
$admin = $this->container->get('sylius.factory.admin_user')->createNew();

$admin->setEmail('administrator@test.com');
$admin->setPlainPassword('pswd');

$this->container->get('sylius.repository.admin_user')->add($admin);
```

Administration Security

In **Sylius** by default you have got the administration panel routes (/admin/*) secured by a firewall - its configuration can be found in the security.yml file.

Only the logged in **AdminUsers** are eligible to enter these routes.

Learn more

- *Customer & ShopUser - Documentation*

Addresses

Countries

Countries are a part of the *Addressing* concept. The **Country** entity represents a real country that your shop is willing to sell its goods in (for example the UK). It has an ISO code to be identified easily (ISO 3166-1 alpha-2).

Countries might also have **Provinces**, which is in fact a general name for an administrative division, within a country. Therefore we understand provinces as states of the USA, voivodeships of Poland, cantons of Belgium or Bundesländer of Germany.

How to add a country?

To give you a better insight into Countries, let's have a look on how to prepare and add a Country to the system programmatically. We will do it with a province at once.

You will need factories for countries and provinces in order to create them:

```
/** @var CountryInterface $country */
$country = $this->container->get('sylius.factory.country')->createNew();

/** @var ProvinceInterface $province */
$province = $this->container->get('sylius.factory.province')->createNew();
```

To the newly created objects assign codes.

```
// US - the United States of America
$country->setCode('US');
// US_CA - California
$province->setCode('US_CA');
```

Provinces may be added to a country via a collection. Create one and add the province object to it and using the prepared collection add the province to the Country.

```
$provinces = new ArrayCollection();
$provinces->add($province);

$country->setProvinces($provinces);
```

You can of course simply add single province:

```
$country->addProvince($province);
```

Finally you will need a repository for countries to add the country to your system.

```
/** @var RepositoryInterface $countryRepository */
$countryRepository = $this->get('sylius.repository.country');

$countryRepository->add($country);
```

From now on the country will be available to use in your system.

Learn more

- *Addressing - Bundle Documentation*
- *Addressing - Component Documentation*

Zones

Zones are a part of the *Addressing* concept.

Zones and ZoneMembers

Zones consist of **ZoneMembers**. It can be any kind of zone you need - for instance if you want to have all the EU countries in one zone, or just a few chosen countries that have the same taxation system in one zone, or you can even distinguish zones by the ZIP code ranges in the USA.

Three different types of zones are available:

- **country** zone, which consists of countries.
- **province** zone, which is constructed from provinces.

- **zone**, which is a group of other zones.

How to add a Zone?

Let's see how you can add a Zone to your system programmatically.

Firstly you will need a factory for zones - There is a specific one.

```
/** @var ZoneFactoryInterface $zoneFactory */
$zoneFactory = $this->container->get('sylius.factory.zone');
```

Using the ZoneFactory create a new zone with its members. Let's take the UK as an example.

```
/** @var ZoneInterface $zone */
$zone = $zoneFactory->createWithMembers(['GB_ENG', 'GB_NIR', 'GB_SCT'. 'GB_WLS']);
```

Now give it a code, name and type:

```
$zone->setCode('GB');
$zone->setName('United Kingdom');
// available types are the type constants from the ZoneInterface
$zone->setType(ZoneInterface::TYPE_PROVINCE);
```

Finally get the zones repository from the container and add the newly created zone to the system.

```
/** @var RepositoryInterface $zoneRepository */
$zoneRepository = $this->container->get('sylius.repository.zone');

$zoneRepository->add($zone);
```

Matching a Zone

Zones are not very useful alone, but they can be a part of a complex taxation/shipping or any other system. A service implementing the *ZoneMatcherInterface* is responsible for matching the **Address** to a specific **Zone**.

```
/** @var ZoneMatcherInterface $zoneMatcher */
$zoneMatcher = $this->get('sylius.zone_matcher');
$zone = $zoneMatcher->match($user->getAddress());
```

ZoneMatcher can also return all matching zones. (not only the most suitable one)

```
/** @var ZoneMatcherInterface $zoneMatcher */
$zoneMatcher = $this->get('sylius.zone_matcher');
$zones = $zoneMatcher->matchAll($user->getAddress());
```

Internally, Sylius uses this service to define the shipping and billing zones of an *Order*, but you can use it for many different things and it is totally up to you.

Learn more

- *Addressing - Bundle Documentation*
- *Addressing - Component Documentation*

Addresses

Every address in Sylius is represented by the **Address** model. It has a few important fields:

- `firstName`
- `lastName`
- `phoneNumber`
- `company`
- `countryCode`
- `provinceCode`
- `street`
- `city`
- `postcode`

Note: The Address has a relation to a **Customer** - which is really useful during the *Checkout addressing step*.

How to create an Address programmatically?

In order to create a new address, use a factory. Then complete your address with required data.

```php
/** @var AddressInterface $address */
$address = $this->container->get('sylius.factory.address')->createNew();

$address->setFirstName('Harry');
$address->setLastName('Potter');
$address->setCompany('Ministry of Magic');
$address->setCountryCode('UK');
$address->setProvinceCode('UKJ');
$address->setCity('Little Whinging');
$address->setStreet('4 Privet Drive');
$address->setPostcode('000001');

// and finally having the address you can assign it to any Order
$order->setShippingAddress($address);
```

Learn more

- *Addressing - Component Documentation*
- *Addressing - Bundle Documentation*

Address Book

The Address Book concept is a very convenient solution for the customers of your shop, that come back. Once they provide an address it is saved in the system and can be reused the next time.

Sylius handles the address book in a not complex way:

The Addresses Collection on a Customer

On the Customer entity we are holding a collection of addresses:

```php
/**
 * @var Collection|AddressInterface[]
 */
protected $addresses;
```

We can operate on it as usual - by adding and removing objects.

Besides the Customer entity has a **default address** field that is the default address used both for shipping and billing, the one that will be filling the form fields by default.

How to add an address to the address book manually?

If you would like to add an address to the collection of Addresses of a chosen customer that's all what you should do:

Create a new address:

```php
/** @var AddressInterface $address */
$address = $this->container->get('sylius.factory.address')->createNew();

$address->setFirstName('Ronald');
$address->setLastName('Weasley');
$address->setCompany('Ministry of Magic');
$address->setCountryCode('UK');
$address->setProvinceCode('UKJ');
$address->setCity('Otter St Catchpole');
$address->setStreet('The Burrow');
$address->setPostcode('000001');
```

Then find a customer to which you would like to assign it, and add the address.

```php
$customer = $this->container->get('sylius.repository.customer')->findOneBy(['email' =>
↪ 'ron.weasley@magic.com']);

$customer->addAddress($address);
```

Remember to flush the customer's manager to save this change.

```php
$this->container->get('sylius.manager.customer')->flush();
```

Learn more

- *Customer & ShopUser Concept Documentation*
- *Addressing - Component Documentation*
- *Addressing - Bundle Documentation*
- *Countries*
- *Zones*
- *Addresses*
- *Address Book*

- *Customer and ShopUser*

- *AdminUser*

- *Addresses*

- *Customer and ShopUser*

- *AdminUser*

- *Addresses*

1.1.6 Products

This is a guide to understanding products handling in Sylius together with surrounding concepts. Read about Associations, Reviews, Attributes, Taxons etc.

Products

This is a guide to understanding products handling in Sylius together with surrounding concepts.

Products

Product model represents unique products in your Sylius store. Every product can have different variations and attributes.

> **Warning:** Each product has to have at least one variant to be sold in the shop.

How to create a Product?

Before we learn how to create products that can be sold, let's see how to create a product without its complex dependencies.

```php
/** @var ProductFactoryInterface $productFactory **/
$productFactory = $this->get('sylius.factory.product');

/** @var ProductInterface $product */
$product = $productFactory->createNew();
```

Creating an empty product is not enough to save it in the database. It needs to have a `name`, a `code` and a `slug`.

```php
$product->setName('T-Shirt');
$product->setCode('00001');
$product->setSlug('t-shirt');

/** @var RepositoryInterface $productRepository */
$productRepository = $this->get('sylius.repository.product');

$productRepository->add($product);
```

After being added via the repository, your product will be in the system. But the customers won't be able to buy it.

Variants

ProductVariant represents a unique kind of product and can have its own pricing configuration, inventory tracking etc.

Variants may be created out of Options of the product, but you are also able to use product variations system without the options at all.

Virtual Product Variants, that do not require shipping

Tip: On the ProductVariant there is a possibility to make a product virtual - by setting its `shippingRequired` property to `false`. In such a way you can have products that will be downloadable or installable for instance.

How to create a Product with a Variant?

You may need to sell product in different Variants - for instance you may need to have books both in hardcover and in paperback. Just like before, use a factory, create the product, save it in the Repository. And then using the ProductVariantFactory create a variant for your product.

```php
/** @var ProductVariantFactoryInterface $productVariantFactory **/
$productVariantFactory = $this->get('sylius.factory.product_variant');

/** @var ProductVariantInterface $productVariant */
$productVariant = $productVariantFactory->createNew();
```

Having created a Variant, provide it with the required attributes and attach it to your Product.

```php
$productVariant->setName('Hardcover');
$productVariant->setCode('1001');
$productVariant->setPosition(1);
$productVariant->setProduct($product);
```

Finally save your Variant in the database using a repository.

```php
/** @var RepositoryInterface $productVariantRepository */
$productVariantRepository = $this->get('sylius.repository.product_variant');

$productVariantRepository->add($variant);
```

Options

In many cases, you will want to have product with different variations. The simplest example would be a piece of clothing, like a T-Shirt available in different sizes and colors or a glass available in different shapes or colors. In order to automatically generate appropriate variants, you need to define options.

Every option type is represented by **ProductOption** and references multiple **ProductOptionValue** entities.

For example you can have two options - Size and Color. Each of them will have their own values.

- **Size**

    - S

- – M
- – L
- – XL
- – XXL

- **Color**

  - – Red
  - – Green
  - – Blue

After defining possible options for a product let's move on to **Variants** which are in fact combinations of options.

How to create a Product with Options and Variants?

Firstly let's learn how to prepare an exemplary Option and its values.

```php
/** @var ProductOptionInterface $option */
$option = $this->get('sylius.factory.product_option')->createNew();
$option->setCode('t_shirt_color');
$option->setName('T-Shirt Color');

// Prepare an array with values for your option, with codes, locale code and option
↪values.
$valuesData = [
    'OV1' => ['locale' => 'en_US', 'value' => 'Red'],
    'OV2' => ['locale' => 'en_US', 'value' => 'Blue'],
    'OV3' => ['locale' => 'en_US', 'value' => 'Green'],
];

foreach ($valuesData as $code => $values) {
    /** @var ProductOptionValueInterface $optionValue */
    $optionValue = $this->get('sylius.factory.product_option_value')->createNew();

    $optionValue->setCode($code);
    $optionValue->setFallbackLocale($values['locale']);
    $optionValue->setCurrentLocale($values['locale']);
    $optionValue->setValue($values['value']);

    $option->addValue($optionValue);
}
```

After you have an Option created and you keep it as `$option` variable let's add it to the Product and generate **Variants**.

```php
// Assuming that you have a basic product let's add the previously created option to
↪it.
$product->addOption($option);

// Having option of a product you can generate variants. Sylius has a service for
↪that operation.
/** @var ProductVariantGeneratorInterface $variantGenerator */
$variantGenerator = $this->get('sylius.generator.product_variant');
```

(continues on next page)

```
$variantGenerator->generate($product);

// And finally add the product, with its newly generated variants to the repository.
/** @var RepositoryInterface $productRepository */
$productRepository = $this->get('sylius.repository.product');

$productRepository->add($product);
```

Learn more:

- *Product - Bundle Documentation*
- *Product - Component Documentation*

Product Reviews

Product Reviews are a marketing tool that let your customers give opinions about the products they buy in your shop.
They have a `rating` and `comment`.

Rating

The rating of a product review is required and must be between 1 and 5.

Product review state machine

When you look inside the `CoreBundle/Resources/config/app/state_machine/`
`sylius_product_review.yml` you will find out that a Review can have 3 different states:

- `new`,
- `accepted`,
- `rejected`

There are only two possible transitions: `accept` (from `new` to `accepted`) and `reject` (from `new` to `rejected`).
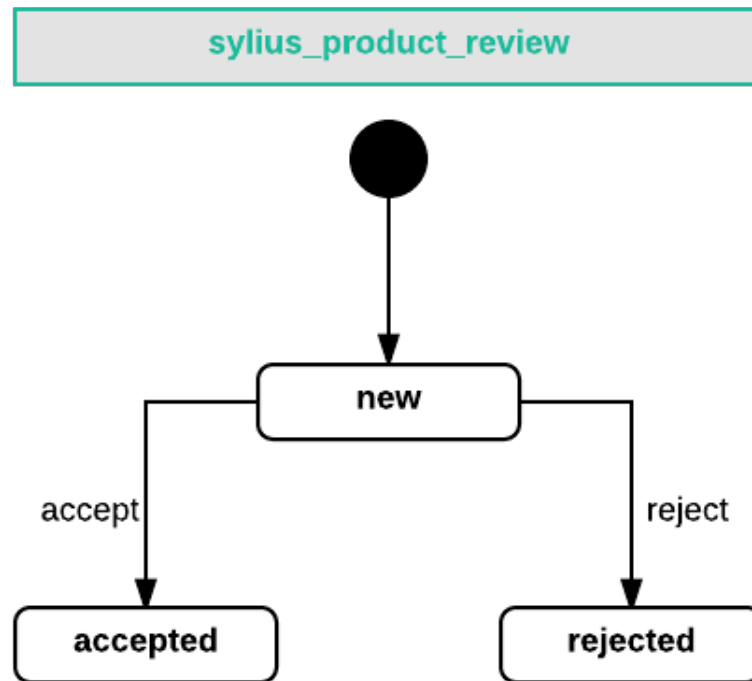
When a review is accepted **the average rating of a product is updated**.

How is the average rating calculated?

The average rating is updated by the AverageRatingUpdater service.

It wraps the AverageRatingCalculator, and uses it inside the `updateFromReview` method.

How to add a ProductReview programmatically?

Create a new review using a factory:

```
/** @var ReviewInterface $review */
$review = $this->container->get('sylius.factory.product_review')->createNew();
```

Fill the content of your review.

```
$review->setTitle('My Review');
$review->setRating(5);
$review->setComment('This product is really great');
```

Then get a customer from the repository, which you would like to make an author of this review.

```
$customer = $this->container->get('sylius.repository.customer')->findOneBy(['email' =>
→ 'john.doe@test.com']);

$review->setAuthor($customer);
```

Remember to set the object that is the subject of your review and then add the review to the repository.

```
$review->setReviewSubject($product);

$this->container->get('sylius.repository.product_review')->add($review);
```

Learn more

- *Product - Bundle Documentation*
- *Product - Component Documentation*

Product Associations

Associations of products can be used as a marketing tool for suggesting your customers, what products to buy together with the one they are currently considering. Associations can increase your shop's efficiency. You choose what strategy you prefer. They are fully configurable.

Association Types

The type of an association can be different. If you sell food you can suggest inspiring ingredients, if you sell products for automotive you can suggest buying some tools that may be useful for a home car mechanic. Exemplary association types can be: `up-sell`, `cross-sell`, `accessories`, `alternatives` and whatever you imagine.

How to create a new Association Type?

Create a new Association Type using a dedicated factory. Give the association a `code` and a `name` to easily recognize the type.

```
/** @var ProductAssociationTypeInterface $associationType */
$associationType = $this->container->get('sylius.factory.product_association_type')->
↪createNew();

$associationType->setCode('accessories');
$associationType->setName('Accessories');
```

To have the new association type in the system add it to the repository.

```
$this->container->get('sylius.repository.product_association_type')->add(
↪$associationType);
```

How to add a new Association to a Product?

Find in your system a product to which you would like to add an association. We will use a Go Pro camera as an example.

```
$product = $this->container->get('sylius.repository.product')->findOneBy(['code' =>
↪'go-pro-camera']);
```

Next create a new Association which will connect our camera with its accessories. Such an association needs the AssociationType we have created in the previous step above.

```
/** @var ProductAssociationInterface $association */
$association = $this->container->get('sylius.factory.product_association')->
↪createNew();

/** @var ProductAssociationTypeInterface $associationType */
$associationType = $this->container->get('sylius.repository.product_association_type
↪')->findOneBy(['code' => 'accessories']);

$association->setType($associationType);
```

Let's add all products from a certain taxon to the association we have created. To do that find a desired taxon by code and get all its products. Perfect accessories for a camera will be SD cards.

```
/** @var TaxonInterface $taxon */
$taxon = $this->container->get('sylius.repository.taxon')->findOneBy(['code' => 'sd-
↪cards']);

$associatedProducts = $taxon->getProducts();
```

Having a collection of products from the SD cards taxon iterate over them and add them one by one to the association.

```
foreach ($associatedProducts as $associatedProduct) {
    $association->addAssociatedProduct($associatedProduct);
}
```

Finally add the created association with SD cards to our Go Pro camera product.

```
$product->addAssociation($association);
```

And to save everything in the database you need to add the created association to the repository.

```
$this->container->get('sylius.repository.product_association')->add($association);
```

Learn more:

- *Product - Concept Documentation*

Attributes

Attributes in Sylius are used to describe traits shared among entities. The best example are products, that may be of the same category and therefore they will have many similar attributes such as **number of pages for a book**, **brand of a T-shirt** or simply **details of any product**.

Attribute

The **Attribute** model has a translatable name (like for instance `Book pages`), code (`book_pages`) and type (`integer`). There are a few available types of an Attribute:

- text (*default*)
- checkbox
- integer

- percent

- textarea

- date

- datetime

What these types may be useful for?

- text - brand of a T-Shirt

- checkbox - show whether a T-Shirt is made of cotton or not

- integer - number of elements when a product is a set of items.

- percent - show how much cotton is there in a piece of clothing

- textarea - display more detailed information about a product

- date - release date of a movie

- datetime - accurate date and time of an event

How to create an Attribute?

To give you a better insight into Attributes, let's have a look how to prepare and add an Attribute with a Product to the system programatically.

To assign Attributes to Products firstly you will need a factory for ProductAttributes. The AttributeFactory has a special method createTyped($type), where $type is a string.

The Attribute needs a `code` and a `name` before it can be saved in the repository.

```
/** @var AttributeFactoryInterface $attributeFactory */
$attributeFactory = $this->container->get('sylius.factory.product_attribute');

/** @var AttributeInterface $attribute */
$attribute = $attributeFactory->createTyped('text');

$attribute->setName('Book cover');
$attribute->setCode('book_cover');

$this->container->get('sylius.repository.product_attribute')->add($attribute);
```

In order to assign value to your Attribute you will need a factory of ProductAttributeValues, use it to create a new value object.

```
/** @var FactoryInterface $attributeValueFactory */
$attributeValueFactory = $this->container->get('sylius.factory.product_attribute_value
↪');

/** @var AttributeValueInterface $hardcover */
$hardcover = $attributeValueFactory->createNew();
```

Attach the new AttributeValue to your Attribute and set its `value`, which is what will be rendered in frontend.

```
$hardcover->setAttribute($attribute);

$hardcover->setValue('hardcover');
```

Finally let's find a product that will have your newly created attribute.

```
/** @var ProductInterface $product */
$product = $this->container->get('sylius.repository.product')->findOneBy(['code' =>
↪'code']);

$product->addAttribute($hardcover);
```

Now let's see what has to be done if you would like to add an attribute of `integer` type. Let's find such a one in the repository, it will be for example the `BOOK-PAGES` attribute.

```
/** @var AttributeInterface $bookPagesAttribute */
$bookPagesAttribute = $this->container->get('sylius.repository.product_attribute')->
↪findOneBy(['code' => 'BOOK-PAGES']);

/** @var AttributeValueInterface $pages */
$pages = $attributeValueFactory->createNew();

$pages->setAttribute($bookPagesAttribute);

$pages->setValue(500);

$product->addAttribute($pages);
```

After adding attributes remember to **flush the product manager**.

```
$this->container->get('sylius.manager.product')->flush();
```

Your Product will now have two Attributes.

Learn more

- *Attribute - Component Documentation*

Pricing

Pricing is a part of Sylius responsible for providing the product prices per channel.

Note: All prices in Sylius are saved in the **base currency** of each channel separately.

Currency per Channel

As you already know Sylius operates on *Channels*.

Each channel has a **base currency** in which all prices are saved.

Note: Whenever you operate on concepts that have specified values per channel (like *ProductVariant's price*, *Promotion's fixed discount* etc.)

Exchange Rates

Each currency defined in the system should have an ExchangeRate configured.

ExchangeRate is a separate entity that holds a relation between two currencies and specifies their exchange rate.

Exchange rates are used for viewing the *approximate* price in a currency different from the base currency of a channel.

Learn more

- *Currency - Component Documentation*
- *Currencies Concept Documentation*

Taxons

We understand Taxons in Sylius as you would normally define categories. Sylius gives you a possibility to categorize your products in a very flexible way, which is one of the most vital functionalities of the modern e-commerce systems. The Taxons system in Sylius works in a hierarchical way. Let's see exemplary categories trees:

```
Category
 |
 |\__ Clothes
 |        \_ T-Shirts
 |         \_ Shirts
 |          \_ Dresses
 |           \_ Shoes
 |
 \__ Books
         \_ Fantasy
          \_ Romance
           \_ Adventure
            \_ Other

Gender
 |
 \_ Male
  \_ Female
```

How to create a Taxon?

As always with Sylius resources, to create a new object you need a factory. If you want to create a single, not nested category:

```php
/** @var FactoryInterface $taxonFactory */
$taxonFactory = $this->get('sylius.factory.taxon');

/** @var TaxonInterface $taxon */
$taxon = $taxonFactory->createNew();

$taxon->setCode('category');
$taxon->setName('Category');
```

But if you want to have a tree of categories, create another taxon and add it as a **child** to the previously created one.

```
/** @var TaxonInterface $childTaxon */
$childTaxon = $taxonFactory->createNew();

$childTaxon->setCode('clothes');
$childTaxon->setName('Clothes');

$taxon->addChild($childTaxon);
```

Finally **the parent taxon** has to be added to the system using a repository, all its child taxons will be added with it.

```
/** @var TaxonRepositoryInterface $taxonRepository */
$taxonRepository = $this->get('sylius.repository.taxon');

$taxonRepository->add($taxon);
```

How to assign a Taxon to a Product?

In order to categorize products you will need to assign your taxons to them - via the `addProductTaxon()` method on Product.

```
/** @var ProductInterface $product */
$product = $this->container->get('sylius.factory.product')->createNew();
$product->setCode('product_test');
$product->setName('Test');

/** @var TaxonInterface $taxon */
$taxon = $this->container->get('sylius.factory.taxon')->createNew();
$taxon->setCode('food');
$taxon->setName('Food');

/** @var RepositoryInterface $taxonRepository */
$taxonRepository = $this->container->get('sylius.repository.taxon');
$taxonRepository->add($taxon);


/** @var ProductTaxonInterface $productTaxon */
$productTaxon = $this->container->get('sylius.factory.product_taxon')->createNew();
$productTaxon->setTaxon($taxon);
$productTaxon->setProduct($product);

$product->addProductTaxon($productTaxon);

/** @var EntityManagerInterface $productManager */
$productManager = $this->container->get('sylius.manager.product');

$productManager->persist($product);
$productManager->flush();
```

What is the mainTaxon of a Product?

The product entity in Sylius core has a field `mainTaxon`. This field is used, for instance, for breadcrumbs generation. But you can also use it for your own logic, like for instance links generation.

To set it on your product you need to use the `setMainTaxon()` method.

Learn more

- *Taxonomy - Bundle Documentation*
- *taxonomy - Component Documentation*

Inventory

Sylius leverages a very simple approach to inventory management. The current stock of an item is stored on the **ProductVariant** entity as the `onHand` value.

InventoryUnit

InventoryUnit has a relation to a Stockable on it, in case of Sylius Core it will be a relation to the **ProductVariant** that implements the StockableInterface on the **OrderItemUnit** that implements the InventoryUnitInterface.

It represents a physical unit of the product variant that is in the shop.

Inventory On Hold

Putting inventory items `onHold` is a way of reserving them before the customer pays for the order. Items are put on hold when the checkout is completed.

Tip: Putting items `onHold` does not remove them from `onHand` yet. If a customer buys 2 tracked items out of 5 being in the inventory (`5 onHand`), after the checkout there will be `5 onHand` and `2 onHold`.

Availability Checker

There is a service that will help you check the availability of items in the inventory - AvailabilityChecker.

It has two methods `isStockAvailable` (is there at least one item available) and `isStockSufficient` (is there a given amount of items available).

Tip: There are two respective twig functions for checking inventory: `sylius_inventory_is_available` and `sylius_inventory_is_sufficient`.

OrderInventoryOperator

Inventory Operator is the service responsible for managing the stock amounts of every *ProductVariant* on an Order with the following methods:

- `hold` - is called when the order's checkout is completed, it puts the inventory units onHold, while still not removing them from onHand,
- `sell` - is called when the order's payment are assigned with the state `paid`. The inventory items are then removed from onHold and onHand,
- `release` - is a way of making onHold items of an order back to only onHand,

---

- `giveBack` - is a way of returning sold items back to the inventory onHand,

- `cancel` - this method works both when the order is paid and unpaid. It uses both `giveBack` and `release` methods.

How does Inventory work on examples?

Tip: You can see all use cases we have designed in Sylius in our Behat scenarios for inventory.

Learn more

- *Order concept documentation*
- *Inventory Bundle documentation*
- *Inventory Component documentation*

Search

Having a products search functionality in an eCommerce system is a very popular use case. Sylius provides a products search functionality that is a grid filter.

Grid filters

For simple use cases of products search use the **filters of grids**. For example, the shop's categories each have a `search` filter in the products grid:

```yaml
# Sylius/Bundle/ShopBundle/Resources/config/grids/product.yml
filters:
    search:
        type: string
        label: false
        options:
            fields: [translation.name]
        form_options:
            type: contains
```

It searches by product names that contains a string that the user typed in the search bar.

The search bar looks like below:
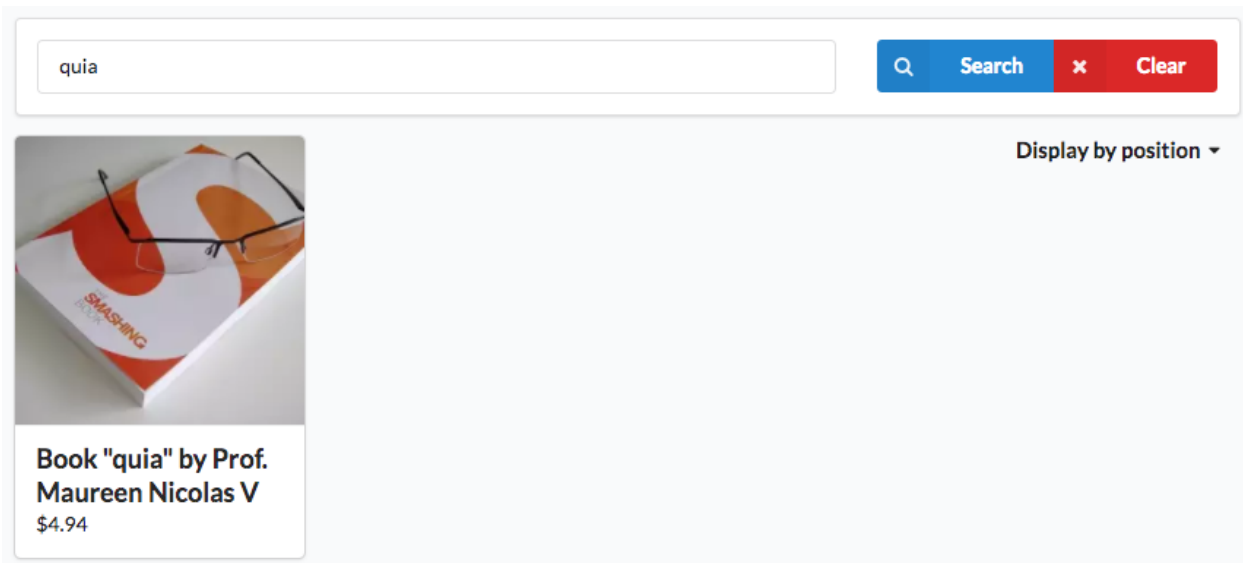
Customizing search filter

The search bar in many shops should be more sophisticated, than just a simple text search. You may need to add searching by price, reviews, sizes or colors.

If you would like to extend this built-in functionality read *this article about grids customizations*, and *the GridBundle docs*.

ElasticSearch

When the grids filtering is not enough for you, and your needs are more complex you should go for the ElasticSearch integration.

There is the Sylius/SyliusElasticSearchPlugin integration extension, which you can use to extend Sylius functionalities with ElasticSearch.

All you have to do is require the plugin in your project via composer, install the ElasticSearch server, and configure ElasticSearch in your application. Everything is well described in the Sylius/SyliusElasticSearchPlugin's readme.

Learn more

- SyliusElasticSearchPlugin
- *Grid Bundle documentation*
- *Grid Component documentation*
- *Products*
- *Product Reviews*
- *Product Associations*
- *Attributes*
- *Pricing*
- *Taxons*

1.1.7 Carts & Orders

In this chapter you will learn everything you need to know about orders in Sylius. This concept comes together with a few additional ones, like promotions, payments, shipments or checkout in general.

You should also have a look here if you are looking for Cart, which is in Sylius an Order in the `cart` state.

Carts & Orders

In this chapter you will learn everything you need to know about orders in Sylius. This concept comes together with a few additional ones, like promotions, payments, shipments or checkout in general.

Warning: **Cart** in Sylius is in fact an Order in the state `cart`.

Orders

Order model is one of the most important in Sylius, where many concepts of e-commerce meet. It represents an order that can be either placed or in progress (cart).

Order holds a collection of **OrderItem** instances, which represent products from the shop, as its physical copies, with chosen variants and quantities.

Each Order is **assigned to the channel** in which it has been created as well as the **language** the customer was using while placing the order. The order currency code will be the base currency of the current channel by default.

How to create an Order programmatically?

To programmatically create an Order you will of course need a factory.

```php
/** @var FactoryInterface $order */
$orderFactory = $this->container->get('sylius.factory.order');

/** @var OrderInterface $order */
$order = $orderFactory->createNew();
```

Then get a channel to which you would like to add your Order. You can get it from the context or from the repository by code for example.

```
/** @var ChannelInterface $channel */
$channel = $this->container->get('sylius.context.channel')->getChannel();

$order->setChannel($channel);
```

Next give your order a locale code.

```
/** @var string $localeCode */
$localeCode = $this->container->get('sylius.context.locale')->getLocaleCode();

$order->setLocaleCode($localeCode);
```

And a currency code:

```
$currencyCode = $this->container->get('sylius.context.currency')->getCurrencyCode();

$order->setCurrencyCode($currencyCode);
```

What is more the proper Order instance should also have the **Customer** assigned. You can get it from the repository by email.

```
/** @var CustomerInterface $customer */
$customer = $this->container->get('sylius.repository.customer')->findOneBy(['email' =>
→ 'shop@example.com']);

$order->setCustomer($customer);
```

A very important part of creating an Order is adding **OrderItems** to it. Assuming that you have a **Product** with a **ProductVariant** assigned already in the system:

```
/** @var ProductVariantInterface $variant */
$variant = $this->container->get('sylius.repository.product_variant')->findOneBy([]);

// Instead of getting a specific variant from the repository
// you can get the first variant of off a product by using $product->getVariants()->
→first()
// or use the **VariantResolver** service - either the default one or your own.
// The default product variant resolver is available at id - 'sylius.product_variant_
→resolver.default'

/** @var OrderItemInterface $orderItem */
$orderItem = $this->container->get('sylius.factory.order_item')->createNew();
$orderItem->setVariant($variant);
```

In order to change the amount of items use the **OrderItemQuantityModifier**.

```
$this->container->get('sylius.order_item_quantity_modifier')->modify($orderItem, 3);
```

Add the item to the order. And then call the **CompositeOrderProcessor** on the order to have everything recalculated.

```
$order->addItem($orderItem);

$this->container->get('sylius.order_processing.order_processor')->process($order);
```

Finally you have to save your order using the repository.

```
/** @var OrderRepositoryInterface $orderRepository */
$orderRepository = $this->container->get('sylius.repository.order');

$orderRepository->add($order);
```

The Order State Machine

Order has also its own state, which can have the following values:

- `cart` - before the checkout is completed, it is the initial state of an Order,
- `new` - when checkout is completed the cart is transformed into a `new` order,
- `fulfilled` - when the order payments and shipments are completed,
- `cancelled` - when the order was cancelled.
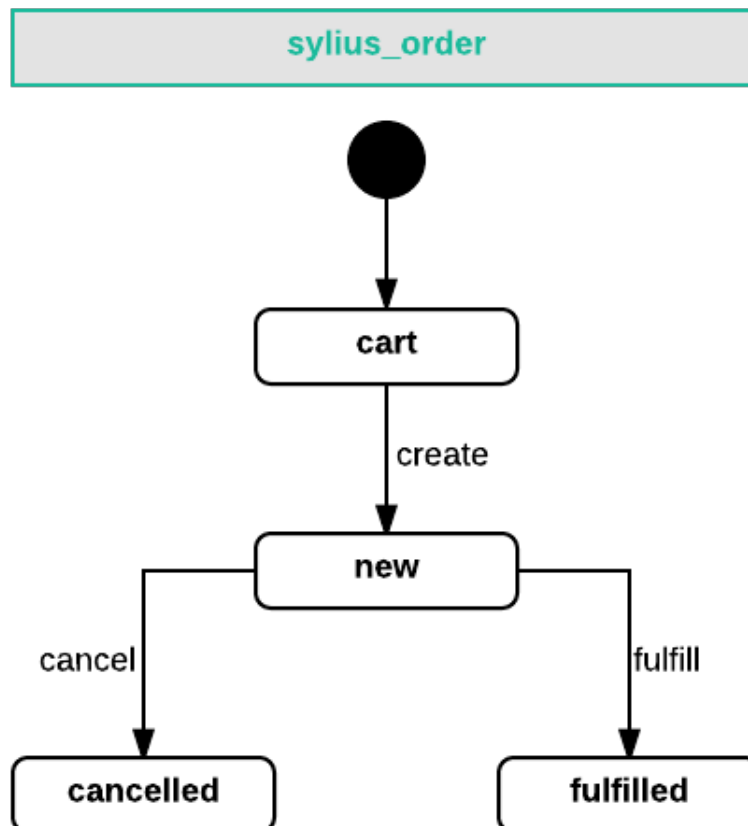
Tip: The state machine of order is an obvious extension to the *state machine of checkout*.

Shipments of an Order

An **Order** in Sylius holds a collection of Shipments on it. Each shipment in that collection has its own shipping method and has its own state machine. This lets you to divide an order into several different shipments that have own shipping states (like sending physical objects via DHL and sending a link to downloadable files via e-mail).

Tip: If you are not familiar with the shipments concept *check the documentation*.

State machine of Shipping in an Order
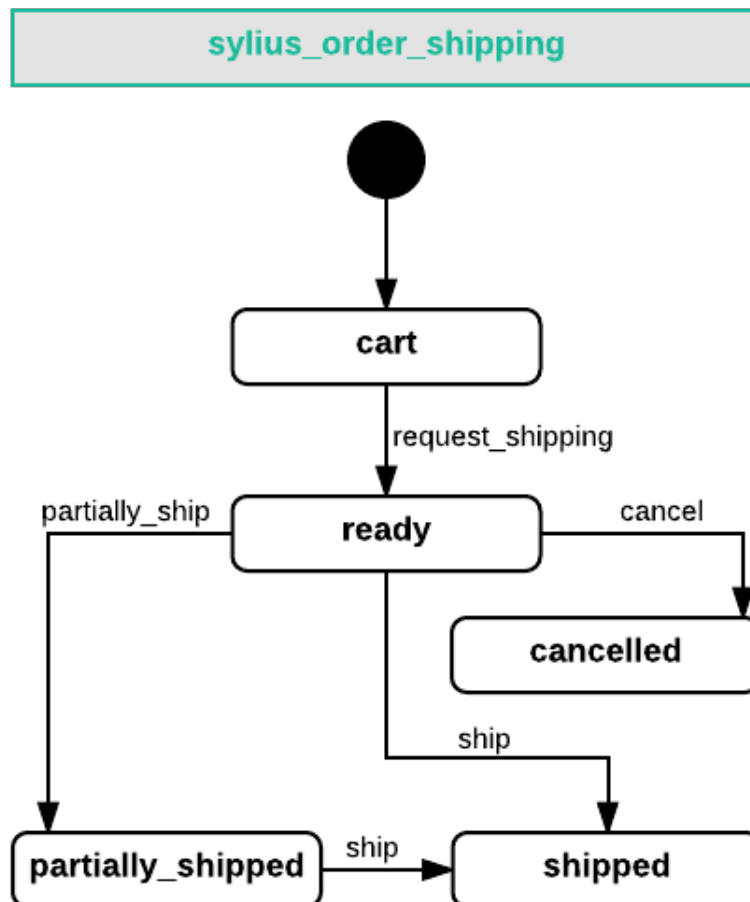
How to add a Shipment to an Order?

You will need to create a shipment, give it a desired shipping method and add it to the order. Remember to process the order using order processor and then flush the order manager.

```
/** @var ShipmentInterface $shipment */
$shipment = $this->container->get('sylius.factory.shipment')->createNew();
```

```
$shipment->setMethod($this->container->get('sylius.repository.shipping_method')->
↪findOneBy(['code' => 'UPS']));

$order->addShipment($shipment);

$this->container->get('sylius.order_processing.order_processor')->process($order);
$this->container->get('sylius.manager.order')->flush();
```

Shipping costs of an Order

Shipping costs of an order are stored as Adjustments. When a new shipment is added to a cart the order processor assigns a shipping adjustment to the order that holds the cost.

Shipping a Shipment with a state machine transition

Just like in every state machine you can execute its transitions manually. To **ship** a shipment of an order you have to apply two transitions `request_shipping` and `ship`.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderShippingTransitions::GRAPH);
$stateMachine->apply(OrderShippingTransitions::TRANSITION_REQUEST_SHIPPING);
$stateMachine->apply(OrderShippingTransitions::TRANSITION_SHIP);

$this->container->get('sylius.manager.order')->flush();
```

After that the `shippingState` of your order will be `shipped`.

Payments of an Order

An **Order** in Sylius holds a collection of Payments on it. Each payment in that collection has its own payment method and has its own payment state. It lets you to divide paying for an order into several different methods that have own payment states.

Tip: If you are not familiar with the Payments concept *check the documentation*.

State machine of Payment in an Order
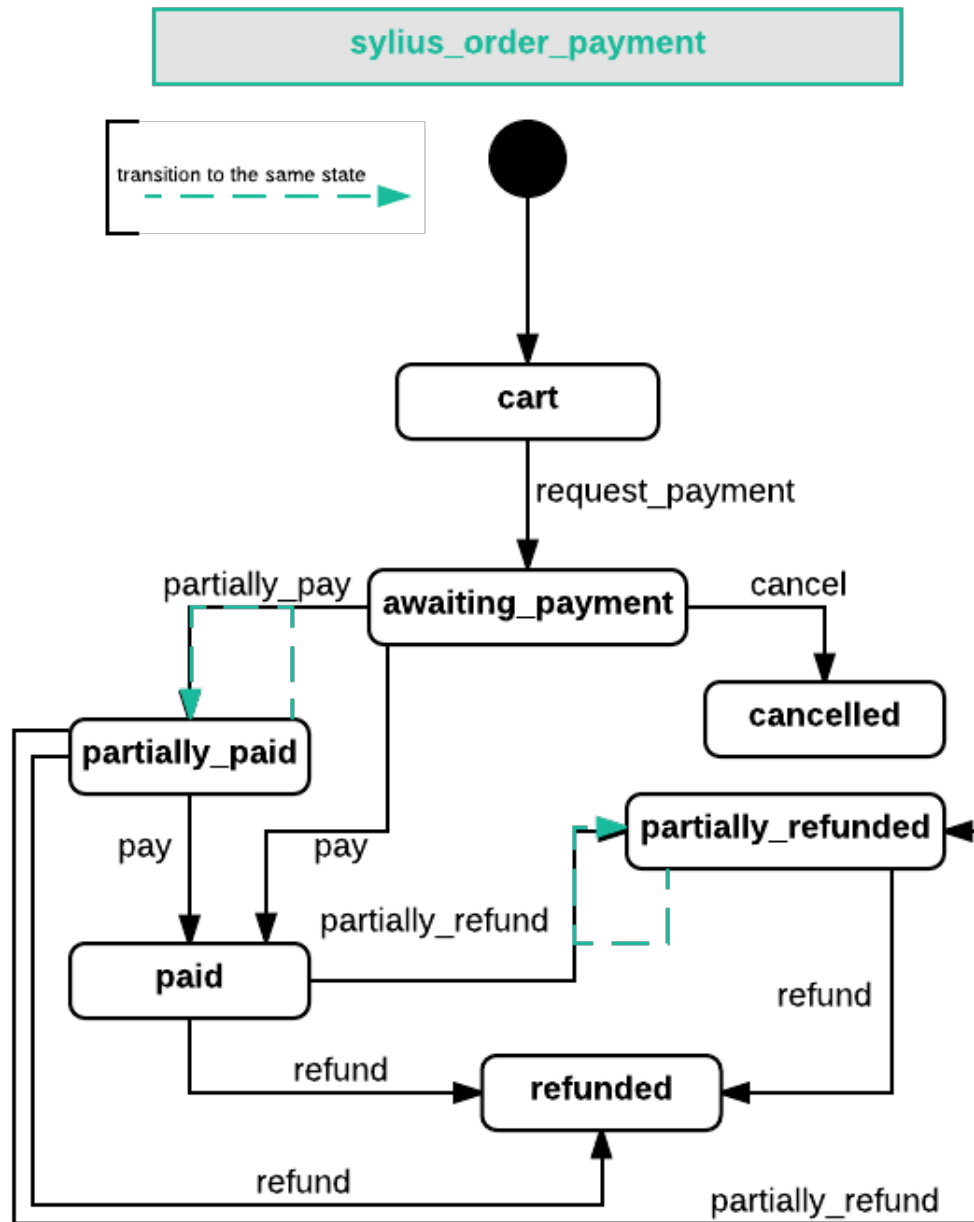
How to add a Payment to an Order?

You will need to create a payment, give it a desired payment method and add it to the order. Remember to process the order using order processor and then flush the order manager.

```
/** @var PaymentInterface $payment */
$payment = $this->container->get('sylius.factory.payment')->createNew();
```

```
$payment->setMethod($this->container->get('sylius.repository.payment_method')->
→findOneBy(['code' => 'offline']));

$payment->setCurrencyCode($currencyCode);

$order->addPayment($payment);
```

Completing a Payment with a state machine transition

Just like in every state machine you can execute its transitions manually. To **pay** for a payment of an order you have to apply two transitions `request_payment` and `pay`.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderPaymentTransitions::GRAPH);
$stateMachine->apply(OrderPaymentTransitions::TRANSITION_REQUEST_PAYMENT);
$stateMachine->apply(OrderPaymentTransitions::TRANSITION_PAY);

$this->container->get('sylius.manager.order')->flush();
```

If it was the only payment assigned to that order now the `paymentState` of your order will be `paid`.

Learn more

- *Order - Component Documentation*
- *Order - Bundle Documentation*

Taxation

Sylius' taxation system allows you to apply appropriate taxes for different items, billing zones and using custom calculators.

Tax Categories

In order to process taxes in your store, you need to configure at least one **TaxCategory**, which represents a specific type of merchandise. If all your items are taxed with the same rate, you can have a simple "Taxable Goods" category assigned to all items.

If you sell various products and some of them have different taxes applicable, you could create multiple categories. For example, "Clothing", "Books" and "Food".

Additionally to tax categories, you can have different zones, in order to apply correct taxes for customers coming from any country in the world.

How to create a TaxCategory programmatically?

In order to create a TaxCategory use the dedicated factory. Your TaxCategory requires a `name` and a `code`.

```
/** @var TaxCategoryInterface $taxCategory */
$taxCategory = $this->container->get('sylius.factory.tax_category')->createNew();

$taxCategory->setCode('taxable_goods');
$taxCategory->setName('Taxable Goods');

$this->container->get('sylius.repository.tax_category')->add($taxCategory);
```

Since now you will have a new TaxCategory available.

How to set a TaxCategory on a ProductVariant?

In order to have taxes calculated for your products you have to set TaxCategories for each ProductVariant you create. Read more about Products and Variants *here*.

```
/** @var TaxCategoryInterface $taxCategory */
$taxCategory = $this->container->get('sylius.repository.tax_category')->findOneBy([
→'code' => 'taxable_goods']);

/** @var ProductVariantInterface $variant */
$variant = $this>container->get('sylius.repository.product_variant')->findOneBy(['code
→' => 'mug']);

$variant->setTaxCategory($taxCategory);
```

Tax Rates

A tax rate is essentially a percentage amount charged based on the sales price. Tax rates also contain other important information:

- Whether product prices are inclusive of this tax
- The zone in which the order address must fall within
- The tax category that a product must belong to in order to be considered taxable
- Calculator to use for computing the tax

TaxRates included in price

The **TaxRate** entity has a field for configuring if you would like to have taxes included in the price of a subject or not.

If you have a TaxCategory with a 23% VAT TaxRate *includedInPrice* ($taxRate->isIncludedInPrice() returns true), then the price shown on the ProductVariant in that TaxCategory will be increased by 23% all the time. See the Behat scenario below:

```
Given the store has included in price "VAT" tax rate of 23%
And the store has a product "T-Shirt" priced at "$10.00"
When I add product "T-Shirt" to my cart
Then my cart total should be "$10.00"
And my cart taxes should be "$1.87"
```

If the TaxRate *will not be included* ($taxRate->isIncludedInPrice() returns false) then the price of ProductVariant will be shown without taxes, but when this ProductVariant will be added to cart taxes will be shown in the Taxes Total in the cart. See the Behat scenario below:

```
Given the store has excluded from price "VAT" tax rate of 23%
And the store has a product "T-Shirt" priced at "$10.00"
When I add product "T-Shirt" to my cart
Then my cart total should be "$12.30"
And my cart taxes should be "$2.30"
```

How to create a TaxRate programmatically?

Note: Before creating a tax rate you need to know that you can have different tax zones, in order to apply correct taxes for customers coming from any country in the world. To understand how zones work, please refer to the Zones chapter of this book.

Use a factory to create a new, empty TaxRate. Provide a `code`, a `name`. Set the amount of charge in float. Then choose a calculator and zone (retrieved from the repository beforehand).

Finally you can set the TaxCategory of your new TaxRate.

```
/** @var TaxRateInterface $taxRate */
$taxRate = $this->container->get('sylius.factory.tax_rate')->createNew();

$taxRate->setCode('7%');
$taxRate->setName('7%');
$taxRate->setAmount(0.07);
$taxRate->setCalculator('default');

// Get a Zone from the repository, for example the 'US' zone
/** @var ZoneInterface $zone */
$zone = $this->container->get('sylius.repository.zone')->findOneBy(['code' => 'US']);

$taxRate->setZone($zone);

// Get a TaxCategory from the repository, for example the 'alcohol' category
/** @var TaxCategoryInterface $taxCategory */
$taxCategory = $this->container->get('sylius.repository.tax_category')->findOneBy([
    'code' => 'alcohol']);

$taxRate->setCategory($taxCategory);

$this->container->get('sylius.repository.tax_rate')->add($taxRate);
```

Default Tax Zone

The **default tax zone** concept is used for situations when we want to show taxes included in price even when we do not know the address of the Customer, therefore we cannot choose a proper Zone, which will have proper TaxRates.

Since we are using the concept of *Channels*, we will use **the Zone assigned to the Channel as default Zone for Taxation**.

Note: To understand how zones work, please refer to the Zones chapter of this book.

Applying Taxes

For applying Taxes **Sylius** is using the OrderTaxesProcessor, which has the services that implement the OrderTaxesApplicatorInterface inside.

Calculators

For calculating Taxes **Sylius** is using the DefaultCalculator. You can create your custom calculator for taxes by creating a class that implements the CalculatorInterface and registering it as a `sylius.tax_calculator.your_calculator_name` service.

Learn more

- *Taxation - Bundle Documentation*
- *taxation - Component Documentation*

Adjustments

Adjustment is a resource closely connected to the *Orders' concept*. It **influences the order's total**.

Adjustments may appear on the Order, the OrderItems and the OrderItemUnits.

There are a few types of adjustments in Sylius:

- Order Promotion Adjustments,
- OrderItem Promotion Adjustments,
- OrderItemUnit Promotion Adjustments,
- Shipping Adjustments,
- Shipping Promotion Adjustments,
- Tax Adjustments

And they can be generally divided into three *groups*: **promotion adjustments**, **shipping adjustments** and **taxes adjustments**.

Also note that adjustments can be either **positive**: charges (with a +) or **negative**: discounts (with a −).

How to create an Adjustment programmatically?

The Adjustments alone are a bit useless. They should be created alongside Orders.

As usual, get a factory and create an adjustment. Then give it a `type` - you can find all the available types on the AdjustmentInterface. The adjustment needs also the `amount` - which is the amount of money that will be **added to the orders total**.

Note: The `amount` is always saved in the **base currency**.

Additionally you can set the `label` that will be displayed on the order view and whether your adjustment is `neutral` - **neutral adjustments** do not affect the order's total (like for example taxes included in price).

```
/** @var AdjustmentInterface $adjustment */
$adjustment = $this->container->get('sylius.factory.adjustment')->createNew();

$adjustment->setType(AdjustmentInterface::ORDER_PROMOTION_ADJUSTMENT);
$adjustment->setAmount(200);
$adjustment->setNeutral(false);
$adjustment->setLabel('Test Promotion Adjustment');

$order->addAdjustment($adjustment);
```

Note: Remember that if you are creating OrderItem adjustments you have to add them on the OrderItem level. The same happens with the OrderItemUnit adjustments, which have to be added on the OrderItemUnit level.

To see changes on the order you need to update it in the database.

```
$this->container->get('sylius.manager.order')->flush();
```

Learn more

- *Promotions - Concept Documentation*
- *Taxation - Concept Documentation*
- *Shipments - Concept Documentation*

Promotions

The system of **Promotions** in **Sylius** is really flexible. It is a combination of promotion rules and actions.

Promotions have a few parameters - a unique `code`, `name`, `usageLimit`, the period of time when it works. There is a possibility to define **exclusive promotions** (no other can be applied if an exclusive promotion was applied) and **priority** that is useful for them, because the exclusive promotion should get the top priority.

Tip: The `usageLimit` of a promotion is the **total number of times this promotion can be used**.

Tip: **Promotion priorities** are numbers that you assign to the promotion. The larger the number, the higher the priority. So a promotion with priority 3 would be applied before a promotion with priority set to 1.

What can you use the priority for? Well, imagine that you have two different promotions, one's action is to give 10% discount on whole order and the other one gives 5$ discount from the order total. Business (and money) wise, which one should we apply first? ;)

How to create a Promotion programmatically?

Just as usual, use a factory. The promotion needs a `code` and a `name`.

```
/** @var PromotionInterface $promotion */
$promotion = $this->container->get('sylius.factory.promotion')->createNew();

$promotion->setCode('simple_promotion_1');
$promotion->setName('Simple Promotion');
```

Of course an empty promotion would be useless - it is just a base for adding **Rules** and **Actions**. Let's see how to make it functional.

Promotion Rules

The promotion **Rules** restrict in what circumstances a promotion will be applied. An appropriate **RuleChecker** (each Rule type has its own RuleChecker) may check if the Order:

- Contains a number of items from a specified taxon (for example: *contains 4 products that are categorized as t-shirts*)

- Has a specified total price of items from a given taxon (for example: *all mugs in the order cost 20$ in total*)

- Has total price of at least a defined value (for example: *the orders' items total price is equal at least 50$*)

And many more similar, suitable to your needs.

Rule Types

The types of rules that are configured in **Sylius** by default are:

- **Cart Quantity** - checks if there is a given amount of items in the cart,

- **Item Total** - checks if items in the cart cost a given amount of money,

- **Taxon** - checks if there is at least one item from given taxons in the cart,

- **Items From Taxon Total** - checks in the cart if items from a given taxon cost a given amount of money,

- **Nth Order** - checks if this is for example the second order made by the customer,

- **Shipping Country** - checks if the order's shipping address is in a given country.

How to create a new PromotionRule programmatically?

Creating a **PromotionRule** is really simple since we have the PromotionRuleFactory. It has dedicated methods for creating all types of rules available by default.

In the example you can see how to create a simple Cart Quantity rule. It will check if there are at least 5 items in the cart.

```
/** @var PromotionRuleFactoryInterface $ruleFactory */
$ruleFactory = $this->container->get('sylius.factory.promotion_rule');

$quantityRule = $ruleFactory->createCartQuantity('5');

// add your rule to the previously created Promotion
$promotion->addRule($quantityRule);
```

Note: **Rules** are just constraints that have to be fulfilled by an order to make the promotion **eligible**. To make something happen to the order you will need **Actions**.

PromotionRules configuration reference

Each PromotionRule type has a very specific structure of its configuration array:

PromotionRule type	Rule Configuration Array
`cart_quantity`	`['count' => $count]`
`item_total`	`[$channelCode => ['amount' => $amount]]`
`has_taxon`	`['taxons' => $taxons]`
`total_of_items_from_taxon`	`[$channelCode => ['taxon' => $taxonCode, 'amount' => $amount]]`
`nth_order`	`['nth' => $nth]`
`contains_product`	`['product_code' => $productCode]`

Promotion Actions

Promotion Action is basically what happens when the rules of a Promotion are fulfilled, what discount is applied on the whole Order (or its Shipping cost).

There are a few kinds of actions in **Sylius**:

- fixed discount on the order (for example: -5$ off the order total)

- percentage discount on the order (for example: -10% on the whole order)

- fixed unit discount (for example: -1$ off the order total but *distributed and applied on each order item unit*)

- percentage unit discount (for example: -10% off the order total but *distributed and applied on each order item unit*)

- add product (for example: gives a free bonus sticker)

- shipping discount (for example: -6$ on the costs of shipping)

Tip: Actions are applied on all items in the Order. If you are willing to apply discounts on specific items in the order check Filters at the bottom of this article.

How to create an PromotionAction programmatically?

In order to create a new PromotionAction we can use the dedicated PromotionActionFactory.

It has special methods for creating all types of actions available by default. In the example below you can how to create a simple Fixed Discount action, that reduces the total of an order by 10$.

```
/** @var PromotionActionFactoryInterface $actionFactory */
$actionFactory = $this->container->get('sylius.factory.promotion_action');

$action = $actionFactory->createFixedDiscount(10);
```

```
// add your action to the previously created Promotion
$promotion->addAction($action);
```

Note: All **Actions** are assigned to a Promotion and are executed while the Promotion is applied. This happens via the CompositeOrderProcessor service. See details of **applying Promotions** below.

And finally after you have an **PromotionAction** and a **PromotionRule** assigned to the **Promotion** add it to the repository.

```
$this->container->get('sylius.repository.promotion')->add($promotion);
```

PromotionActions configuration reference

Each PromotionAction type has a very specific structure of its configuration array:

PromotionAction type	Action Configuration Array
order_fixed_discount	[$channelCode => ['amount' => $amount]]
unit_fixed_discount	[$channelCode => ['amount' => $amount]]
order_percentage_discount	['percentage' => $percentage]
unit_percentage_discount	[$channelCode => ['percentage' => $percentage]]
shipping_percentage_discount	['percentage' => $percentage]

Applying Promotions

Promotions in Sylius are handled by the PromotionProcessor which inside uses the PromotionApplicator.

The **PromotionProcessor**'s method `process()` is executed on the subject of promotions - an Order:

- firstly it iterates over the promotions of a given Order and first **reverts** them all,
- then it checks the eligibility of all promotions available in the system on the given Order
- and finally it applies all the eligible promotions to that order.

How to apply a Promotion manually?

Let's assume that you would like to **apply a 10% discount on everything** somewhere in your code.

To achieve that, create a Promotion with an PromotionAction that gives 10% discount. You don't need rules.

```
/** @var PromotionInterface $promotion */
$promotion = $this->container->get('sylius.factory.promotion')->createNew();

$promotion->setCode('discount_10%');
$promotion->setName('10% discount');

/** @var PromotionActionFactoryInterface $actionFactory */
$actionFactory = $this->container->get('sylius.factory.promotion_action');
```

```
$action = $actionFactory->createPercentageDiscount(10);

$promotion->addAction($action);

$this->container->get('sylius.repository.promotion')->add($promotion);

// and now get the PromotionApplicator and use it on an Order (assuming that you have
→one)
$this->container->get('sylius.promotion_applicator')->apply($order, $promotion);
```

Promotion Filters

Filters are really handy when you want to apply promotion's actions to groups of products in an Order. For example if you would like to apply actions only on products from a desired taxon - use the available by default TaxonFilter.

Read these scenarios regarding promotion filters to have a better understanding of them.

Learn more

- *Promotion - Component Documentation*
- *Promotion - Bundle Documentation*
- *How to create a custom promotion rule?*
- *How to create a custom promotion action?*

Coupons

The concept of coupons is closely connected to the *Promotions Concept*.

Coupon Parameters

A **Coupon** besides a code has a date when it expires, the usageLimit and it counts how many times it was already used.

How to create a coupon with a promotion programmatically?

> **Warning:** The promotion has to be couponBased = true in order to be able to hold a collection of Coupons that belong to it.

Let's create a promotion that will have a single coupon that activates the free shipping promotion.

```
/** @var PromotionInterface $promotion */
$promotion = $this->container->get('sylius.factory.promotion')->createNew();

$promotion->setCode('free_shipping');
$promotion->setName('Free Shipping');
```

Remember to set a **channel** for your promotion and to make it **couponBased**!

```
$promotion->addChannel($this->container->get('sylius.repository.channel')->findOneBy([
↪'code' => 'US_Web_Store']));

$promotion->setCouponBased(true);
```

Then create a coupon and add it to the promotion:

```
/** @var CouponInterface $coupon */
$coupon = $this->container->get('sylius.factory.promotion_coupon')->createNew();

$coupon->setCode('FREESHIPPING');

$promotion->addCoupon($coupon);
```

Now create an PromotionAction that will take place after applying this promotion - 100% discount on shipping

```
/** @var PromotionActionFactoryInterface $actionFactory */
$actionFactory = $this->container->get('sylius.factory.promotion_action');

// Provide the amount in float ( 1 = 100%, 0.1 = 10% )
$action = $actionFactory->createShippingPercentageDiscount(1);

$promotion->addAction($action);

$this->container->get('sylius.repository.promotion')->add($promotion);
```

Finally to see the effects of your promotion with coupon you need to **apply a coupon on the Order**.

How to apply a coupon to an Order?

To apply your promotion with coupon that gives 100% discount on the shipping costs you need an order that has shipments. Set your promotion coupon on that order - this is what happens when a customer provides a coupon code during checkout.

And after that call the OrderProcessor on the order to have the promotion applied.

```
$order->setPromotionCoupon($coupon);

$this->container->get('sylius.order_processing.order_processor')->process($order);
```

Promotion Coupon Generator

Making up new codes might become difficult if you would like to prepare a lot of coupons at once. That is why Sylius provides a service that generates random codes for you - CouponGenerator. In its **PromotionCouponGeneratorInstruction** you can define the amount of coupons that will be generated, the length of their codes, expiration date and usage limit.

```
// Find a promotion you desire in the repository
$promotion = $this->container->get('sylius.repository.promotion')->findOneBy(['code'␣
↪=> 'simple_promotion']);

// Get the CouponGenerator service
```

```php
/** @var CouponGeneratorInterface $generator */
$generator = $this->container->get('sylius.promotion_coupon_generator');

// Then create a new empty PromotionCouponGeneratorInstruction
/** @var PromotionCouponGeneratorInstructionInterface $instruction */
$instruction = new PromotionCouponGeneratorInstruction();

// By default the instruction will generate 5 coupons with codes of length equal to 6
// You can easily change it with the ``setAmount()`` and ``setLength()`` methods
$instruction->setAmount(10);

// Now use the ``generate()`` method with your instruction on the promotion where you
→want to have Coupons
$generator->generate($promotion, $instruction);
```

The above piece of code will result in a set of 10 coupons that will work with the promotion identified by the
`simple_promotion` code.

Learn more

- *Promotions Concept Documentation*
- *promotion - Component Documentation*
- *promotion - Bundle Documentation*

Shipments

A **Shipment** is a representation of a shipping request for an Order. Sylius can attach multiple shipments to each single
Order.

How is a Shipment created for an Order?

> **Warning:** Read more about creating *Orders* where the process of assigning Shipments is clarified.

The Shipment State Machine

A Shipment that is attached to an Order will have its own state machine with the following states available: `cart`,
`ready`, `cancelled`, `shipped`.

The allowed transitions between these states are:

```yaml
transitions:
    create:
        from: [cart]
        to: ready
    ship:
        from: [ready]
        to: shipped
    cancel:
```

```
        from: [ready]
        to: cancelled
```
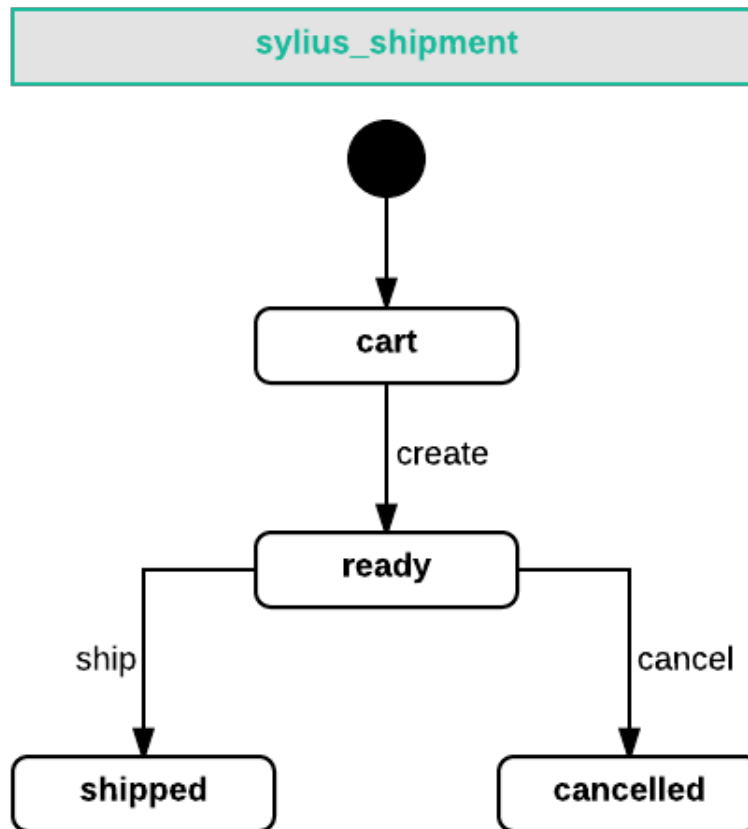

Shipping Methods

ShippingMethod in Sylius is an entity that represent the way an order can be shipped to a customer.

How to create a ShippingMethod programmatically?

As usual use a factory to create a new ShippingMethod. Give it a `code`, set a desired shipping calculator and set a `zone`. It also need a configuration, for instance of the amount (cost). At the end add it to the system using a repository.

```
$shippingMethod = $this->container->get('sylius.factory.shipping_method')->
→createNew();

$shippingMethod->setCode('DHL');
$shippingMethod->setCalculator(DefaultCalculators::FLAT_RATE);
$shippingMethod->setConfiguration(['channel_code' => ['amount' => 50]]);

$zone = $this->container->get('sylius.repository.zone')->findOneByCode('US');
$shippingMethod->setZone($zone);
```

```
$this->container->get('sylius.repository.shipping_method')->add($shippingMethod);
```

In order to have your shipping method available in checkout add it to a desired channel.

```
$channel = $this->container->get('sylius.repository.channel')->findOneByCode('channel_
↪code');
$channel->addShippingMethod($shippingMethod);
```

Shipping Zones

Sylius has an approach of **Zones** used also for shipping. As in each e-commerce you may be willing to ship only to certain countries for example. Therefore while configuring your **ShippingMethods** pay special attention to the zones you are assigning to them. You have to prepare methods for each zone, because the available methods are retrieved for the zone the customer has basing on his address.

Shipping Cost Calculators

The shipping cost calculators are services that are used to calculate the cost for a given shipment.

The CalculatorInterface has a method `calculate()` that takes object with a configuration and returns *integer* that is the cost of shipping for that subject. It also has a `getType()` method that works just like in the forms.

To select a proper service we have a one that decides for us - the DelegatingCalculator. Basing on the **ShippingMethod** assigned on the Shipment it will get its calculator type and configuration and calculate the cost properly.

```
$shippingCalculator = $this->container->get('sylius.shipping_calculator');

$cost = $shippingCalculator->calculate($shipment);
```

Built-in Calculators

The already defined calculators in Sylius are described as constants in the SyliusComponentShippingCalculatorDefaultCalculators

- **FlatRateCalculator** - just returns the `amount` from the ShippingMethod's configuration.

- **PerUnitRateCalculator** - returns the `amount` from the ShippingMethod's configuration multiplied by the `units` count.

Shipment complete events

There are two events that are triggered on the shipment `ship` action:

Event id
`sylius.shipment.pre_ship`
`sylius.shipment.post_ship`

Learn more

- *Shipping - Component Documentation*

Payments

Sylius contains a very flexible payments management system with support for many gateways (payment providers). We are using a payment abstraction library - Payum, which handles all sorts of capturing, refunding and recurring payments logic.

On Sylius side, we integrate it into our checkout and manage all the payment data.

Payment

Every payment in Sylius, successful or failed, is represented by the **Payment** model, which contains basic information and a reference to appropriate order.

Payment State Machine

A Payment that is assigned to an order will have it's own state machine with a few available states: `cart`, `new`, `processing`, `completed`, `failed`, `cancelled`, `refunded`.

The available transitions between these states are:

```yaml
transitions:
    create:
        from: [cart]
        to: new
    process:
        from: [new]
        to: processing
    complete:
        from: [new, processing]
        to: completed
    fail:
        from: [new, processing]
        to: failed
    cancel:
        from: [new, processing]
        to: cancelled
    refund:
        from: [completed]
        to: refunded
```
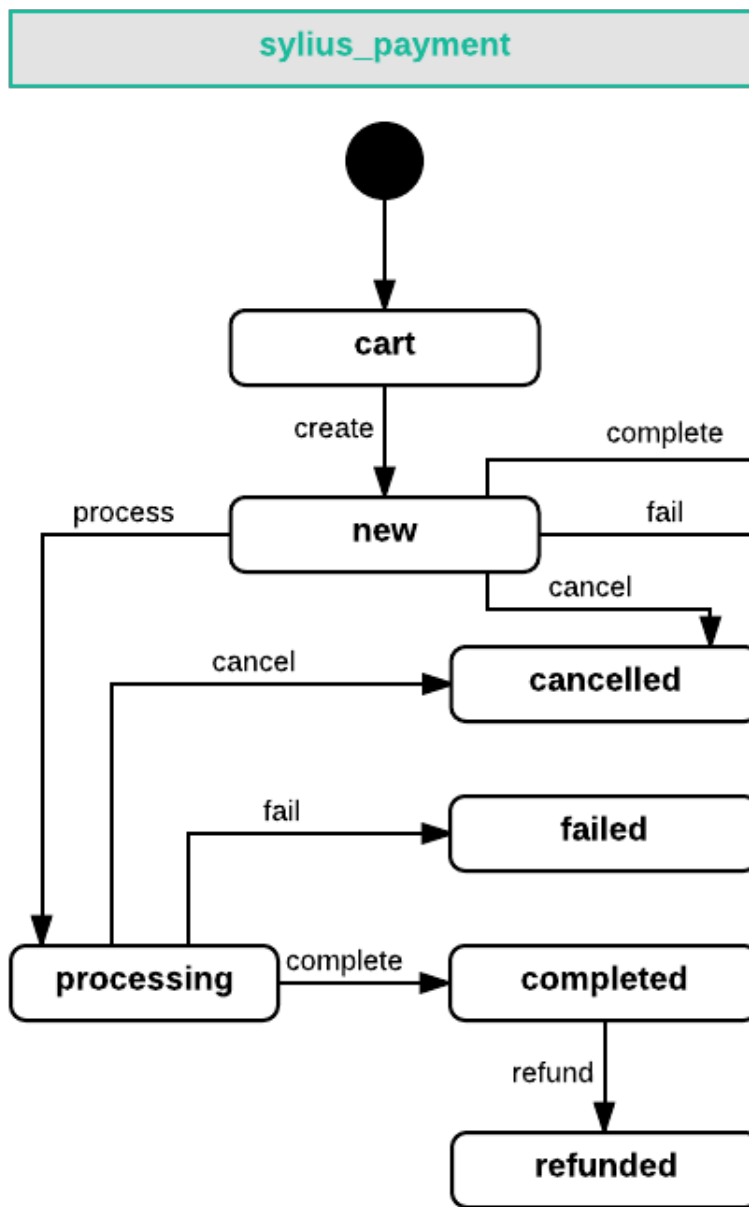
Of course, you can define your own states and transitions to create a workflow, that perfectly matches your needs. Full configuration can be seen in the PaymentBundle/Resources/config/app/state_machine.yml.

Changes to payment happen through applying appropriate transitions.

How to create a Payment programmatically?

We cannot create a Payment without an Order, therefore let's assume that you have an Order to which you will assign a new payment.

```
$payment = $this->container->get('sylius.factory.payment')->createNew();

$payment->setOrder($order);
$payment->setCurrencyCode('USD');

$this->container->get('sylius.repository.payment')->add($payment);
```

Tip: Not familiar with the Order concept? Check *here*.

Payment Methods

A **PaymentMethod** represents a way that your customer pays during the checkout process. It holds a reference to a specific `gateway` with custom configuration. Gateway is configured for each payment method separately using the payment method form.

How to create a PaymentMethod programmatically?

As usual, use a factory to create a new PaymentMethod and give it a unique code.

```
$paymentMethod = $this->container->get('sylius.factory.payment_method')->
↪createWithGateway('offline');
$paymentMethod->setCode('ALFA1');

$this->container->get('sylius.repository.payment_method')->add($paymentMethod);
```

In order to have your new payment method available in the checkout remember to **add your desired channel to the payment method**:

```
$paymentMethod->addChannel($channel)
```

Payment Gateway configuration

Payment Gateways that already have a Sylius bridge

First you need to create the configuration form type for your gateway. Have a look at the configuration form types of Paypal and Stripe.

Then you should register its configuration form type with `sylius.gateway_configuration_type` tag. After that it will be available in the Admin panel in the gateway choice dropdown.

Tip: If you are not sure how your configuration form type should look like, head to Payum documentation.

Other Payment Gateways

Note: Learn more about integrating payment gateways in the Payum docs.

When the Payment Gateway you are trying to use does have a bridge available and you integrate them on your own, use our guide on *extension development*.

Tip: You'll probably need also this kind of configuration in your `app/config/config.yml` for the gateway's factory:

```yaml
payum:
    gateways:
        yourgateway:
            factory: yourgateway
```

Troubleshooting

Sylius stores the payment output inside the **details** column of the **sylius_payment** table. It can provide valuable information when debugging the payment process.

PayPal Error Code 10409

The 10409 code, also known as the *"Checkout token was issued for a merchant account other than yours"* error. You have most likely changed the PayPal credentials from `config.yml` during the checkout process. Clear the cache and try again:

```
bin/console cache:clear
```

Payment complete events

There are two events that are triggered on the payment complete action:

Event id
sylius.payment.pre_complete
sylius.payment.post_complete

Learn more

- *Payment - Component Documentation*
- Payum - Project Documentation

Checkout

Checkout is a process that begins when the Customer decides to finish their shopping and pay for their order. The process of specifying address, payment and a way of shipping transforms the **Cart** into an **Order**.

Checkout State Machine

The Order Checkout state machine has 5 states available: `cart`, `addressed`, `shipping_selected`, `payment_selected`, `completed` and a set of defined transitions between them. These states are saved as the **checkoutState** of the **Order**.

Besides the steps of checkout, each of them can be done more than once. For instance if the Customer changes their mind and after selecting payment they want to change the shipping address they have already specified, they can of course go back and readdress it.

The transitions on the order checkout state machine are:

```yaml
transitions:
    address:
        from: [cart]
        to: addressed
    readdress:
        from: [payment_selected, shipping_selected, addressed]
        to: cart
    select_shipping:
        from: [addressed]
        to: shipping_selected
    reselect_shipping:
        from: [payment_selected, shipping_selected]
        to: addressed
    select_payment:
        from: [shipping_selected]
        to: payment_selected
    reselect_payment:
        from: [payment_selected]
        to: shipping_selected
    complete:
        from: [payment_selected]
        to: completed
```
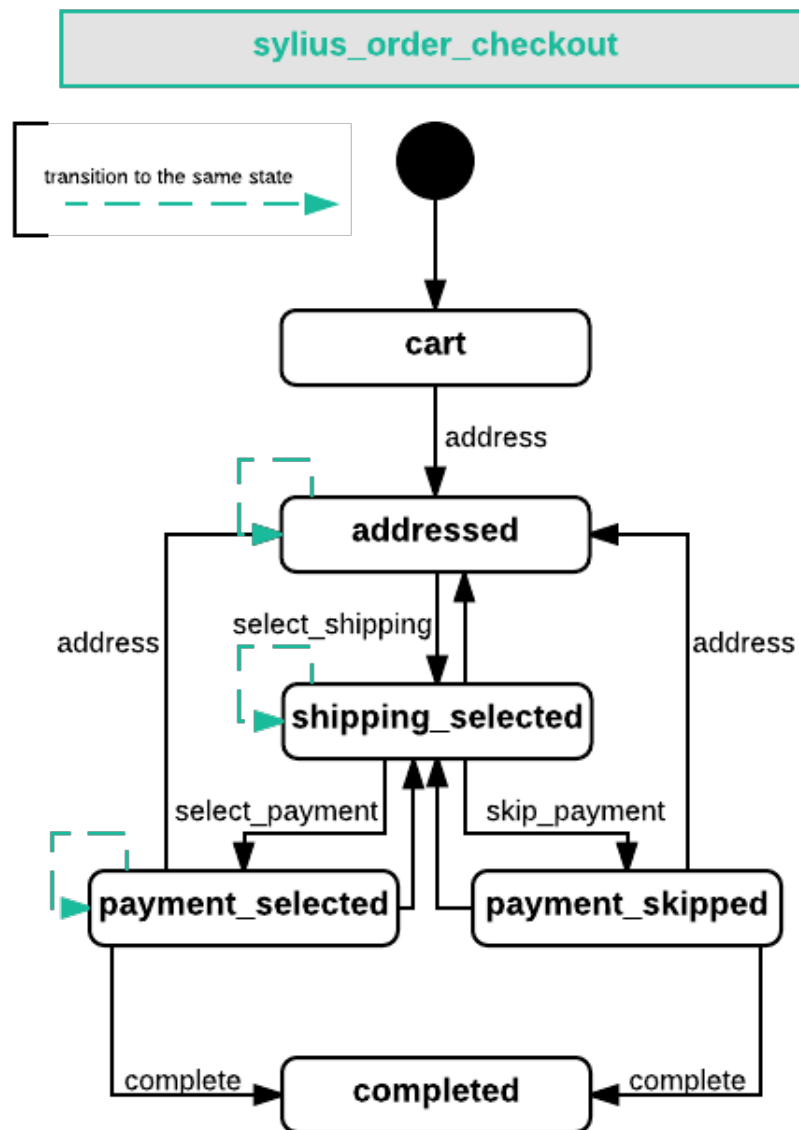
Steps of Checkout

Checkout in Sylius is divided into 4 steps. Each of these steps occurs when the Order goes into a certain state. See the Checkout state machine in the state_machine.yml together with the routing file for checkout: checkout.yml.

Note: Before performing Checkout *you need to have an Order created*.

Addressing

This is a step where the customer provides both **shipping and billing addresses**.

Transition after step	Template
cart-> addressed	SyliusShopBundle:Checkout:addressing.html.twig

How to perform the Addressing Step programmatically?

Firstly if the **Customer** is not yet set on the Order it will be assigned depending on the case:

- An already logged in **User** - the Customer is set for the Order using the CartBlamerListener, that determines the user basing on the event.

- An existent **User** that is not logged in - If there is an account in the system registered under the e-mail that has been provided - they are asked for a password to log in before continuing inside the addressing form.

- A **Customer** that was present in the system before (we've got their e-mail) - the Customer instance is updated via cascade, the order is assigned to it.

- A new **Customer** with unknown e-mail - a new Customer instance is created and assigned to the order.

Hint: If you do not understand the Users and Customers concept in Sylius go to the *Users Concept documentation*.

The typical **Address** consists of: country, city, street and postcode - to assign it to an Order either create it manually or retrieve from the repository.

```
/** @var AddressInterface $address */
$address = $this->container->get('sylius.factory.address')->createNew();

$address->setFirstName('Anne');
$address->setLastName('Shirley');
$address->setStreet('Avonlea');
$address->setCountryCode('CA');
$address->setCity('Canada');
$address->setPostcode('C0A 1N0');

$order->setShippingAddress($address);
$order->setBillingAddress($address);
```

Having the **Customer** and the **Address** set you can apply a state transition to your order. Get the StateMachine for the Order via the StateMachineFactory with a proper schema, and apply a transition and of course flush your order after that via the manager.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderCheckoutTransitions::GRAPH);
$stateMachine->apply(OrderCheckoutTransitions::TRANSITION_ADDRESS);

$this->container->get('sylius.manager.order')->flush();
```

What happens during the transition?

The method process($order) of the CompositeOrderProcessor is run.

Selecting shipping

It is a step where the customer selects the way their order will be shipped to them. Basing on the ShippingMethods configured in the system the options for the Customer are provided together with their prices.

Transition after step	Template
`addressed`-> `shipping_selected`	`SyliusShopBundle:Checkout:shipping.html.twig`

How to perform the Selecting shipping Step programmatically?

Before approaching this step be sure that your Order is in the `addressed` state. In this state your order will already have a default ShippingMethod assigned, but in this step you can change it and have everything recalculated automatically.

Firstly either create new (see how in the Shipments concept) or retrieve a **ShippingMethod** from the repository to assign it to your order's shipment created defaultly in the addressing step.

```
// Let's assume you have a method with code 'DHL' that has everything set properly
$shippingMethod = $this->container->get('sylius.repository.shipping_method')->
↪findOneByCode('DHL');

// Shipments are a Collection, so even though you have one Shipment by default you
↪have to iterate over them
foreach ($order->getShipments() as $shipment) {
    $shipment->setMethod($shippingMethod);
}
```

After that get the StateMachine for the Order via the StateMachineFactory with a proper schema, and apply a proper transition and flush the order via the manager.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderCheckoutTransitions::GRAPH)
$stateMachine->apply(OrderCheckoutTransitions::TRANSITION_SELECT_SHIPPING);

$this->container->get('sylius.manager.order')->flush();
```

What happens during the transition?

The method `process($order)` of the CompositeOrderProcessor is run. Here this method is responsible for: controlling the **shipping charges** which depend on the chosen ShippingMethod, controlling the **promotions** that depend on the shipping method.

Skipping shipping step

What if in the order you have only products that do not require shipping (they are downloadable for example)?

Note: When all of the *ProductVariants* of the order have the `shippingRequired` property set to `false`, then Sylius assumes that the whole order **does not require shipping**, and **the shipping step of checkout will be skipped**.

Selecting payment

This is a step where the customer chooses how are they willing to pay for their order. Basing on the PaymentMethods configured in the system the possibilities for the Customer are provided.

Transition after step	Template
`shipping_selected->` `payment_selected`	`SyliusShopBundle:Checkout:payment.html.` `twig`

How to perform the Selecting payment step programmatically?

Before this step your Order should be in the `shipping_selected` state. It will have a default Payment selected after the addressing step, but in this step you can change it.

Firstly either create new (see how in the Payments concept) or retrieve a **PaymentMethod** from the repository to assign it to your order's payment created defaultly in the addressing step.

```
// Let's assume that you have a method with code 'paypal' configured
$paymentMethod = $this->container->get('sylius.repository.payment_method')->
↪findOneByCode('paypal');

// Payments are a Collection, so even though you hve one Payment by default you have
↪to iterate over them
foreach ($order->getPayments() as $payment) {
    $payment->setMethod($paymentMethod);
}
```

After that get the StateMachine for the Order via the StateMachineFactory with a proper schema, and apply a proper transition and flush the order via the manager.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderCheckoutTransitions::GRAPH)
$stateMachine->apply(OrderCheckoutTransitions::TRANSITION_SELECT_PAYMENT);

$this->container->get('sylius.manager.order')->flush();
```

What happens during the transition?

The method `process($order)` of the CompositeOrderProcessor is run and checks all the adjustments on the order.

Finalizing

In this step the customer gets an order summary and is redirected to complete the payment they have selected.

Transition after step	Template
`payment_selected-> completed`	`SyliusShopBundle:Checkout:summary.html.twig`

How to complete Checkout programmatically?

Before executing the completing transition you can set some notes to your order.

```
$order->setNotes('Thank you dear shop owners! I am allergic to tape so please use␣
↪something else for packaging.')
```

After that get the StateMachine for the Order via the StateMachineFactory with a proper schema, and apply a proper transition and flush the order via the manager.

```
$stateMachineFactory = $this->container->get('sm.factory');

$stateMachine = $stateMachineFactory->get($order, OrderCheckoutTransitions::GRAPH);
$stateMachine->apply(OrderCheckoutTransitions::TRANSITION_COMPLETE);

$this->container->get('sylius.manager.order')->flush();
```

What happens during the transition?

- The Order will have the **checkoutState** - `completed`,

- The Order will have the general **state** - `new` instead of `cart` it has had before the transition,

- When the Order is transitioned from `cart` to `new` the **paymentState** is set to `awaiting_payment` and the **shippingState** to `ready`

The Checkout is finished after that.

Checkout related events

On each step of checkout a dedicated event is triggered.

Event id
`sylius.order.pre_address`
`sylius.order.post_address`
`sylius.order.pre_select_shipping`
`sylius.order.post_select_shipping`
`sylius.order.pre_payment`
`sylius.order.post_payment`
`sylius.order.pre_complete`
`sylius.order.post_complete`

Learn more

- *State Machine - Documentation*

- *Orders - Concept Documentation*

- *Orders*

- *Taxation*

- *Adjustments*

- *Promotions*

- *Coupons*

- *Payments*

- *Shipments*

- *Checkout*

- *Orders*

- *Taxation*

- *Adjustments*

- *Promotions*

- *Coupons*

- *Payments*

- *Shipments*

- *Checkout*

1.1.8 Themes

Here you will learn basics about the Theming concept of Sylius. How to change the theme of your shop? keep reading!

Themes

Themes

Theming is a method of customizing how your channels look like in Sylius. Each channel can have a different theme.

What is the purpose of using themes?

There are some criteria that you have to analyze before choosing either *standard Symfony template overriding* or themes.

When you should choose standard template overriding:

- you have only one channel

- **or** you do not need different looks/themes on each of you channels

- you need only basic changes in the views (changing colors, some blocks rearranging)

When you should use Sylius themes:

- you have more than one channel for a single Sylius instance

- **and** you want each channel to have their own look and behaviour

- you change a lot of things in the views

How to enable themes in a project?

To use themes inside of your project you need to add these few lines to your `app/config/config.yml`.

```
sylius_theme:
    sources:
        filesystem: ~
```

How to create themes?

Let's see how to customize the login view inside of your custom theme.

1. Inside of the `app/themes/` directory create a new directory for your theme:

Let it be `CrimsonTheme/` for instance.

2. Create `composer.json` for your theme:

```json
{
    "name": "acme/crimson-theme",
    "authors": [
        {
            "name": "James Potter",
            "email": "prongs@example.com"
        }
    ],
    "extra": {
        "sylius-theme": {
            "title": "Crimson Theme"
        }
    }
}
```

3. Install theme assets

Theme assets are installed by running the `sylius:theme:assets:install` command, which is supplementary for and should be used after `assets:install`.

```
bin/console sylius:theme:assets:install
```

The command run with `--symlink` or `--relative` parameters creates symlinks for every installed asset file, not for entire asset directory (eg. if `AcmeBundle/Resources/public/asset.js` exists, it creates symlink `web/bundles/acme/asset.js` leading to `AcmeBundle/Resources/public/asset.js` instead of symlink `web/bundles/acme/` leading to `AcmeBundle/Resources/public/`). When you create a new asset or delete an existing one, it is required to rerun this command to apply changes (just as the hard copy option works).

Note: Whenever you install a new bundle with assets you will need to run `sylius:theme:assets:install` again to make sure they are accessible in your theme.

---

4. Customize a template:

In order to customize the login view you should take the content of `@SyliusShopBundle/views/login.html.twig` file and paste it to your theme directory: `app/themes/CrimsonTheme/SyliusShopBundle/views/login.html.twig`

Let's remove the registration column in this example:

```twig
{% extends '@SyliusShop/layout.html.twig' %}

{% form_theme form 'SyliusUiBundle:Form:theme.html.twig' %}

{% import 'SyliusUiBundle:Macro:messages.html.twig' as messages %}

{% block content %}
    {% include '@SyliusShop/Login/_header.html.twig' %}
```

```
    <div class="ui padded segment">
        <div class="ui one column very relaxed stackable grid">
            <div class="column">
                <h4 class="ui dividing header">{{ 'sylius.ui.registered_customers
→'|trans }}</h4>
                <p>{{ 'sylius.ui.if_you_have_an_account_sign_in_with_your_email_
→address'|trans }}.</p>
                {{ form_start(form, {'action': path('sylius_shop_login_check'), 'attr
→': {'class': 'ui loadable form', 'novalidate': 'novalidate'}}) }}
                    {% include '@SyliusShop/Login/_form.html.twig' %}
                    <button type="submit" class="ui blue submit button">{{ 'sylius.ui.
→login'|trans }}</button>
                    <a href="{{ path('sylius_shop_request_password_reset_token') }}"
→class="ui right floated button">{{ 'sylius.ui.forgot_password'|trans }}</a>
                {{ form_end(form, {'render_rest': false}) }}
            </div>
        </div>
    </div>
{% endblock %}
```

Tip: Learn more about customizing templates *here*.

---

5. Choose your new theme on the channel:

In the administration panel go to channels and change the theme of your desired channel to `Crimson Theme`.

6. If changes are not yet visible, clear the cache:

```
$ php bin/console cache:clear
```

Learn more

- *Theme - Bundle Documentation*.
- *Themes*
- *Themes*

The Customization Guide

The Customization Guide is helpful while wanting to adapt Sylius to your personal business needs.

2.1 The Customization Guide

The Customization Guide is helpful while wanting to adapt Sylius to your personal business needs.

2.1.1 Customizing Models

All models in Sylius are placed in the `Sylius\Component\*ComponentName*\Model` namespaces alongside with their interfaces.

> **Warning:** Many models in Sylius are **extended in the Core component**. If the model you are willing to override exists in the `Core` you should be extending the `Core` one, not the base model from the component.

Note: Note that there are **translatable models** in Sylius also. The guide to translatable entities can be found below the regular one.

Why would you customize a Model?

To give you an idea of some purposes of models customizing have a look at a few examples:

- Add `flag` field to the `Country`

- Add `secondNumber` to the `Customer`

- Change the `reviewSubject` of a `Review` (in Sylius we have `ProductReviews` but you can imagine for instance a `CustomerReview`)

• Add `icon` to the `PaymentMethod`

And of course many similar operations limited only by your imagination. Let's now see how you should perform such customizations.

How to customize a Model?

Let's take the `Sylius\Component\Addressing\Country` as an example. This one is not extended in Core. How can you check that?

For the `Country` run:

```
$ php bin/console debug:container --parameter=sylius.model.country.class
```

As a result you will get the `Sylius\Component\Addressing\Model\Country` - this is the class that you need to be extending.

Assuming that you would want to add another field on the model - for instance a `flag`.

1. The first thing to do is to write your own class which will extend the base `Country` class.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Addressing\Model\Country as BaseCountry;

class Country extends BaseCountry
{
    /**
     * @var bool
     */
    private $flag;

    /**
     * @return bool|null
     */
    public function getFlag(): ?bool
    {
        return $this->flag;
    }

    /**
     * @param bool $flag
     */
    public function setFlag(bool $flag): void
    {
        $this->flag = $flag;
    }
}
```

2. Next define your entity's mapping.

The file should be placed in `AppBundle/Resources/config/doctrine/Country.orm.yml`

```yaml
AppBundle\Entity\Country:
    type: entity
    table: sylius_country
```

```
    fields:
        flag:
            type: boolean
            nullable: true
```

3. Finally you'll need to override the model's class in the `app/config/config.yml`.

Under the `sylius_*` where `*` is the name of the bundle of the model you are customizing, in our case it will be the `SyliusAddressingBundle` -> `sylius_addressing`.

```
sylius_addressing:
    resources:
        country:
            classes:
                model: AppBundle\Entity\Country
```

4. Update the database. There are two ways to do it.

- via direct database schema update:

```
$ php bin/console doctrine:schema:update --force
```

- via migrations:

Which we strongly recommend over updating the schema.

```
$ php bin/console doctrine:migrations:diff
$ php bin/console doctrine:migrations:migrate
```

Tip: Read more about the database modifications and migrations in the Symfony documentation here.

---

5. Additionally if you want to give the administrator an ability to add the `flag` to any of countries, you'll need to update its form type. Check how to do it *here*.

What happens while overriding Models?

- Parameter `sylius.model.country.class` contains `AppBundle\Entity\Country`.
- `sylius.repository.country` represents Doctrine repository for your new class.
- `sylius.manager.country` represents Doctrine object manager for your new class.
- `sylius.controller.country` represents the controller for your new class.
- All Doctrine relations to `Sylius\Component\Addressing\Model\Country` are using your new class as *target-entity*, you do not need to update any mappings.
- `CountryType` form type is using your model as `data_class`.
- `Sylius\Component\Addressing\Model\Country` is automatically turned into Doctrine Mapped Superclass.

How to customize a translatable Model?

One of translatable entities in Sylius is the Shipping Method. Let's try to extend it with a new field. Shipping methods may have different delivery time, let's save it on the `estimatedDeliveryTime` field.

Just like for regular models you can also check the class of translatable models like that:

```
$ php bin/console debug:container --parameter=sylius.model.shipping_method.class
```

1. The first thing to do is to write your own class which will extend the base `ShippingMethod` class.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Core\Model\ShippingMethod as BaseShippingMethod;
use Sylius\Component\Shipping\Model\ShippingMethodTranslation;

class ShippingMethod extends BaseShippingMethod
{
    /**
     * @var string
     */
    private $estimatedDeliveryTime;

    /**
     * @return string
     */
    public function getEstimatedDeliveryTime(): string
    {
        return $this->estimatedDeliveryTime;
    }

    /**
     * @param string $estimatedDeliveryTime
     */
    public function setEstimatedDeliveryTime(string $estimatedDeliveryTime): void
    {
        $this->estimatedDeliveryTime = $estimatedDeliveryTime;
    }

    /**
     * {@inheritdoc}
     */
    protected function createTranslation(): ShippingMethodTranslation
    {
        return new ShippingMethodTranslation();
    }
}
```

Note: Remember to set the translation class properly, just like above in the `createTranslation()` method.

2. Next define your entity's mapping.

The file should be placed in `AppBundle/Resources/config/doctrine/ShippingMethod.orm.yml`

```yaml
AppBundle\Entity\ShippingMethod:
    type: entity
    table: sylius_shipping_method
    fields:
        estimatedDeliveryTime:
```

(continues on next page)

```
            type: string
            nullable: true
```

3. Finally you'll need to override the model's class in the `app/config/config.yml`.

Under the `sylius_*` where `*` is the name of the bundle of the model you are customizing, in our case it will be the `SyliusShippingBundle` -> `sylius_shipping`.

```
sylius_shipping:
    resources:
        shipping_method:
            classes:
                model: AppBundle\Entity\ShippingMethod
```

4. Update the database. There are two ways to do it.

  • via direct database schema update:

```
$ php bin/console doctrine:schema:update --force
```

  • via migrations:

Which we strongly recommend over updating the schema.

```
$ php bin/console doctrine:migrations:diff
$ php bin/console doctrine:migrations:migrate
```

Tip: Read more about the database modifications and migrations in the Symfony documentation here.

---

5. Additionally if you need to add the `estimatedDeliveryTime` to any of your shipping methods in the admin panel, you'll need to update its form type. Check how to do it *here*.

Warning: If you want the new field of your entity to be translatable, you need to extend the Translation class of your entity. In case of the ShippingMethod it would be the `Sylius\Component\Shipping\Model\ShippingMethodTranslation`. Also the form on which you will add the new field should be the TranslationType.

How to customize translatable fields of a translatable Model?

Suppose you want to add a translatable property to a translatable entity, for example to the Shipping Method. Let's assume that you would like the Shipping method to include a message with the delivery conditions. Let's save it on the `deliveryConditions` field.

Just like for regular models you can also check the class of translatable models like that:

```
$ php bin/console debug:container --parameter=sylius.model.shipping_method_
↪translation.class
```

1. In order to add a translatable property to your entity you need to define it on the `AppBundle\Entity\ShippingMethodTranslation` class of your bundle, that will extend the base `Sylius\Component\Shipping\Model\ShippingMethodTranslation`.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Shipping\Model\ShippingMethodTranslation as
→BaseShippingMethodTranslation;

class ShippingMethodTranslation extends BaseShippingMethodTranslation
{
    /**
     * @var string
     */
    private $deliveryConditions;

    /**
     * @return string
     */
    public function getDeliveryConditions(): string
    {
        return $this->deliveryConditions;
    }

    /**
     * @param string $deliveryConditions
     */
    public function setDeliveryConditions(string $deliveryConditions): void
    {
        $this->deliveryConditions = $deliveryConditions;
    }
}
```

2. Next define your translation entity's mapping.

The translation's entity file should be placed in `AppBundle/Resources/config/doctrine/` `ShippingMethodTranslation.orm.yml`

```yaml
AppBundle\Entity\ShippingMethodTranslation:
    type: entity
    table: sylius_shipping_method_translation
    fields:
        deliveryConditions:
            type: string
            nullable: true
```

3. You'll need to provide access to the new fields in the `ShippingMethod` class by extending the base Shipping-Method class.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Core\Model\ShippingMethod as BaseShippingMethod;

class ShippingMethod extends BaseShippingMethod
{
    /**
     * @return string
```

(continues on next page)

(continued from previous page)

```php
     */
    public function getDeliveryConditions(): string
    {
        return $this->getTranslation()->getDeliveryConditions();
    }

    /**
     * @param string $deliveryConditions
     */
    public function setDeliveryConditions(string $deliveryConditions): void
    {
        $this->getTranslation()->setDeliveryConditions($deliveryConditions);
    }
}
```

Note: Remember that if the original entity is not translatable you will need to initialize the translations collection in the constructor, and use the TranslatableTrait. Take a careful look at the Sylius translatable entities.

4. As we are overriding not only the translation class but also the base class, we need to create an emty mapping also for this base class.

The mapping file should be placed in `AppBundle/Resources/config/doctrine/ShippingMethod.orm.yml`

```yaml
AppBundle\Entity\ShippingMethod:
    type: entity
    table: sylius_shipping_method
```

5. Finally you'll need to override the model's classes in the `app/config/config.yml`.

Under the `sylius_*` where `*` is the name of the bundle of the model you are customizing, in our case it will be the `SyliusShippingBundle` -> `sylius_shipping`.

```yaml
sylius_shipping:
    resources:
        shipping_method:
            classes:
                model: AppBundle\Entity\ShippingMethod
            translation:
                classes:
                    model: AppBundle\Entity\ShippingMethodTranslation
```

6. Update the database. There are two ways to do it.

- via direct database schema update:

```
$ php bin/console doctrine:schema:update --force
```

- via migrations:

Which we strongly recommend over updating the schema.

```
$ php bin/console doctrine:migrations:diff
$ php bin/console doctrine:migrations:migrate
```

Tip: Read more about the database modifications and migrations in the Symfony documentation here.

---

6. Additionally if you need to add the `deliveryConditions` to any of your shipping methods in the admin panel, you'll need to update its form type. Check how to do it *here*.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.2 Customizing Forms

The forms in Sylius are placed in the `Sylius\Bundle\*BundleName*\Form\Type` namespaces and the extensions will be placed in *AppBundleFormExtension*.

Why would you customize a Form?

There are plenty of reasons to modify forms that have already been defined in Sylius. Your business needs may sometimes slightly differ from our internal assumptions.

You can:

- add completely **new fields**,
- **modify** existing fields, make them required, change their HTML class, change labels etc.,
- **remove** fields that are not used.

How to customize a Form?

If you want to modify the form for the `Customer Profile` in your system there are a few steps that you should take. Assuming that you would like to (for example):

- Add a `contactHours` field,
- Remove the `gender` field,
- Change the label for the `lastName` from `sylius.form.customer.last_name` to `app.form.customer.surname`

These will be the steps that you will have to take to achieve that:

1. If you are planning to add new fields remember that beforehand they need to be added on the model that the form type is based on.

> In case of our example if you need to have the `contactHours` on the model and the entity mapping for the `Customer` resource. To get to know how to prepare that go *there*.

2. Create a **Form Extension**.

Your form has to extend a proper base class. How can you check that?

For the `CustomerProfileType` run:

```
$ php bin/console debug:container sylius.form.type.customer_profile
```

As a result you will get the `Sylius\Bundle\CustomerBundle\Form\Type\CustomerProfileType` - this is the class that you need to be extending.

```php
<?php

namespace AppBundle\Form\Extension;

use Sylius\Bundle\CustomerBundle\Form\Type\CustomerProfileType;
use Symfony\Component\Form\AbstractTypeExtension;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;

final class CustomerProfileTypeExtension extends AbstractTypeExtension
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        // Adding new fields works just like in the parent form type.
        $builder->add('contactHours', TextType::class, [
            'required' => false,
            'label' => 'app.form.customer.contact_hours',
        ]);

        // To remove a field from a form simply call ->remove(`fieldName`).
        $builder->remove('gender');

        // You can change the label by adding again the same field with a changed
→`label` parameter.
        $builder->add('lastName', TextType::class, [
            'label' => 'app.form.customer.surname',
        ]);
    }

    /**
     * {@inheritdoc}
     */
    public function getExtendedType(): string
    {
        return CustomerProfileType::class;
    }
}
```

Note: Of course remember that you need to define new labels for your fields in the `app\Resources\translations\messages.en.yml` for english contents of your messages.

3. After creating your class, register this extension as a service in the `app/config/services.yml`:

```yaml
services:
    app.form.extension.type.customer_profile:
        class: AppBundle\Form\Extension\CustomerProfileTypeExtension
        tags:
```

```
            - { name: form.type_extension, extended_type:
↪Sylius\Bundle\CustomerBundle\Form\Type\CustomerProfileType }
```

Note: Of course remember that you need to render the new fields you have created, and remove the rendering of the fields that you have removed **in your views**.

In our case you will need a new template: *app/Resources/SyliusShopBundle/views/Account/profileUpdate.html.twig*.

In **Twig** for example you can render your modified form in such a way:

```
<div class="two fields">
    <div class="field">{{ form_row(form.birthday) }}</div>
    <div class="field">{{ form_row(form.contactHours) }}</div>
</div>
```

Need more information?

Warning: Some of the forms already have extensions in Sylius. Learn more about Extensions here.

For instance the `ProductVariant` admin form is defined under `Sylius/Bundle/ProductBundle/Form/Type/ProductVariantType.php` and later extended in `Sylius/Bundle/CoreBundle/Form/Extension/ProductVariantTypeExtension.php`. If you again extend the base type form like this:

```
services:
    app.form.extension.type.product_variant:
        class: AppBundle\Form\Extension\ProductVariantTypeMyExtension
        tags:
            - { name: form.type_extension, extended_type:
↪Sylius\Bundle\ProductBundle\Form\Type\ProductVariantType, priority: -5 }
```

your form extension will also be executed. Whether before or after the other extensions depends on priority tag set.

Having a look at the extensions and possible additionally defined event handlers can also be useful when form elements are embedded dynamically, as is done in the `ProductVariantTypeExtension` by the `CoreBundle`:

```php
<?php

...

final class ProductVariantTypeExtension extends AbstractTypeExtension
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        ...

        $builder->addEventListener(FormEvents::PRE_SET_DATA, function (FormEvent
↪$event) {
            $productVariant = $event->getData();
```

```
            $event->getForm()->add('channelPricings', ChannelCollectionType::class, [
                'entry_type' => ChannelPricingType::class,
                'entry_options' => function (ChannelInterface $channel) use (
↪$productVariant) {
                    return [
                        'channel' => $channel,
                        'product_variant' => $productVariant,
                        'required' => false,
                    ];
                },
                'label' => 'sylius.form.variant.price',
            ]);
        });
    }

    ...
}
```

The `channelPricings` get added on `FormEvents::PRE_SET_DATA`, so when you wish to remove or alter this form definition, you will also have to set up an event listener and then remove the field:

```php
<?php

...

final class ProductVariantTypeMyExtension extends AbstractTypeExtension
{
    ...

    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        ...

        $builder
            ->addEventListener(FormEvents::PRE_SET_DATA, function (FormEvent $event) {
                $event->getForm()->remove('channelPricings');
            })
            ->addEventSubscriber(new AddCodeFormSubscriber(NULL, ['label' => 'app.
↪form.my_other_code_label']))
        ;

        ...

    }
}
```

Adding constraints inside a form extension

> **Warning:** When adding your constraints dynamically from inside a form extension, be aware to add the correct validation groups.

Although it is advised to follow the *Validation Customization Guide*, it might happen that you want to define the form constraints from inside the form extension. They will not be used unless the correct validation group(s) has been added. The example below shows how to add the default *sylius* group to a constraint.

```php
<?php

...

final class CustomerProfileTypeExtension extends AbstractTypeExtension
{
    ...

    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        ...

        // Adding new fields works just like in the parent form type.
        $builder->add('contactHours', TextType::class, [
            'required' => false,
            'label' => 'app.form.customer.contact_hours',
            'constraints' => [
                new Range([
                    'min' => 8,
                    'max' => 17,
                    'groups' => ['sylius'],
                ]),
            ],
        ]);

        ...
    }

    ...
}
```

Overriding forms completely

Tip: If you need to create a new form type on top of an existing one - create this new alternative form type and define *getParent()* to the old one. See details in the Symfony docs.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.3 Customizing Repositories

Warning: In **Sylius** we are using both default Doctrine repositories and the custom ones. Often you will be needing to add your very own methods to them. You need to check before which repository is your resource using.

Why would you customize a Repository?

Different sets of different resources can be obtained in various scenarios in your application. You may need for instance:

- finding Orders by a Customer and a chosen Product

- finding Products by a Taxon

- finding Comments by a Customer

How to customize a Repository?

Let's assume that you would want to find products that you are running out of in the inventory.

1. Create your own repository class under the `AppBundle\Repository` namespace. Remember that it has to extend a proper base class. How can you check that?

For the `ProductRepository` run:

```
$ php bin/console debug:container sylius.repository.product
```

As a result you will get the `Sylius\Bundle\CoreBundle\Doctrine\ORM\ProductRepository` - this is the class that you need to be extending.

```php
<?php

namespace AppBundle\Repository;

use Sylius\Bundle\CoreBundle\Doctrine\ORM\ProductRepository as BaseProductRepository;

class ProductRepository extends BaseProductRepository
{
    /**
     * @param int $limit
     *
     * @return array
     */
    public function findByOnHand(int $limit = 8): array
    {
        return $this->createQueryBuilder('o')
            ->addSelect('variant')
            ->addSelect('translation')
            ->leftJoin('o.variants', 'variant')
            ->leftJoin('o.translations', 'translation')
            ->addOrderBy('variant.onHand', 'ASC')
            ->setMaxResults($limit)
            ->getQuery()
            ->getResult()
        ;
    }
}
```

We are using the Query Builder in the Repositories. As we are selecting Products we need to have a join to translations, because they are a translatable resource. Without it in the query results we wouldn't have a name to be displayed.

We are sorting the results by the count of how many products are still available on hand, which is saved on the `onHand` field on the specific `variant` of each product. Then we are limiting the query to 8 by default, to get only 8 products that are low in stock.

2. In order to use your repository you need to configure it in the `app/config/config.yml`.

```yaml
sylius_product:
    resources:
        product:
            classes:
                repository: AppBundle\Repository\ProductRepository
```

3. After configuring the `sylius.repository.product` service has your `findByOnHand()` method available. You can form now on use your method in any **Controller**.

```php
<?php

public function lowInStockAction()
{
    $productRepository = $this->container->get('sylius.repository.product');

    $lowInStock = $productRepository->findByOnHand();
}
```

What happens while overriding Repositories?

- The parameter `sylius.repository.product.class` contains `AppBundle\Repository\ProductRepository`.

- The repository service `sylius.repository.product` is using your new class.

- Under the `sylius.repository.product` service you have got all methods from the base repository available plus the one you have added.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.4 Customizing Factories

> **Warning:**   Some factories may already be decorated in the **Sylius** Core. You need to check before decorating which factory (Component or Core) is your resource using.

Why would you customize a Factory?

Differently configured versions of resources may be needed in various scenarios in your application. You may need for instance to:

- create a Product with a Supplier (which is your own custom entity)

- create a disabled Product (for further modifications)

- create a ProductReview with predefined description

and many, many more.

How to customize a Factory?

Let's assume that you would want to have a possibility to create disabled products.

1. Create your own factory class in the `AppBundle\Factory` namespace. Remember that it has to implement a proper interface. How can you check that?

For the `ProductFactory` run:

```
$ php bin/console debug:container sylius.factory.product
```

As a result you will get the `Sylius\Component\Product\Factory\ProductFactory` - this is the class that you need to decorate. Take its interface (`Sylius\Component\Product\Factory\ProductFactoryInterface`) and implement it.

```php
<?php

namespace AppBundle\Factory;

use Sylius\Component\Product\Model\ProductInterface;
use Sylius\Component\Product\Factory\ProductFactoryInterface;

class ProductFactory implements ProductFactoryInterface
{
    /**
     * @var ProductFactoryInterface
     */
    private $decoratedFactory;

    /**
     * @param ProductFactoryInterface $factory
     */
    public function __construct(ProductFactoryInterface $factory)
    {
        $this->decoratedFactory = $factory;
    }

    /**
     * {@inheritdoc}
     */
    public function createNew(): ProductInterface
    {
        return $this->decoratedFactory->createNew();
    }

    /**
     * {@inheritdoc}
     */
    public function createWithVariant(): ProductInterface
    {
        return $this->decoratedFactory->createWithVariant();
    }

    /**
     * @return ProductInterface
     */
    public function createDisabled(): ProductInterface
    {
```

(continues on next page)

```php
        /** @var ProductInterface $product */
        $product = $this->decoratedFactory->createWithVariant();

        $product->setEnabled(false);

        return $product;
    }
}
```

2. In order to decorate the base ProductFactory with your implementation you need to configure it as a decorating service in the `app/Resources/config/services.yml`.

```yaml
services:
    app.factory.product:
        class: AppBundle\Factory\ProductFactory
        decorates: sylius.factory.product
        arguments: ['@app.factory.product.inner']
        public: false
```

3. You can use the new method of the factory in routing.

After the `sylius.factory.product` has been decorated it has got the new `createDisabled()` method. To actually use it overwrite `sylius_admin_product_create_simple` route like below in `app/config/routing/admin/product.yml`:

```yaml
# app/config/routing/admin/product.yml
sylius_admin_product_create_simple:
    path: /products/new/simple
    methods: [GET, POST]
    defaults:
        _controller: sylius.controller.product:createAction
        _sylius:
            section: admin
            factory:
                method: createDisabled # like here for example
            template: SyliusAdminBundle:Crud:create.html.twig
            redirect: sylius_admin_product_update
            vars:
                subheader: sylius.ui.manage_your_product_catalog
                templates:
                    form: SyliusAdminBundle:Product:_form.html.twig
                route:
                    name: sylius_admin_product_create_simple
```

Create a new yaml file located at `app/config/routing/admin.yml`, if it does not exist yet.

```yaml
# app/config/routing/admin.yml
app_admin_product:
    resource: 'admin/product.yml'
```

Remember to import the `app/config/routing/admin.yml` into the `app/config/routing.yml`.

```yaml
# app/config/routing.yml
app_admin:
    resource: 'routing/admin.yml'
    prefix: /admin
```

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

Learn more

- *SyliusResourceBundle creating resources*

2.1.5 Customizing Controllers

All **Sylius** resources use the Sylius/Bundle/ResourceBundle/Controller/ResourceController by default, but some of them have already been extended in Bundles. If you want to override a controller action, check which controller you should be extending.

Note: There are two types of controllers we can define in Sylius:

Resource Controllers - are based only on one Entity, so they return only the resources they have in their name. For instance a `ProductController` should return only products.

Standard Controllers - non-resource; these may use many entities at once, they are useful on more general pages. We are defining these controllers only if the actions we want cannot be done through yaml configuration - like sending emails.

Why would you customize a Controller?

To add your custom actions you need to override controllers. You may need to:

- add a generic action that will render a list of recommended products with a product on its show page.
- render a partial template that cannot be done via yaml resource action.

How to customize a Resource Controller?

Imagine that you would want to render a list of best selling products in a partial template that will be reusable anywhere. Assuming that you already have a method on the `ProductRepository` - you can see such an example *here*. Having this method you may be rendering its result in a new action of the `ProductController` using a partial template.

See example below:

1. Create a new Controller class under the `AppBundle/Controller` namespace.

Remember that it has to extend a proper base class. How can you check that?

For the `ProductController` run:

```
$ php bin/console debug:container sylius.controller.product
```

As a result you will get the `Sylius\Bundle\ResourceBundle\Controller\ResourceController` - this is the class that you need to extend.

Now you have to create the controller that will have a generic action that is basically the `showAction` from the `ResourceController` extended by getting a list of recommended products from your external api.

```php
<?php

namespace AppBundle\Controller;

use FOS\RestBundle\View\View;
use Sylius\Bundle\ResourceBundle\Controller\ResourceController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Sylius\Component\Resource\ResourceActions;

class ProductController extends ResourceController
{
    /**
     * @param Request $request
     *
     * @return Response
     */
    public function showAction(Request $request): Response
    {
        $configuration = $this->requestConfigurationFactory->create($this->metadata,
↪$request);

        $this->isGrantedOr403($configuration, ResourceActions::SHOW);
        $product = $this->findOr404($configuration);

        $recommendationServiceApi = $this->get('app.recommendation_service_api');

        $recommendedProducts = $recommendationServiceApi->getRecommendedProducts(
↪$product);

        $this->eventDispatcher->dispatch(ResourceActions::SHOW, $configuration,
↪$product);

        $view = View::create($product);

        if ($configuration->isHtmlRequest()) {
            $view
                ->setTemplate($configuration->getTemplate(ResourceActions::SHOW . '.
↪html'))
                ->setTemplateVar($this->metadata->getName())
                ->setData([
                    'configuration' => $configuration,
                    'metadata' => $this->metadata,
                    'resource' => $product,
                    'recommendedProducts' => $recommendedProducts,
                    $this->metadata->getName() => $product,
                ])
            ;
        }

        return $this->viewHandler->handle($configuration, $view);
    }
}
```

2. In order to use your controller and its actions you need to configure it in the `app/config/config.yml`.

```yaml
sylius_product:
```

```yaml
    resources:
        product:
            classes:
                controller: AppBundle\Controller\ProductController
```

How to customize a Standard Controller:

Let's assume that you would like to add some logic to the Homepage.

1. Create a new Controller class under the `AppBundle/Controller/Shop` namespace.

If you still need the methods of the original HomepageController, then copy its body to the new class.

```php
<?php

namespace AppBundle\Controller\Shop;

use Symfony\Bundle\FrameworkBundle\Templating\EngineInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

final class HomepageController
{
    /**
     * @var EngineInterface
     */
    private $templatingEngine;

    /**
     * @param EngineInterface $templatingEngine
     */
    public function __construct(EngineInterface $templatingEngine)
    {
        $this->templatingEngine = $templatingEngine;
    }

    /**
     * @param Request $request
     *
     * @return Response
     */
    public function indexAction(Request $request): Response
    {
        return $this->templatingEngine->renderResponse('@SyliusShop/Homepage/index.
→html.twig');
    }

    /**
     * @param Request $request
     *
     * @return Response
     */
    public function customAction(Request $request): Response
    {
        // Put your custom logic here
```

```
    }
}
```

2. The next thing you have to do is to override the `sylius.controller.shop.homepage` service definition in the `app/config/services.yml`.

```yaml
# app/config/services.yml
services:
    sylius.controller.shop.homepage:
        class: AppBundle\Controller\Shop\HomepageController
        arguments: ['@templating']
        public: true
```

Remember to import the `app/config/services.yml` into the `app/config/config.yml`.

```yaml
# app/config/config.yml
imports:
    - { resource: "services.yml" }
```

Tip: Run `$ php bin/console debug:container sylius.controller.shop.homepage` to check if the class has changed to your implementation.

From now on your `customAction` of the `HomepageController` will be available alongside the `indexAction` from the base class.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.6 Customizing Validation

The default validation group for all resources is `sylius`, but you can configure your own validation.

How to customize validation?

Let's take the example of changing the length of `name` for the `Product` entity - watch out the field `name` is hold on the `ProductTranslation` model.

In the `sylius` validation group the minimum length is equal to 2. What if you'd want to have at least 10 characters?

1. Create the `AppBundle/Resources/config/validation.yml`.

In this file you need to overwrite the whole validation of your field that you are willing to modify. Take this configuration from the `Sylius/Bundle/ProductBundle/Resources/config/validation/ProductTranslation.xml` - you can choose format `xml` or `yaml`.

Give it a new, custom validation group - `[app_product]`.

```
Sylius\Component\Product\Model\ProductTranslation:
    properties:
        name:
            - NotBlank:
                message: sylius.product.name.not_blank
                groups: [app_product]
            - Length:
                min: 10
                minMessage: sylius.product.name.min_length
                max: 255
                maxMessage: sylius.product.name.max_length
                groups: [app_product]
```

Tip: When using custom validation messages see here how to add them.

---

2. Configure the new validation group in the `app/config/services.yml`.

```
# app/config/services.yml
parameters:
    sylius.form.type.product_translation.validation_groups: [app_product]
    sylius.form.type.product.validation_groups: [app_product] # the product class
↪also needs to be aware of the translation'a validation
```

Remember to import the `app/config/services.yml` into the `app/config/config.yml`.

```
# app/config/config.yml
imports:
    - { resource: "services.yml" }
```

Done. Now in all forms where the Product `name` is being used, your new validation group will be applied, not letting users add products with name shorter than 10 characters.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.7 Customizing Menus

Adding new positions in your menu is done **via events**.

You have got the `Sylius\Bundle\UiBundle\Menu\Event\MenuBuilderEvent` with `FactoryInterface` and `ItemInterface` of KnpMenu, this lets you manipulate the whole menu.

You've got six events that you should be subscribing to:

```
sylius.menu.shop.account # For the menu of the MyAccount section in shop
sylius.menu.admin.main # For the Admin Panel menu
sylius.menu.admin.customer.show # For the buttons menu on top of the show page of the
↪Customer (/admin/customers/{id})
sylius.menu.admin.order.show # For the buttons menu on top of the show page of the
↪Order (/admin/orders/{id})
```

(continues on next page)

```
sylius.menu.admin.product.form # For the tabular menu on the left hand side of the
↪new/edit pages of the Product (/admin/products/new & /admin/products/{id}/edit)
sylius.menu.admin.product_variant.form # For the tabular menu on the left hand side
↪of the new/edit pages of the ProductVariant (/admin/products/{productId}/variants/
↪new & /admin/products/{productId}/variants/{id}/edit)
```

How to customize Admin Menu?

Tip: Admin Panel menu is the one in the left expandable sidebar on the `/admin/` url.

---

1. In order to add items to the Admin menu in **Sylius** you have to create a `AppBundle\Menu\AdminMenuListener` class.

In the example below we are adding a one new item and sub-item to the Admin panel menu.

```php
<?php

namespace AppBundle\Menu;

use Sylius\Bundle\UiBundle\Menu\Event\MenuBuilderEvent;

final class AdminMenuListener
{
    /**
     * @param MenuBuilderEvent $event
     */
    public function addAdminMenuItems(MenuBuilderEvent $event): void
    {
        $menu = $event->getMenu();

        $newSubmenu = $menu
            ->addChild('new')
            ->setLabel('Custom Admin Submenu')
        ;

        $newSubmenu
            ->addChild('new-subitem')
            ->setLabel('Custom Admin Menu Itemu')
        ;
    }
}
```

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the `sylius.menu.admin.main` event in the `AppBundle/Resources/config/services.yml`.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.menu_builder:
        class: AppBundle\Menu\AdminMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.admin.main, method:
↪addAdminMenuItems }
```

3. Result:

After these two steps your admin panel menu should look like that, the new items appear at the bottom:
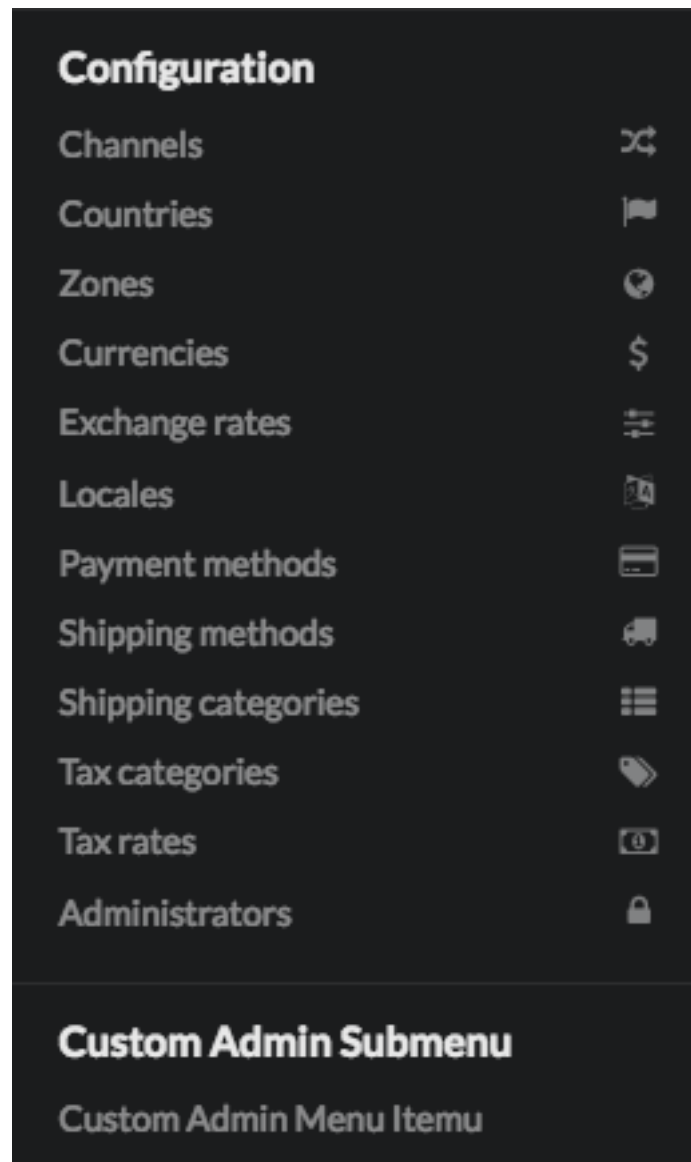
How to customize Account Menu?

Tip: My Account panel menu is the one in the left sidebar on the `/account/dashboard/` url.

---

1. In order to add items to the Account menu in **Sylius** you have to create a `AppBundle\Menu\AccountMenuListener` class.

In the example below we are adding a one new item to **the menu in the My Account section of shop**.

```php
<?php

namespace AppBundle\Menu;
```

<div align="right">(continues on next page)</div>

```php
use Sylius\Bundle\UiBundle\Menu\Event\MenuBuilderEvent;

final class AccountMenuListener
{
    /**
     * @param MenuBuilderEvent $event
     */
    public function addAccountMenuItems(MenuBuilderEvent $event): void
    {
        $menu = $event->getMenu();

        $menu
            ->addChild('new', ['route' => 'sylius_shop_account_dashboard'])
            ->setLabel('Custom Account Menu Item')
            ->setLabelAttribute('icon', 'star')
        ;
    }
}
```

As you can see above the new item can be given a route, a label and an icon.

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the `sylius.menu.shop.account` event in the `AppBundle/Resources/config/services.yml`.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.shop.menu_builder:
        class: AppBundle\Menu\AccountMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.shop.account, method:
↪addAccountMenuItems }
```

3. Result:

After these two steps your user account menu should look like that, the new item appears at the bottom:
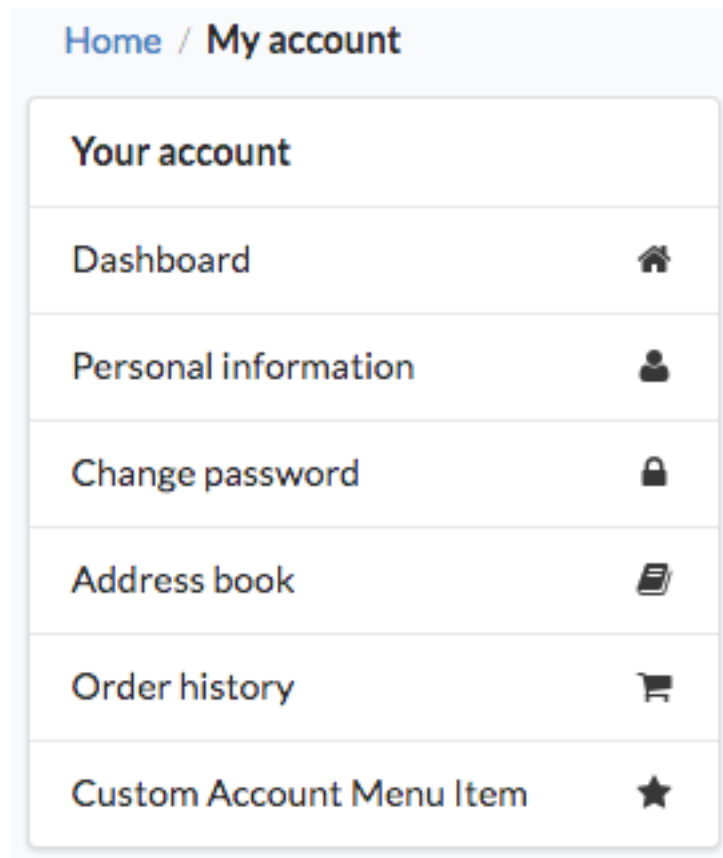
How to customize Admin Customer Show Menu?

Tip: Admin customer menu is the set of buttons in the right top corner on the `/admin/customers/{id}` url.

---

1. In order to add buttons to the Admin Customer Show menu in **Sylius** you have to create a `AppBundle\Menu\AdminCustomerShowMenuListener` class.

Note: This menu is build from buttons. There are a few button types available: `edit`, `show`, `delete`, `link` (default), and `transition` (for state machines).

Buttons (except for the `link` and `transition` types) already have a defined color, icon and label. The `link` and `transition` types buttons can be customized with the `setLabel('label')`, `setLabelAttribute('color', 'color')` and `setLabelAttribute('icon', 'icon')` methods.

The `delete` button must have also the `resource_id` attribute set (for csrf token purposes).

In the example below, we are adding one new button to the Admin Customer Show Menu. It has the type set, even though the `link` type is default to make the example easily customizable.

```php
<?php

namespace AppBundle\Menu;
```

```
use Sylius\Bundle\AdminBundle\Event\CustomerShowMenuBuilderEvent;

final class AdminCustomerShowMenuListener
{
    /**
     * @param CustomerShowMenuBuilderEvent $event
     */
    public function addAdminCustomerShowMenuItems(CustomerShowMenuBuilderEvent
↪$event): void
    {
        $menu = $event->getMenu();
        $customer = $event->getCustomer();

        if (null !== $customer->getUser()) {
            $menu
                ->addChild('impersonate', [
                    'route' => 'sylius_admin_impersonate_user',
                    'routeParameters' => ['username' => $customer->getUser()->
↪getEmailCanonical()]
                ])
                ->setAttribute('type', 'link')
                ->setLabel('Impersonate')
                ->setLabelAttribute('icon', 'unhide')
                ->setLabelAttribute('color', 'blue')
            ;
        }
    }
}
```

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the `sylius.menu.admin.customer.show` event in the `AppBundle/Resources/config/services.yml`.

```
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.customer.show.menu_builder:
        class: AppBundle\Menu\AdminCustomerShowMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.admin.customer.show,␣
↪method: addAdminCustomerShowMenuItems }
```

How to customize Admin Order Show Menu?

Tip: Admin order show menu is the set of buttons in the right top corner on the `/admin/orders/{id}` url.

1. In order to add buttons to the Admin Order Show menu in **Sylius** you have to create a `AppBundle\Menu\AdminOrderShowMenuListener` class.

Note: This menu is build from buttons. There are a few button types available: `edit`, `show`, `delete`, `link` (default), and `transition` (for state machines).

Buttons (except for the `link` and `transition` types) already have a defined color, icon and label. The `link` and `transition` types buttons can be customized with the `setLabel('label')`,

setLabelAttribute('color', 'color') and setLabelAttribute('icon', 'icon') methods.

The delete button must have also the resource_id attribute set (for csrf token purposes).

In the example below, we are adding one new button to the Admin Order Show Menu. It is a transition type button, that will let the admin fulfill the order.

> **Warning:** There is no sylius_admin_order_fulfill route in Sylius. Create this route if you need it.

```php
<?php

namespace AppBundle\Menu;

use Sylius\Bundle\AdminBundle\Event\OrderShowMenuBuilderEvent;
use Sylius\Component\Order\OrderTransitions;

final class AdminOrderShowMenuListener
{
    /**
     * @param OrderShowMenuBuilderEvent $event
     */
    public function addAdminOrderShowMenuItems(OrderShowMenuBuilderEvent $event): void
    {
        $menu = $event->getMenu();
        $order = $event->getOrder();
        $stateMachine = $event->getStateMachine();

        if ($stateMachine->can(OrderTransitions::TRANSITION_FULFILL)) {
            $menu
                ->addChild('fulfill', [
                    'route' => 'sylius_admin_order_fulfill',
                    'routeParameters' => ['id' => $order->getId()]
                ])
                ->setAttribute('type', 'transition')
                ->setLabel('Fulfill')
                ->setLabelAttribute('icon', 'checkmark')
                ->setLabelAttribute('color', 'green')
            ;
        }
    }
}
```

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the sylius.menu.admin.order.show event in the AppBundle/Resources/config/services.yml.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.order.show.menu_builder:
        class: AppBundle\Menu\AdminOrderShowMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.admin.order.show,
→method: addAdminOrderShowMenuItems }
```

How to customize Admin Product Form Menu?

Tip: Admin product form menu is the set of tabs on your left hand side on the `/admin/products/new` and `/admin/products/{id}/edit` urls.

Warning: This part of the guide assumes you already know how to customize *models* and *forms*.

---

1. In order to add a new tab to the Admin Product Form menu in **Sylius** you have to create a `AppBundle\Menu\AdminProductFormMenuListener` class.

Note: This menu is build from tabs, each coupled with their own template containing the necessary part of the form.

So lets say you want to add the product's manufacturer details to the tabs. Provided you have created a new template with all the required form fields and saved it etc. as `AppBundle\Resources\views\Admin\Product\Tab\_manufacturer.html.twig`, we will use it in the example below.

```php
<?php

namespace AppBundle\Menu;

use Sylius\Bundle\AdminBundle\Event\ProductMenuBuilderEvent;

final class AdminProductFormMenuListener
{
    /**
     * @param ProductMenuBuilderEvent $event
     */
    public function addItems(ProductMenuBuilderEvent $event): void
    {
        $menu = $event->getMenu();

        $menu
            ->addChild('manufacturer')
            ->setAttribute('template', '@App/Admin/Product/Tab/_manufacturer.html.twig
↪')
            ->setLabel('Manufacturer')
        ;
    }
}
```

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the `sylius.menu.admin.product.form` event in the `AppBundle/Resources/config/services.yml`.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.product.form.menu_builder:
        class: AppBundle\Menu\AdminProductFormMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.admin.product.form,␣
↪method: addItems }
```

How to customize Admin Product Variant Form Menu?

Tip: Admin product variant form menu is the set of tabs on your left hand side on the `/admin/product/{productId}/variants/new` and `/admin/product/{productId}/variants/{id}/edit` urls.

Warning: This part of the guide assumes you already know how to customize *models* and *forms*.

1. In order to add a new tab to the Admin Product Variant Form menu in **Sylius** you have to create a `AppBundle\Menu\AdminProductVariantFormMenuListener` class.

Note: This menu is build from tabs, each coupled with their own template containing the necessary part of the form.

So lets say you want to add the product variant's media to the tabs. Provided you have created a new template with the required form fields and saved it etc. as `AppBundle\Resources\views\Admin\ProductVariant\Tab\_media.html.twig`, we will use it in the example below.

```php
<?php

namespace AppBundle\Menu;

use Sylius\Bundle\AdminBundle\Event\ProductVariantMenuBuilderEvent;

final class AdminProductVariantFormMenuListener
{
    /**
     * @param ProductVariantMenuBuilderEvent $event
     */
    public function addItems(ProductVariantMenuBuilderEvent $event): void
    {
        $menu = $event->getMenu();

        $menu
            ->addChild('media')
            ->setAttribute('template', '@App/Admin/ProductVariant/Tab/_media.html.twig
→')
            ->setLabel('Media')
        ;
    }
}
```

2. After creating your class with a proper method for the menu customizations you need, subscribe your listener to the `sylius.menu.admin.product_variant.form` event in the `AppBundle/Resources/config/services.yml`.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.product_variant.form.menu_builder:
        class: AppBundle\Menu\AdminProductVariantFormMenuListener
        tags:
            - { name: kernel.event_listener, event: sylius.menu.admin.product_variant.
→form, method: addItems }
```

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.8 Customizing Templates

Note: There are two kinds of templates in Sylius. **Shop** and **Admin** ones, plus you can create your own to satisfy your needs.

Why would you customize a template?

The most important case for modifying the existing templates is of course **integrating your own layout of the system**. Sometimes even if you have decided to stay with the default layout provided by Sylius, you need to **slightly modify it to meet your business requirements**. You may just need to **add your logo anywhere**.

Methods of templates customizing

Warning: There are three ways of customizing templates of Sylius:

The first one is simple **templates overriding** inside of the `app/Resources` directory of your project. Using this method you can completely change the content of templates.

The second method is **templates customization via events**. You are able to listen on these template events, and by that add your own blocks without copying and pasting the whole templates. This feature is really useful when *creating Sylius Extensions*.

The third method is **using Sylius themes**. Creating a Sylius theme requires a few more steps than basic template overriding, but allows you to have a different design on multiple channels of the same Sylius instance. *Learn more about themes here*.

How to customize templates by overriding?

Note: How do you know which template you should be overriding? Go to the page that you are going to modify, at the bottom in the Symfony toolbar click on the route, which will redirect you to the profiler. In the Request Attributes section under `_sylius [ template => ...]` you can check the path to the current template.

- **Shop** templates: customizing Login Page template:

The default login template is: `SyliusShopBundle:login.html.twig`. In order to override it you need to create your own: `app/Resources/SyliusShopBundle/views/login.html.twig`.

Copy the contents of the original template to make your work easier. And then modify it to your needs.

```
{% extends '@SyliusShop/layout.html.twig' %}

{% import '@SyliusUi/Macro/messages.html.twig' as messages %}

{% block content %}
<div class="ui column stackable center page grid">
    {% if last_error %}
        {{ messages.error(last_error.messageKey|trans(last_error.messageData,
↪'security')) }}
    {% endif %}

    {# You can add a headline for instance to see if you are changing things in the␣
↪correct place. #}
    <h1>
        This Is My Headline
    </h1>

    <div class="five wide column"></div>
    <form class="ui six wide column form segment" action="{{ path('sylius_shop_login_
↪check') }}" method="post" novalidate>
        <div class="one field">
            {{ form_row(form._username, {'value': last_username|default('')}) }}
        </div>
        <div class="one field">
            {{ form_row(form._password) }}
        </div>
        <div class="one field">
            <button type="submit" class="ui fluid large primary submit button">{{
↪'sylius.ui.login_button'|trans }}</button>
        </div>
    </form>
</div>
{% endblock %}
```

Done! If you do not see any changes on the `/shop/login` url, clear your cache:

```
$ php bin/console cache:clear
```

- **Admin** templates: Customization of the Country form view.

The default template for the Country form is: `SyliusAdminBundle:Country:_form.html.twig`. In order to override it you need to create your own: `app/Resources/SyliusAdminBundle/views/Country/_form.html.twig`.

Copy the contents of the original template to make your work easier. And then modify it to your needs.

```
<div class="ui segment">
    {{ form_errors(form) }}
    {{ form_row(form.code) }}
    {{ form_row(form.enabled) }}
</div>
<div class="ui segment">

    {# You can add a headline for instance to see if you are changing things in the␣
↪correct place. #}
    <h1>My Custom Headline</h1>

    <h4 class="ui dividing header">{{ 'sylius.ui.provinces'|trans }}</h4>
```

(continues on next page)

```
    {{ form_row(form.provinces, {'label': false}) }}
</div>
```

Done! If you do not see any changes on the `/admin/countries/new` url, clear your cache:

```
$ php bin/console cache:clear
```

How to customize templates via events?

Sylius uses the Events mechanism provided by the SonataBlockBundle.

How to locate template events?

The events naming convention uses the routing to the place where we are adding it, but instead of _ we are using ., followed by a slot name (like `sylius_admin_customer_show` route results in the `sylius.admin.customer.show.slot_name` events). The slot name describes where exactly in the template's structure should the event occur, it will be `before` or `after` certain elements.

Although when the resource name is not just one word (like `product_variant`) then the underscore stays in the event prefix string. Then `sylius_admin_product_variant_create` route will have the `sylius.admin.product_variant.create.slot_name` events.

Let's see how the event is rendered in a default Sylius Admin template. This is the rendering of the event that occurs on the create action of Resources, at the bottom of the page (after the content of the create form):

```
{# First we are setting the event_prefix based on route as it was mentioned before #}
{% set event_prefix = metadata.applicationName ~ '.admin.' ~ metadata.name ~ '.create
↪' %}

{# And then the slot name is appended to the event_prefix #}
{{ sonata_block_render_event(event_prefix ~ '.after_content', {'resource': resource})␣
↪}}
```

Note: Besides the events that are named based on routing, Sylius also has some other general events: those that will appear on every Sylius admin or shop. Examples: `sylius.shop.layout.slot_name` or `sylius.admin.layout.slot_name`. They are rendered in the `layout.html.twig` views for both Admin and Shop.

Tip: In order to find events in Sylius templates you can simply search for the `sonata_block_render_event` phrase in your project's directory.

How to use template events for customizations?

When you have found an event in the place where you want to add some content, here's what you have to do.

Let's assume that you would like to add some content after the header in the Sylius shop views. You will need to look at the `/SyliusShopBundle/Resources/views/layout.html.twig` template, which is the basic layout of Sylius shop, and then in it find the appropriate event.

For the space below the header it will be `sylius.shop.layout.after_header`.

- Create an `.html.twig` file that will contain what you want to add.

```
{# AppBundle/Resources/views/block.html.twig #}

<h1> Test Block Title </h1>
```

- And register a listener for the chosen event:

> **Warning:** The name of the event should be preceded by the `sonata.block.event.` string.

```yaml
services:
    app.block_event_listener.homepage.layout.after_header:
        class: Sylius\Bundle\UiBundle\Block\BlockEventListener
        arguments:
            - '@@App/block.html.twig'
        tags:
            - { name: kernel.event_listener, event: sonata.block.event.sylius.shop.
↪layout.after_header, method: onBlockEvent }
```

Tip: While configuring it in `yaml` remember about having two `@` for the argument reference to your template, just like above `'@@App/block.html.twig'`, what escapes the second `@` and lets it not to be interpreted as a service.

In `xml` the double `@` is not required: it would be just `<argument>@App/block.html.twig</argument>`

That's it. Your new block should appear in the view.

Tip: Learn more about adding custom Admin JS & CSS in the cookbook *here*.

How to use themes for customizations?

You can refer to the theme documentation available here: - *Themes (The book)* - *SyliusThemeBundle (Bundle documentation)*

Global Twig variables

Each of the Twig templates in Sylius is provided with the `sylius` variable, that comes from the ShopperContext.

The **ShopperContext** is composed of `ChannelContext`, `CurrencyContext`, `LocaleContext` and `CustomerContext`. Therefore it has access to the current channel, currency, locale and customer.

The variables available in Twig are:

Twig variable	ShopperContext method name
sylius.channel	getChannel()
sylius.currencyCode	getCurrencyCode()
sylius.localeCode	getLocaleCode()
sylius.customer	getCustomer()

How to use these Twig variables?

You can check for example what is the current channel by dumping the `sylius.channel` variable.

```
{{ dump(sylius.channel) }}
```

That's it, this will dump the content of the current Channel object.

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.9 Customizing Translations

Note: We've adopted a convention of overriding translations in the `app\Resources\translations` directory.

Why would you customize a translation?

If you would like to change any of the translation keys defined in Sylius in any desired language.

For example:

- change "Last name" into "Surname"
- change "Add to cart" into "Buy"

There are many other places where you can customize the text content of pages.

How to customize a translation?

In order to customize a translation in your project:

1. If you don't have it yet, create `app\Resources\translations\messages.en.yml` for English translations.

Note: You can create different files for different locales (languages). For example `messages.pl.yml` should hold only Polish translations, as they will be visible when the current locale is `PL`. Check the *Locales* docs for more information.

2. In this file, configure the desired key and give it a translation.

If you would like to change the translation of "Email" into "Username" on the login form you have to override its translation key which is `sylius.form.customer.email`.

```yaml
sylius:
    form:
        customer:
            email: Username
```

Tip: How to check what the proper translation key is for your message: When you are on the page where you are trying to customize a translation, click the Translations icon in the Symfony Profiler. In this section you can see all messages with their associated keys on that page.
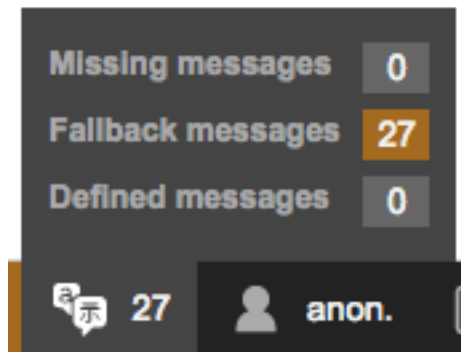
Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.10 Customizing Flashes

Why would you customize a flash?

If you would like to change any of the flash messages defined in Sylius in any desired language.

For example:

- change the content of a flash when you add resource in the admin
- change the content of a flash when you register in the shop

and many other places where you can customize the text content of the default flashes.

How to customize a flash message?

In order to customize a resource flash in your project:

1. Create the `app\Resources\translations\flashes.en.yml` for english contents of your flashes.

Note: You can create different files for different locales (languages). For example `flashes.pl.yml` should hold only polish flashes, as they will be visible when the current locale is `PL`. Check *Locales* docs for more information.

2. In this file configure the desired flash key and give it a translation.

If you would like to change the flash message while updating a Taxon, you will need to configure the flash under the `sylius.taxon.update` key:

```
sylius:
    taxon:
        update: This category has been successfully edited.
```

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

2.1.11 Customizing State Machines

> **Warning:** Not familiar with the State Machine concept? Read the docs *here*!

> **Note: Customizing logic via State Machines vs. Events**
>
> The logic in which Sylius operates can be customized in two ways. First of them is using the state machines: what is really useful when you need to modify business logic for instance modify the flow of the checkout, and the second is listening on the kernel events related to the entities, which is helpful for modifying the HTTP responses visible directly to the user, like displaying notifications, sending emails.

How to customize a State Machine?

> **Tip:** First of all if you are attempting to change anything in any state machine in **Sylius** you will need a special file: `app/config/state_machine.yml` which has to be imported in the `app/config/config.yml`.

```
# app/config/config.yml
imports:
    - { resource: "state_machine.yml" }
```

How to add a new state?

Let's assume that you would like to add a new **state** to *the Order state machine*. You will need to add these few lines to the `state_machine.yml`:

```
# app/config/state_machine.yml
winzou_state_machine:
    sylius_order:
        states:
            your_new_state: ~ # here name your state as you wish
```

After that your new step will be available alongside other steps that already were defined in that state machine.

Tip: Run $ `php bin/console debug:winzou:state-machine sylius_order` to check if the state machine has changed to your implementation.

How to add a new transition?

Let's assume that you would like to add a new **transition** to *the Order state machine*, that will allow moving from the `cancelled` state backwards to `new`. Let's call it "restoring".

You will need to add these few lines to the `state_machine.yml`:

```yaml
# app/config/state_machine.yml
winzou_state_machine:
    sylius_order:
        transitions:
            restore:
                from: [cancelled]
                to: new
```

After that your new transition will be available alongside other transitions that already were defined in that state machine.

Tip: Run $ `php bin/console debug:winzou:state-machine sylius_order` to check if the state machine has changed to your implementation.

How to remove a state and its transitions?

Warning: If you are willing to remove a state or a transition you have to override **the whole states/transitions section** of the state machine you are willing to modify. See how we do it in the *customization of the Checkout process*.

How to add a new callback?

Let's assume that you would like to add a new **callback** to *the Order state machine*, that will do something on an already defined transition.

You will need to add these few lines to the `state_machine.yml`:

```yaml
# app/config/state_machine.yml
winzou_state_machine:
    sylius_order:
        callbacks:
            after:
                sylius_send_email:
                    # here you are choosing the transition on which the action should
↪take place - we are using the one we have created before
                    on: ["cancel"]
                    # it is just an example, use an existent service and its method
↪here!
```

(continues on next page)

```
                    do: ["@service", "sendEmail"]
                    # this will be the object of an Order here
                    args: ["object"]
```

After that your new callback will be available alongside other callbacks that already were defined in that state machine and will be called on the desired transition

How to modify a callback?

If you would like to modify an existent callback of for example the state machine of ProductReviews, so that it does not count the average rating but does something else - you need to add these few lines to the `state_machine.yml`:

```
# app/config/state_machine.yml
winzou_state_machine:
    sylius_review:
        callbacks:
            after:
                update_price:
                    on: "accept"
                    # here you can change the service and its method that is called␣
→for your own service
                    do: ["@sylius.review.updater.your_service", update]
                    args: ["object"]
```

How to disable a callback?

If you would like to turn off a callback of a state machine you need to set its `disabled` option to true. On the example of the state machine of ProductReview, we can turn off the `update_price` callback:

```
# app/config/state_machine.yml
winzou_state_machine:
    sylius_review:
        callbacks:
            after:
                update_price:
                    disabled: true
```

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

Learn more

- Winzou StateMachine Bundle
- *State Machine Concept*

2.1.12 Customizing Grids

Note: We assume that you are familiar with grids. If not check the documentation of the *Grid Bundle* and *Grid Component* first.

Why would you customize grids?

When you would like to change how the index view of an entity looks like in the administration panel, then you have to override its grid.

- remove a field from a grid
- change a field of a grid
- reorder fields
- override an entire grid

How to customize grids?

Tip: One way to change anything in any grid in **Sylius** is to create a special file in the `app/config/` directory: `app/config/grids.yml` which has to be imported in the `app/config/config.yml`.

```yaml
# app/config/config.yml
imports:
    - { resource: "grids.yml" }
```

How to customize fields of a grid?

How to remove a field from a grid?

If you would like to remove a field from an existing Sylius grid, you will need to disable it in the `app/config/grids.yml`.

Let's imagine that we would like to hide the **title of product review** field on the `sylius_admin_product_review` grid.

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product_review:
            fields:
                title:
                    enabled: false
```

That's all. Now the `title` field will be disabled (invisible).

How to modify a field of a grid?

If you would like to modify for instance a label of any field from a grid, that's what you need to do:

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product_review:
            fields:
                date:
                    label: "When was it added?"
```

How to customize filters of a grid?

How to remove a filter from a grid?

If you would like to remove a filter from an existing Sylius grid, you will need to disable it in the `app/config/grids.yml`.

Let's imagine that we would like to hide the **titles filter of product reviews** on the `sylius_admin_product_review` grid.

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product_review:
            filters:
                title:
                    enabled: false
```

That's all. Now the `title` filter will be disabled.

How to customize actions of a grid?

How to remove an action from a grid?

If you would like to disable some actions in any grid, you just need to set its `enabled` option to `false` like below:

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product_review:
            actions:
                item:
                    delete:
                        type: delete
                        enabled: false
```

How to modify an action of a grid?

If you would like to change the link to which an action button is redirecting, this is what you have to do:

> **Warning:** The `show` button does not exist in the `sylius_admin_product` grid by default. It is assumed that
> you already have it customized, and your grid has the `show` action.

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product:
            actions:
                item:
                    show:
                        type: show
                        label: Show in the shop
                        options:
                            link:
                                route: sylius_shop_product_show
                                parameters:
                                    slug: resource.slug
```

The above grid modification will change the redirect of the `show` action to redirect to the shop, instead of admin show.
Also the label was changed here.

How to modify positions of fields, filters and actions in a grid?

For fields, filters and actions it is possible to easily change the order in which they are displayed in the grid.

See an example of fields order modification on the `sylius_admin_product_review` grid below:

```yaml
# app/config/grids.yml
sylius_grid:
    grids:
        sylius_admin_product_review:
            fields:
                date:
                    position: 5
                title:
                    position: 6
                rating:
                    position: 3
                status:
                    position: 1
                reviewSubject:
                    position: 2
                author:
                    position: 4
```

Customizing grids by events

There is also another way to customize grids: **via events**. Every grid configuration dispatches an event when its
definition is being converted.

For example, **sylius_admin_product** grid dispatches such an event:

```
sylius.grid.admin_product # For the grid of products in admin
```

To show you an example of a grid customization using events, we will remove a field from a grid using that method. Here are the steps, that you need to take:

1. In order to remove fields from the product grid in **Sylius** you have to create a `AppBundle\Grid\AdminProductsGridListener` class.

In the example below we are removing the `images` field from the `sylius_admin_product` grid.

```php
<?php

namespace AppBundle\Grid;

use Sylius\Component\Grid\Event\GridDefinitionConverterEvent;

final class AdminProductsGridListener
{
    /**
     * @param GridDefinitionConverterEvent $event
     */
    public function removeImageField(GridDefinitionConverterEvent $event)
    {
        $grid = $event->getGrid();

        $grid->removeField('image');
    }
}
```

2. After creating your class with a proper method for the grid customizations you need, subscribe your listener to the `sylius.grid.admin_product` event in the `AppBundle/Resources/config/services.yml`.

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.listener.admin.products_grid:
        class: AppBundle\Grid\AdminProductsGridListener
        tags:
            - { name: kernel.event_listener, event: sylius.grid.admin_product, method: removeImageField }
```

3. Result:

After these two steps your admin product grid should not have the image field.

How to override an entire grid?

Tip: This is the other way to customize grids. If you need to change more, than just slight adjustments we do recommend to override an entire grid file in the `app/Resources/` directory.

Let's assume that you would like to modify the `shipping_categories` grid by removing filters and the delete action from it.

- To achieve that you need to create the `app/Resources/SyliusAdminBundle/config/grids/shipping_category.yml` file.
- Then into the created file copy the content of `Sylius/Bundle/AdminBundle/Resources/config/grids/shipping_category.yml`.
- And modify it to your needs:

```yaml
# app/Resources/SyliusAdminBundle/config/grids/shipping_category.yml
sylius_grid:
    grids:
        sylius_admin_shipping_category:
            driver:
                name: doctrine/orm
                options:
                    class: "%sylius.model.shipping_category.class%"
                    repository:
                        method: createListQueryBuilder
            fields:
                code:
                    type: string
                    label: sylius.ui.code
                name:
                    type: string
                    label: sylius.ui.name
                createdAt:
                    type: datetime
                    label: sylius.ui.creation_date
                    options:
                        format: d-m-Y H:i
                updatedAt:
                    type: datetime
                    label: sylius.ui.updating_date
                    options:
                        format: d-m-Y H:i
            actions:
                main:
                    create:
                        type: create
                item:
                    update:
                        type: update
```

In the above example the `delete` action and the filters section have been removed.

- That's it! The grid has been modified and it will look like that:

Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

Learn more

- *Grid - Component Documentation*
- *Grid - Bundle Documentation*
- *Customizing Models*
- *Customizing Forms*
- *Customizing Repositories*
- *Customizing Factories*
- *Customizing Controllers*
- *Customizing Validation*
- *Customizing Menus*
- *Customizing Templates*
- *Customizing Translations*
- *Customizing Flashes*
- *Customizing State Machines*
- *Customizing Grids*

2.1.13 Good to know

See also:

All the customizations can be done either in your application directly or in *Plugins*!

- *Customizing Models*
- *Customizing Forms*
- *Customizing Repositories*
- *Customizing Factories*
- *Customizing Controllers*
- *Customizing Validation*
- *Customizing Menus*
- *Customizing Templates*
- *Customizing Translations*
- *Customizing Flashes*
- *Customizing State Machines*
- *Customizing Grids*

Sylius Plugins

The collection of Sylius Plugins together with the guide on Plugins development. Remember that you can use all the *customization techniques* in Plugins.

3.1 Sylius Plugins

Sylius as a platform has a lot of space for various customizations and extensions. It aims to provide a simple schema for developing plugins. Anything you can imagine can be implemented and added to the Sylius framework as a plugin.

3.1.1 The official list of plugins

Tip: **The official list of Sylius Plugins is available on the Sylius website** here.

3.1.2 What are the plugins for?

The plugins either modify or extend Sylius default behaviour, providing useful features that are built on top of the Sylius Core.

Exemplary features may be: Social media buttons, newsletter, wishlists, payment gateways integrations etc.

3.1.3 How to make a Sylius Plugin official?

Since Sylius is an open-source platform, there is a certain flow in order for the plugin to become officially adopted by the community.

1. Develop the plugin using *the official Plugin Development guide*.

2. Remember about the tests and code quality!

---

3. Send it to the project maintainers. It can be via email to any member of the Sylius Core team, or the official Sylius Slack.

4. Wait for your Plugin to be featured the list of plugins on the Sylius website.

How to create a plugin for Sylius?

Sylius plugin is nothing more but a regular Symfony bundle adding custom behaviour to the default Sylius application.

The best way to create your own plugin is to use Sylius plugin skeleton, which has built-in infrastructure for designing and testing using Behat.

1. Create project using Composer.

```
$ composer create-project sylius/plugin-skeleton SyliusMyFirstPlugin
```

Note: The plugin can be created anywhere, not only inside a Sylius application, because it already has the test environment inside.

2. Get familiar with basic plugin design.

The skeleton comes with simple application that greets a customer. There are feature scenarios in `features` directory; exemplary bundle with a controller, a template and a routing configuration in `src`; and the testing infrastructure in `tests`.

Note: The `tests/Application` directory contains a sample Symfony application used to test your plugin.

3. Remove boilerplate files and rename your bundle.

In most cases you don't want your Sylius plugin to greet the customer like it is now, so feel free to remove unnecessary controllers, assets and features. You will also want to change the plugin's namespace from `Acme\SyliusExamplePlugin` to a more meaningful one. Keep in mind that these changes also need to be done in `tests/Application` and `composer.json`.

Tip: Refer to chapter 5 for the naming conventions to be used.

4. Implement your awesome features.

Looking at existing Sylius plugins like

- Sylius/ShopAPIPlugin
- bitbag-commerce/PayUPlugin
- stefandoorn/sitemap-plugin
- bitbag-commerce/CmsPlugin

is a great way to start developing your own plugins.

You are strongly encouraged to use BDD with Behat, phpspec and PhpUnit to ensure your plugin's extraordinary quality.

Tip: For the plugins, the suggested way of modifying Sylius is using *the Customization Guide*. There you will find a lot of help while trying to modify templates, state machines, controllers and many, many more.

---

5. Naming conventions

Besides the way you are creating plugins (based on our skeleton or on your own), there are a few naming conventions that should be followed:

- Repository name should use PascalCase, must have a `Sylius*` prefix and a `Plugin` suffix

- Project composer name should use dashes as a separator, must have a `sylius` prefix and a `plugin` suffix, e.g.: `sylius-invoice-plugin`.

- Bundle class name should start with vendor name, followed by `Sylius` and suffixed by `Plugin` (instead of `Bundle`), e.g.: `VendorNameSyliusInvoicePlugin`.

- Bundle extension should be named similar, but suffixed by the Symfony standard `Extension`, e.g.: `VendorNameSyliusInvoiceExtension`.

- Bundle class must use the `Sylius\Bundle\CoreBundle\Application\SyliusPluginTrait` trait.

- Namespace should follow PSR-4. The top-level namespace should be the vendor name. The second-level should be prefixed by `Sylius` and suffixed by `Plugin` (e.g. `VendorName\SyliusInvoicePlugin`)

Note: Following the naming strategy for the bundle class & extension class prevents configuration key collision. Following the convention mentioned above generates the default configuration key as e.g. `vendor_name_sylius_invoice_plugin`.

The rules are to be applied to all bundles which will provide an integration with the whole Sylius platform (`sylius/sylius` or `sylius/core-bundle` as dependency).

Reusable components for the whole Symfony community, which will be based just on some Sylius bundles should follow the regular Symfony conventions.

Example

Assuming you are creating the invoicing plugin as used above, this will result in the following set-up.

1. Name your repository: `vendor-name/sylius-invoice-plugin`.

2. Create bundle class in `src/VendorNameSyliusInvoicePlugin.php`:

```php
<?php

declare(strict_types=1);

namespace VendorName\SyliusInvoicePlugin;
```

<div align="right">(continues on next page)</div>

```php
use Sylius\Bundle\CoreBundle\Application\SyliusPluginTrait;
use Symfony\Component\HttpKernel\Bundle\Bundle;

final class VendorNameSyliusInvoicePlugin extends Bundle
{
    use SyliusPluginTrait;
}
```

3. Create extension class in `src/DependencyInjection/VendorNameSyliusInvoiceExtension.php`:

```php
<?php

declare(strict_types=1);

namespace VendorName\SyliusInvoicePlugin\DependencyInjection;

use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Extension\Extension;
use Symfony\Component\DependencyInjection\Loader\XmlFileLoader;

final class VendorNameSyliusInvoiceExtension extends Extension
{
    public function load(array $config, ContainerBuilder $container): void
    {
        $config = $this->processConfiguration($this->getConfiguration([], $container),
→ $config);
        $loader = new XmlFileLoader($container, new FileLocator(__DIR__ . '/../
→Resources/config'));
    }
}
```

4. In `composer.json`, define the correct namespacing for the PSR-4 autoloader:

```json
{
    "autoload": {
        "psr-4": {
            "VendorName\\SyliusInvoicePlugin\\": "src/"
        }
    },
    "autoload-dev": {
        "psr-4": {
            "Tests\\VendorName\\SyliusInvoicePlugin\\": "tests/"
        }
    },
}
```

Plugin Development Guide

Sylius plugins are one of the most powerful ways to extend Sylius functionalities. They're not bounded by Sylius release cycle and can be developed quicker and more effectively. They also allow sharing our (developers) work in an open-source community, which is not possible with regular application customizations.

BDD methodology says the most accurate way to explain some process is using an example. With respect to that rule, let's create some simple first plugin together!

Idea

The most important thing is a concept. You should be aware, that not every customization should be made as a plugin for Sylius. If you:

- share the common logic between multiple projects

- think provided feature could be useful for the whole Sylius community and want to share it for free or sell it

then you should definitely consider the creation of a plugin. On the other hand, if:

- your feature is specific for your project

- you don't want to share your work in the community (maybe **yet**)

then don't be afraid to make a regular Sylius customization.

Tip: For needs of this tutorial, we will implement a simple plugin, making it possible to mark a product variant **available on demand**.

How to start?

The first step is to create a new plugin using our `PluginSkeleton`.

```
$ composer create-project sylius/plugin-skeleton IronManSyliusProductOnDemandPlugin
```

Note: Remember about naming convention! Sylius plugin should start with your vendor name, followed by `Sylius` prefix and with `Plugin` suffix at the end. Let's say your vendor name is **IronMan**. Come on **IronMan**, let's create your plugin!

Naming changes

`PluginSkeleton` provides some default classes and configurations. However, they must have some default values and names that should be changed to reflect your plugin functionality. Basing on the vendor and plugin names established above, these are the changes that should be made:

- In `composer.json`:

    - `sylius/plugin-skeleton` -> `iron-man/sylius-product-on-demand-plugin`

    - `Acme example plugin for Sylius.` -> `Plugin allowing to mark product variants as available on demand in Sylius.` (or sth similar)

    - `Acme\\SyliusExamplePlugin\\` -> `IronMan\\SyliusProductOnDemandPlugin\\` (the same changes should be done in namespaces in `src/` directory

    - `Tests\\Acme\\SyliusExamplePlugin\\` -> `Tests\\IronMan\\SyliusProductOnDemandPlugin\\` (the same changes should be done in namespaces in `tests/` directory

- `AcmeSyliusExamplePlugin` should be renamed to `IronManSyliusProductOnDemandPlugin`

- `AcmeSyliusExampleExtension` should be renamed to `IronManSyliusProductOnDemandExtension`

- In `src/DependencyInjection/Configuration.php`:

    - `acme_sylius_example_plugin` -> `iron_man_sylius_product_on_demand_plugin`

- In `tests/Application/app/AppKernel.php`:

  - `\Acme\SyliusExamplePlugin\AcmeSyliusExamplePlugin()` ->
    `\IronMan\SyliusProductOnDemandPlugin\SyliusProductOnDemandPlugin()`

- In `phpspec.yml.dist` (if you want to use PHPSpec in your plugin):

  - `Acme\SyliusExamplePlugin` -> `IronMan\SyliusProductOnDemandPlugin`

That's it! All other files are just a boilerplate to show you what can be done in the Sylius plugin. They can be deleted with no harm:

- All files from `features/` directory

- `src/Controller/GreetingController.php`

- `src/Resources/config/admin_routing.yml`

- `src/Resources/config/shop_routing.yml`

- `src/Resources/public/greeting.js`

- `src/Resources/views/dynamic_greeting.html.twig`

- `src/Resources/views/static_greeting.html.twig`

- All files from `tests/Behat/Page/Shop/` (with corresponding services)

- `tests/Context/Ui/Shop/WelcomeContext.php` (with corresponding service)

You should also delete Behat suite named `greeting_customer` from `tests/Behat/Resources/suites.yml`.

Important: You **don't have to** remove all these files mentioned above. They can be adapted to suit your plugin functionality. However, as they provide default, dummy features only for the presentation reasons, it's just easier to delete them and implement new ones on your own.

Specification

We strongly encourage you to follow our BDD path in implementing Sylius plugins. In fact, proper tests are one of the requirements to *have your plugin officially accepted*.

Attention: Even though we're big fans of our Behat and PHPSpec-based workflow, we do not enforce you to use the same libraries. We strongly believe that properly tested code is the biggest value, but everyone should feel well with their own tests. If you're not familiar with PHPSpec, but know PHPUnit (or anything else) by heart - keep rocking with your favorite tool!

Scenario

Let's start with describing how **marking a product variant available on demand** should work

```
@managing_product_variants
Feature: Marking a variant as available on demand
    In order to inform customer about possibility to order a product variant on demand
    As an Administrator
    I want to be able to mark product variant as available on demand
```

(continues on next page)

```gherkin
    Background:
        Given the store operates on a single channel in "United States"
        And the store has a "Iron Man Suite" configurable product
        And the product "Iron Man Suite" has a "Mark XLVI" variant priced at "$400000"
        And I am logged in as an administrator

    @ui
    Scenario: Marking product variant as available on demand
        When I want to modify the "Mark XLVI" product variant
        And I mark it as available on demand
        And I save my changes
        Then I should be notified that it has been successfully edited
        And this variant should be available on demand
```

What is really important, usually you don't need to implement the whole Behat scenario on your own! In the example above only 2 steps would need a custom implementation. Rest of them can be easily reused from **Sylius** Behat suite.

Important: If you're not familiar with our BDD workflow with Behat, take a look at *our BDD guide*. All Behat configurations (contexts, pages, services, suites etc.) are explained there in details.

Behavior implementation

```php
<?php

declare(strict_types=1);

namespace Tests\IronMan\SyliusProductOnDemandPlugin\Behat\Context\Ui\Admin;

use Behat\Behat\Context\Context;
use IronMan\SyliusProductOnDemandPlugin\Entity\ProductVariantInterface;
use
→Tests\IronMan\SyliusProductOnDemandPlugin\Behat\Page\Ui\Admin\ProductVariantUpdatePageInterface;
→
use Webmozart\Assert\Assert;

final class ManagingProductVariantsContext implements Context
{
    /** @var ProductVariantUpdatePageInterface */
    private $productVariantUpdatePage;

    public function __construct(ProductVariantUpdatePageInterface
→$productVariantUpdatePage)
    {
        $this->productVariantUpdatePage = $productVariantUpdatePage;
    }

    /**
     * @When I mark it as available on demand
     */
    public function markVariantAsAvailableOnDemand(): void
    {
        $this->productVariantUpdatePage->markAsAvailableOnDemand();
```

```php
    }

    /**
     * @Then /^(this variant) should be available on demand$/
     */
    public function thisVariantShouldBeAvailableOnDemand(ProductVariantInterface
↪$productVariant): void
    {
        $this->productVariantUpdatePage->open([
            'id' => $productVariant->getId(),
            'productId' => $productVariant->getProduct()->getId(),
        ]);

        Assert::true($this->productVariantUpdatePage->isAvailableOnDemand());
    }
}
```

First step is done - we have a failing test, that that is going to go green when we implement a desired functionality.

Implementation

The goal of our plugin is simple - we need to extend the `ProductVariant` entity and provide a new flag, that could be set on the product variant form. Following customizations are done just like in the **Sylius Customization Guide**, take a look at *customizing models*, *form* and *template*.

> **Attention:** `PluginSkeleton` is focused on delivering the most friendly and testable environment. That's why in `tests/Application` directory, there is a **tiny Sylius application** placed, with your plugin already used. Thanks to that, you can test your plugin with Behat scenarios **within** Sylius application without installing it to any test app manually! There is, however, one important consequence of such an architecture. **Everything** that should be done by a plugin user (configuration import, templates copying etc.) should also be done in `tests/Application` to simulate the real developer behavior - and therefore make your new features testable.

Model

The only field we need to add is an additional `$availableOnDemand` boolean. We should start with the unit tests (written with PHPSpec, PHPUnit, or any other unit testing tool):

```php
<?php

// spec/Entity/ProductVariantSpec.php

declare(strict_types=1);

namespace spec\IronMan\SyliusProductOnDemandPlugin\Entity;

use IronMan\SyliusProductOnDemandPlugin\Entity\ProductVariantInterface;
use PhpSpec\ObjectBehavior;
use Sylius\Component\Core\Model\ProductVariant;

final class ProductVariantSpec extends ObjectBehavior
{
```

```php
    function it_is_sylius_product_variant(): void
    {
        $this->shouldHaveType(ProductVariant::class);
    }

    function it_implements_product_variant_interface(): void
    {
        $this->shouldImplement(ProductVariantInterface::class);
    }

    function it_can_be_available_on_demand(): void
    {
        $this->isAvailableOnDemand()->shouldReturn(false);

        $this->setAvailableOnDemand(true);
        $this->isAvailableOnDemand()->shouldReturn(true);
    }
}
```

```php
<?php

// src/Entity/ProductVariant.php

declare(strict_types=1);

namespace IronMan\SyliusProductOnDemandPlugin\Entity;

use Sylius\Component\Core\Model\ProductVariant as BaseProductVariant;

class ProductVariant extends BaseProductVariant implements ProductVariantInterface
{
    /** @var bool */
    private $availableOnDemand = false;

    public function setAvailableOnDemand(bool $availableOnDemand): void
    {
        $this->availableOnDemand = $availableOnDemand;
    }

    public function isAvailableOnDemand(): bool
    {
        return $this->availableOnDemand;
    }
}
```

```php
<?php

// src/Entity/ProductVariantInterface.php

declare(strict_types=1);

namespace IronMan\SyliusProductOnDemandPlugin\Entity;

use Sylius\Component\Core\Model\ProductVariantInterface as
→BaseProductVariantInterface;
```

```
interface ProductVariantInterface extends BaseProductVariantInterface
{
    public function setAvailableOnDemand(bool $availableOnDemand): void;

    public function isAvailableOnDemand(): bool;
}
```

Of course you need to remember about entity mapping customization as well:

```yaml
# src/Resources/config/doctrine/ProductVariant.orm.yml

IronMan\SyliusProductOnDemandPlugin\Entity\ProductVariant:
    type: entity
    table: sylius_product_variant
    fields:
        availableOnDemand:
            type: boolean
```

Then our new entity should be configured as a resource model:

```yaml
# src/Resources/config/config.yml

sylius_product:
    resources:
        product_variant:
            classes:
                model: IronMan\SyliusProductOnDemandPlugin\Entity\ProductVariant
```

This configuration should be placed in `src/Resources/config/config.yml`. It also has to be imported (`- { resource: "@IronManSyliusProductOnDemandPlugin/Resources/config/config.yml" }`) in `tests/Application/app/config/config.yml` to make it work in Behat tests. And at the end importing this file should be one of the steps described in plugin installation.

> **Warning:** Remember that if you modify or add some mapping, you should either provide a migration for the plugin user (that could be copied to their migration folder) or mention the requirement of migration generation in the installation instructions!

Form

To make our new field available in Admin panel, a form extension is required:

```php
<?php

// src/Form/Extension/ProductVariantTypeExtension.php

declare(strict_types=1);

namespace IronMan\SyliusProductOnDemandPlugin\Form\Extension;

use Symfony\Component\Form\AbstractTypeExtension;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Sylius\Bundle\ProductBundle\Form\Type\ProductVariantType;
use Symfony\Component\Form\FormBuilderInterface;
```

```php
final class ProductVariantTypeExtension extends AbstractTypeExtension
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder->add('availableOnDemand', CheckboxType::class, [
            'label' => 'iron_man_sylius_product_on_demand_plugin.ui.available_on_
→demand',
        ]);
    }

    public function getExtendedType(): string
    {
        return ProductVariantType::class;
    }
}
```

Translation keys placed in `src/Resources/translations/message.{locale}.yml` will be resolved automatically.

```yaml
# src/Resources/translations/message.en.yml

iron_man_sylius_product_on_demand_plugin:
    ui:
        available_on_demand: Available on demand
```

And in your `services.yml` file:

```yaml
# src/Resources/config/services.yml

services:
    iron_man_sylius_product_on_demand_plugin.form.extension.type.product_variant:
        class:␣
→IronMan\SyliusProductOnDemandPlugin\Form\Extension\ProductVariantTypeExtension
        tags:
            - { name: form.type_extension, extended_type:␣
→Sylius\Bundle\ProductBundle\Form\Type\ProductVariantType }
```

Again, you must remember about importing `src/Resources/config/services.yml` in `tests/Application/app/Resources/config/config.yml`.

Template

The last step is extending the template of a product variant form. It can be done in three ways:

- by overwriting template
- by using sonata block events
- by writing a theme

For the needs of this tutorial, we will go the first way. What's crucial, we need to determine which template should be overwritten. Naming for twig files in Sylius, both in **ShopBundle** and **AdminBundle** are pretty clear and straightforward. In this specific case, the template to override is `src/Sylius/Bundle/AdminBundle/Resources/views/ProductVariant/Tab/_details.html.twig`. It should be copied to `src/Resources/views/SyliusAdminBundle/ProductVariant/Tab/` directory, and additional field should be placed somewhere in the template.

```
{# src/Resources/views/SyliusAdminBundle/ProductVariant/Tab/_details.html.twig #}

{#...#}

<div class="ui segment">
    <h4 class="ui dividing header">{{ 'sylius.ui.inventory'|trans }}</h4>
    {{ form_row(form.onHand) }}
    {{ form_row(form.tracked) }}
    {{ form_row(form.version) }}
    {{ form_row(form.availableOnDemand) }}
</div>

{#...#}
```
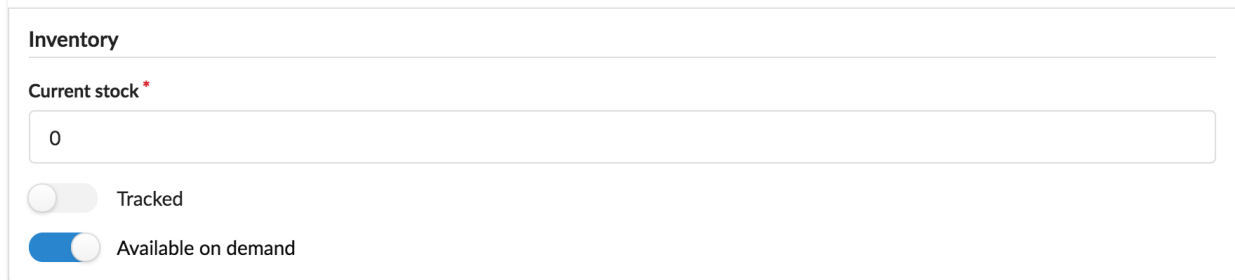
> **Warning:** Beware! Implementing a new template on the plugin level is **not** everything! You must remember that this template should be copied to `app/Resources/views/SyliusAdminBundle/views/` directory (with whole catalogs structure, means `/ProductVariant/Tab` in the application that uses your plugin - and therefore it should be mentioned in installation instruction. The same thing should be done for your test application (you should have `tests/Application/views/SyliusAdminBundle/` catalog with this template copied).
>
> Take a look at *customizing the templates* section in the documentation, for a better understanding of this topic.

Summary

Congratulations! You've created your first, fully tested and documented, customization to Sylius inside a Sylius plugin!

As a result, you should see a new field in product variant form:

As you can see, there are some things to do at the beginning of development, but now, when you are already familiar with the whole structure, each next feature can be provided faster than the previous ones.

What's next?

Of course, it's only the beginning. You could think about plenty of new features associated with this new product variant field. What could be the next step?

- *customizing a product variant grid*, to see new field on the index page
- customizing template of product details page, to show information to customer if product is not available, but can be ordered on demand

- allowing to order **not available yet, but available on demand** variants and therefore customizing the whole *order processing* and *inventory operations*

and even more. The limit is only your imagination (and business value, of course!). For more inspiration, we strongly recommend our *customizing guide*.

At the end, do not hesitate to contact us at contact@sylius.com when you manage to implement a new plugin. We would be happy to check it out and add it to our official plugins list!

Note: Beware, that to have your plugin **officially** accepted, it needs to be created with respect to clean-code principles and properly tested!

Future

We are working hard to make creating Sylius plugins even more developer- and user-friendly. Be in touch with the PluginSkeleton notifications and other announcements from Sylius community. Our plugins base is growing fast - why not be a part of it?

- *How to create a plugin for Sylius?*
- *Plugin Development Guide*
- *How to create a plugin for Sylius?*
- *Plugin Development Guide*

The Cookbook

The Cookbook is a collection of specific solutions for specific needs.

4.1 The Cookbook

The Sylius Cookbook is a collection of solution articles helping you with some specific, narrow problems.

4.1.1 Entities

How to add a custom model?

In some cases you may be needing to add new models to your application in order to cover unique business needs. The process of extending Sylius with new entities is simple and intuitive.

As an example we will take a **Supplier entity**, which may be really useful for shop maintenance.

1. Define your needs

A Supplier needs three essential fields: `name`, `description` and `enabled` flag.

2. Generate the entity

Symfony, the framework Sylius uses, provides the SensioGeneratorBundle, that simplifies the process of adding a model.

> **Warning:** Remember to have the `SensioGeneratorBundle` imported in the AppKernel, as it is not there by default.

You need to use such a command in your project directory.

```
$ php bin/console generate:doctrine:entity
```

The generator will ask you for the entity name and fields. See how it should look like to match our assumptions.

```
  Welcome to the Doctrine2 entity generator


This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: AppBundle:Supplier

Determine the format to use for the mapping information.

Configuration format (yml, xml, php, or annotation) [annotation]: yml

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): name
Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]: true

New field name (press <return> to stop adding fields): description
Field type [string]: text
Is nullable [false]: true
Unique [false]:

New field name (press <return> to stop adding fields): enabled
Field type [string]: boolean
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields):


  Entity generation


> Generating entity class src/AppBundle/Entity/Supplier.php: OK!
> Generating repository class src/AppBundle/Repository/SupplierRepository.php: OK!
> Generating mapping file src/AppBundle/Resources/config/doctrine/Supplier.orm.yml: OK!


  Everything is OK! Now get to work :).
```

3. Update the database using migrations

Assuming that your database was up-to-date before adding the new entity, run:

```
$ php bin/console doctrine:migrations:diff
```

This will generate a new migration file which adds the Supplier entity to your database. Then update the database using the generated migration:

```
$ php bin/console doctrine:migrations:migrate
```

4. Add ResourceInterface to your model class

Go to the generated class file and make it implement the `ResourceInterface`:

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\ResourceInterface;

class Supplier implements ResourceInterface
{
    // ...
}
```

5. Register your entity as a Sylius resource

If you don't have it yet create a file `app/config/resources.yml`, import it in the `app/config/config.yml`.

```yaml
# app/config/config.yml
imports:
    - { resource: "resources.yml" }
```

And add these few lines in the `resources.yml` file:

```yaml
# app/config/resources.yml
sylius_resource:
    resources:
        app.supplier:
            driver: doctrine/orm # You can use also different driver here
            classes:
                model: AppBundle\Entity\Supplier
```

To check if the process was run correctly run such a command:

```
$ php bin/console debug:container | grep supplier
```

The output should be:

```
app.controller.supplier          Sylius\Bundle\ResourceBundle\Controller\ResourceController
app.factory.supplier             Sylius\Component\Resource\Factory\Factory
app.form.type.supplier           Sylius\Bundle\ResourceBundle\Form\Type\DefaultResourceType
app.manager.supplier             alias for "doctrine.orm.default_entity_manager"
app.repository.supplier          Sylius\Bundle\ResourceBundle\Doctrine\ORM\EntityRepository
```

6. Optionally try to use Sylius API to create new resource

See how to work with API in *the separate cookbook here*.

Note: Using API is not mandatory. It is just a nice moment for you to try it out. If you are not interested go to the next point of this cookbook.

---

7. Define grid structure for the new entity

To have templates for your Entity administration out of the box you can use Grids. Here you can see how to configure a grid for the Supplier entity.

```yaml
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
            driver:
                name: doctrine/orm
                options:
                    class: AppBundle\Entity\Supplier
            fields:
                name:
                    type: string
                    label: sylius.ui.name
                description:
                    type: string
                    label: sylius.ui.description
                enabled:
                    type: twig
                    label: sylius.ui.enabled
                    options:
                        template: "@SyliusUi/Grid/Field/enabled.html.twig"
            actions:
                main:
                    create:
                        type: create
                item:
                    update:
                        type: update
                    delete:
                        type: delete
```

Remember to import your grid in the `app/config/grids/grids.yml` file which has to be imported in the `app/config/config.yml`.

```yaml
# app/config/grids/grids.yml
imports:
    - { resource: 'admin/supplier.yml' }
```

```yaml
# app/config/config.yml
imports:
    - { resource: "grids/grids.yml" }
```

8. Define routing for entity administration

Having a grid prepared we can configure routing for the entity administration:

Create the `app/config/routing/admin/supplier.yml` file. Include it in the `app/config/routing/admin.yml`, which should be also included in the `app/config/routing.yml`.

```yaml
# app/config/routing/admin/supplier.yml
app_admin_supplier:
    resource: |
        alias: app.supplier
        section: admin
        templates: SyliusAdminBundle:Crud
        redirect: update
        grid: app_admin_supplier
        vars:
            all:
                subheader: app.ui.supplier
            index:
                icon: 'file image outline'
    type: sylius.resource
```

```yaml
# app/config/routing/admin.yml
app_admin_supplier:
    resource: 'admin/supplier.yml'
```

```yaml
# app/config/routing.yml
app_admin:
    resource: 'routing/admin.yml'
    prefix: /admin
```

9. Add entity administration to the admin menu

Tip: See *how to add links to your new entity administration in the administration menu*.

9. Check the admin panel for your changes

Tip: To see what you can do with your new entity access the `http://localhost:8000/admin/suppliers/` url.

Learn more

- *GridBundle documentation*
- *ResourceBundle documentation*
- *Customization Guide*

How to add a custom translatable model?

In this guide we will create a new translatable model in our system, which is quite similar to *adding a simple model*, although it requires some additional steps.

As an example we will take a **translatable Supplier entity**, which may be really useful for shop maintenance.

1. Define your needs

A Supplier needs three essential fields: `name`, `description` and `enabled` flag. The **name and description** fields need to be translatable.

2. Generate the SupplierTranslation entity

Symfony, the framework Sylius uses, provides the SensioGeneratorBundle, that simplifies the process of adding a model.

> **Warning:** Remember to have the `SensioGeneratorBundle` imported in the AppKernel, as it is not there by default.

You need to use such a command in your project directory.

```
$ php bin/console generate:doctrine:entity
```

The generator will ask you for the entity name and fields. See how it should look like to match our assumptions.

As you can see we have provided only the desired translatable fields.

Below the final `SupplierTranslation` class is presented, it implements the `ResourceInterface`.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\AbstractTranslation;
```

```php
use Sylius\Component\Resource\Model\ResourceInterface;

class SupplierTranslation extends AbstractTranslation implements ResourceInterface
{
    /**
     * @var int
     */
    private $id;

    /**
     * @var string
     */
    private $name;

    /**
     * @var string
     */
    private $description;

    /**
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param string $name
     */
    public function setName($name)
    {
        $this->name = $name;
    }

    /**
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * @param string $description
     */
    public function setDescription($description)
    {
        $this->description = $description;
    }

    /**
     * @return string
     */
    public function getDescription()
    {
        return $this->description;
```

```
    }
}
```

3. Generate the Supplier entity

While generating the entity, similarly to the way the translation was generated, we are providing only non-translatable fields. In our case only the `enabled` field.

```
Welcome to the Doctrine2 entity generator


This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: AppBundle:Supplier

Determine the format to use for the mapping information.

Configuration format (yml, xml, php, or annotation) [annotation]: yml

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): enabled
Field type [string]: boolean
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields):


  Entity generation


  created ./src/AppBundle/Entity/Supplier.php
  created ./src/AppBundle/Resources/config/doctrine/Supplier.orm.yml
> Generating entity class src/AppBundle/Entity/Supplier.php: OK!
> Generating repository class src/AppBundle/Repository/SupplierRepository.php: OK!
> Generating mapping file src/AppBundle/Resources/config/doctrine/Supplier.orm.yml: OK!


  Everything is OK! Now get to work :).
```

Having the stubs generated, we need to extend our class with a connection to SupplierTranslation.

- implement the `ResourceInterface`,

- implement the `TranslatableInterface`,

- use the `TranslatableTrait`,

- initialize the translations collection in the constructor,

- add the `createTranslation()` method,

- implement getters and setters for the properties that are held on the translation model.

As a result you should get such a `Supplier` class:

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\ResourceInterface;
use Sylius\Component\Resource\Model\TranslatableInterface;
use Sylius\Component\Resource\Model\TranslatableTrait;

class Supplier implements ResourceInterface, TranslatableInterface
{
    use TranslatableTrait {
        __construct as private initializeTranslationsCollection;
    }

    public function __construct()
    {
        $this->initializeTranslationsCollection();
    }

    /**
     * @var int
     */
    private $id;

    /**
     * @var bool
     */
    private $enabled;

    /**
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param string $name
     */
    public function setName($name)
    {
        $this->getTranslation()->setName($name);
    }
```

```php
    /**
     * @return string
     */
    public function getName()
    {
        return $this->getTranslation()->getName();
    }

    /**
     * @param string $description
     */
    public function setDescription($description)
    {
        $this->getTranslation()->setDescription($description);
    }

    /**
     * @return string
     */
    public function getDescription()
    {
        return $this->getTranslation()->getDescription();
    }

    /**
     * @param boolean $enabled
     */
    public function setEnabled($enabled)
    {
        $this->enabled = $enabled;
    }

    /**
     * @return bool
     */
    public function getEnabled()
    {
        return $this->enabled;
    }

    /**
     * {@inheritdoc}
     */
    protected function createTranslation()
    {
        return new SupplierTranslation();
    }
}
```

4. Update the database using migrations

Assuming that your database was up-to-date before adding the new entity, run:

```
$ php bin/console doctrine:migrations:diff
```

This will generate a new migration file which adds the Supplier entity to your database. Then update the database

using the generated migration:

```
$ php bin/console doctrine:migrations:migrate
```

5. Register your entity together with translation as a Sylius resource

If you don't have it yet create a file `app/config/resources.yml`, import it in the `app/config/config.yml`.

```yaml
# app/config/config.yml
imports:
    - { resource: "resources.yml" }
```

And add these few lines in the `resources.yml` file:

```yaml
# app/config/resources.yml
sylius_resource:
    resources:
        app.supplier:
            driver: doctrine/orm # You can use also different driver here
            classes:
                model: AppBundle\Entity\Supplier
            translation:
                classes:
                    model: AppBundle\Entity\SupplierTranslation
```

To check if the process was run correctly run such a command:

```
$ php bin/console debug:container | grep supplier
```

The output should be:

```
app.controller.supplier                     Sylius\Bundle\ResourceBundle\Controller\ResourceController
app.controller.supplier_translation         Sylius\Bundle\ResourceBundle\Controller\ResourceController
app.factory.supplier                        Sylius\Component\Resource\Factory\Factory
app.factory.supplier_translation            Sylius\Component\Resource\Factory\Factory
app.manager.supplier                        alias for "doctrine.orm.default_entity_manager"
app.manager.supplier_translation            alias for "doctrine.orm.default_entity_manager"
app.repository.supplier                      Sylius\Bundle\ResourceBundle\Doctrine\ORM\EntityRepository
app.repository.supplier_translation          Sylius\Bundle\ResourceBundle\Doctrine\ORM\EntityRepository
```

6. Prepare new forms for your entity, that will be aware of its translation

You will need both `SupplierType` and `SupplierTranslationType`.

Let's start with the translation type, as it will be included into the entity type.

```php
<?php

namespace AppBundle\Form\Type;

use Sylius\Bundle\ResourceBundle\Form\Type\AbstractResourceType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;

class SupplierTranslationType extends AbstractResourceType
```

(continues on next page)

```php
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class)
            ->add('description', TextareaType::class, [
                'required' => false,
            ])
        ;
    }

    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix()
    {
        return 'app_supplier_translation';
    }
}
```

On the `SupplierTranslationType` we need to define only the translatable fields.

Then let's prepare the entity type, that will include the translation type.

```php
<?php

namespace AppBundle\Form\Type;

use Sylius\Bundle\ResourceBundle\Form\Type\AbstractResourceType;
use Sylius\Bundle\ResourceBundle\Form\Type\ResourceTranslationsType;
use Sylius\Component\Resource\Translation\Provider\TranslationLocaleProviderInterface;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;

class SupplierType extends AbstractResourceType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('translations', ResourceTranslationsType::class, [
                'entry_type' => SupplierTranslationType::class,
            ])
            ->add('enabled', CheckboxType::class, [
                'required' => false,
            ])
        ;
    }

    /**
```

```
     * {@inheritdoc}
     */
    public function getBlockPrefix()
    {
        return 'app_supplier';
    }
}
```

7. Register the new forms as services

Before the newly created forms will be ready to use them, they need to be registered as services:

```yaml
# AppBundle/Resources/config/services.yml
services:
    app.supplier.form.type:
        class: AppBundle\Form\Type\SupplierType
        tags:
            - { name: form.type }
        arguments: ['%app.model.supplier.class%', ['sylius']]
    app.supplier_translation.form.type:
        class: AppBundle\Form\Type\SupplierTranslationType
        tags:
            - { name: form.type }
        arguments: ['%app.model.supplier_translation.class%', ['sylius']]
```

8. Register the forms as resource forms of the Supplier entity

Extend the resource configuration of the `app.supplier` with forms:

```yaml
# app/config/resources.yml
sylius_resource:
    resources:
        app.supplier:
            driver: doctrine/orm # You can use also different driver here
            classes:
                model: AppBundle\Entity\Supplier
                form: AppBundle\Form\Type\SupplierType
            translation:
                classes:
                    model: AppBundle\Entity\SupplierTranslation
                    form: AppBundle\Form\Type\SupplierTranslationType
```

9. Define grid structure for the new entity

To have templates for your Entity administration out of the box you can use Grids. Here you can see how to configure a grid for the Supplier entity.

```yaml
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
```

```yaml
        driver:
            name: doctrine/orm
            options:
                class: AppBundle\Entity\Supplier
        fields:
            name:
                type: string
                label: sylius.ui.name
                sortable: translation.name
            enabled:
                type: twig
                label: sylius.ui.enabled
                options:
                    template: "@SyliusUi/Grid/Field/enabled.html.twig"
        actions:
            main:
                create:
                    type: create
            item:
                update:
                    type: update
                delete:
                    type: delete
```

Remember to import your grid in the `app/config/grids/grids.yml` file which has to be imported in the `app/config/config.yml`.

```yaml
# app/config/grids/grids.yml
imports:
    - { resource: 'admin/supplier.yml' }
```

```yaml
# app/config/config.yml
imports:
    - { resource: "grids/grids.yml" }
```

10. Create template

```twig
# AppBundle/Resources/views/Supplier/_form.html.twig
{% from '@SyliusAdmin/Macro/translationForm.html.twig' import translationForm %}

{{ form_errors(form) }}
{{ translationForm(form.translations) }}
{{ form_row(form.enabled) }}
```

11. Define routing for entity administration

Having a grid prepared we can configure routing for the entity administration:

Create the `app/config/routing/admin/supplier.yml` file. Include it in the `app/config/routing/admin/admin.yml`, which should be also included in the `app/config/routing.yml`.

```
# app/config/routing/admin/supplier.yml
app_admin_supplier:
    resource: |
        alias: app.supplier
        section: admin
        templates: SyliusAdminBundle:Crud
        redirect: update
        grid: app_admin_supplier
        vars:
            all:
                subheader: app.ui.supplier
                templates:
                    form: AppBundle:Supplier:_form.html.twig
            index:
                icon: 'file image outline'
    type: sylius.resource
```

```
# app/config/routing/admin.yml
app_admin_supplier:
    resource: 'admin/supplier.yml'
```

```
# app/config/routing.yml
app_admin:
    resource: 'routing/admin.yml'
    prefix: /admin
```

12. Add entity administration to the admin menu

Tip: See *how to add links to your new entity administration in the administration menu*.

13. Check the admin panel for your changes

Tip: To see what you can do with your new entity access the `http://localhost:8000/admin/suppliers/` url.

Learn more

- *GridBundle documentation*
- *ResourceBundle documentation*
- *Customization Guide*
- *How to add a custom model?*
- *How to add a custom translatable model?*

4.1.2 Api

How to use Sylius API?

In some cases you may be needing to manipulate the resources of your application via its API. This guide aims to introduce you to the world of Sylius API. For more sophisticated examples and cases follow the *API Guide*.

Authentication

Creating OAuth client:

```
$ php bin/console sylius:oauth-server:create-client --grant-type="password" --grant-
→type="refresh_token" --grant-type="token"
```

It will give you such a response:

```
A new client with public id XYZ, secret ABC has been added
```

Run your application on a built-in server:

```
$ php bin/console server:start localhost:8000
```

Tip:

Some test fixtures are provided with a default Sylius fixture suite(which can be obtain by executing: `$ php bin/console sy`

- Sample user: api@example.com

- Sample password: sylius-api

- Sample random client: demo_client

- Sample client secret: demo_client

- Sample access token: SampleToken

To obtain authorization token for the default user run:

```
$ curl http://localhost:8000/api/oauth/v2/token -d "client_id"=XYZ -d "client_secret
→"=ABC -d "grant_type"=password -d "username"=api@example.com -d "password"=sylius-
→api
```

This will give you such a response:

```
{"access_token":"DEF","expires_in":3600,"token_type":"bearer","scope":null,"refresh_
→token":"GHI"}
```

Creating a new resource instance via API

Use the `access_token` in the request that will create a new Supplier (*that we were creating in another cookbook*).

```
$ curl -i -X POST -H "Content-Type: application/json" -H "Authorization: Bearer DEF" -
→d '{"name": "Example", "description": "Lorem ipsum", "enabled": true}' http://
→localhost:8000/api/suppliers/
```

Tip: Read more about authorizing in API *here*.

Viewing a single resource instance via API

If you would like to show details of a resource use this command with object's id as `{id}`. Remember to use **the authorization token**.

Assuming that you have created a supplier in the previous step, it will have id = 1.

```
$ curl -i -H "Authorization: Bearer DEF" http://localhost:8000/api/suppliers/{id}
```

Viewing an index of resource via API

If you would like to see a list of all instances of your resource use such a command (provide the authorization token!):

```
$ curl -i -H "Authorization: Bearer DEF" http://localhost:8000/api/suppliers/
```

Updating a single resource instance via API

If you would like to modify the whole existing resource use such a command (with a valid authorization token of course):

```
$ curl -i -X PUT -H "Content-Type: application/json" -H "Authorization: Bearer DEF" -
→d '{"name": "Modified Name", "description": "Modified description", "enabled":␣
→false}' http://localhost:8000/api/suppliers/1
```

Partially updating a single resource instance via API

If you would like to update just one field of a resource use the PATCH method with such a command:

```
$ curl -i -X PATCH -H "Content-Type: application/json" -H "Authorization: Bearer DEF"␣
→-d '{"enabled": true}' http://localhost:8000/api/suppliers/1
```

Deleting a single resource instance via API

To delete a resource instance you need to call such a command (with an authorization token):

```
$ curl -i -X DELETE -H "Authorization: Bearer DEF" http://localhost:8000/artists/1
```

Learn more

- *API Guide*
- *ResourceBundle documentation*
- *Customization Guide*

> • The Lionframe docs
>
> • *How to use Sylius API?*

4.1.3 Shop

How to customize Sylius Checkout?

Why would you override the Checkout process?

This is a common problem for many Sylius users. Sometimes the checkout process we have designed is not suitable for your custom business needs. Therefore you need to learn how to modify it, when you will need to for example:

- remove shipping step - when you do not ship the products you sell,

- change the order of checkout steps,

- merge shipping and addressing step into one common step,

- or even make the whole checkout a one page process.

See how to do these things below:

How to remove a step from checkout?

Let's imagine that you are trying to create a shop that does not need shipping - it sells downloadable files only.

To meet your needs you will need to adjust checkout process. **What do you have to do then?** See below:

Overwrite the state machine of Checkout

Open the CoreBundle/Resources/config/app/state_machine/sylius_order_checkout.yml and place its content in the `app/Resources/SyliusCoreBundle/config/app/state_machine/sylius_order_checkout.yml` which is a standard procedure of overriding configs in Symfony. Remove the `shipping_selected` and `shipping_skipped` states, `select_shipping` and `skip_shipping` transitions. Remove the `select_shipping` and `skip_shipping` transition from the `sylius_process_cart` callback.

```yaml
# app/Resources/SyliusCoreBundle/config/app/state_machine/sylius_order_checkout.yml
winzou_state_machine:
    sylius_order_checkout:
        class: "%sylius.model.order.class%"
        property_path: checkoutState
        graph: sylius_order_checkout
        state_machine_class: "%sylius.state_machine.class%"
        states:
            cart: ~
            addressed: ~
            payment_skipped: ~
            payment_selected: ~
            completed: ~
        transitions:
            address:
                from: [cart, addressed, payment_selected, payment_skipped]
                to: addressed
            skip_payment:
```

(continues on next page)

```yaml
                    from: [addressed]
                    to: payment_skipped
                select_payment:
                    from: [addressed, payment_selected]
                    to: payment_selected
                complete:
                    from: [payment_selected, payment_skipped]
                    to: completed
        callbacks:
            after:
                sylius_process_cart:
                    on: ["address", "select_payment"]
                    do: ["@sylius.order_processing.order_processor", "process"]
                    args: ["object"]
                sylius_create_order:
                    on: ["complete"]
                    do: ["@sm.callback.cascade_transition", "apply"]
                    args: ["object", "event", "'create'", "'sylius_order'"]
                sylius_save_checkout_completion_date:
                    on: ["complete"]
                    do: ["object", "completeCheckout"]
                    args: ["object"]
                sylius_skip_shipping:
                    on: ["address"]
                    do: ["@sylius.state_resolver.order_checkout", "resolve"]
                    args: ["object"]
                    priority: 1
                sylius_skip_payment:
                    on: ["address"]
                    do: ["@sylius.state_resolver.order_checkout", "resolve"]
                    args: ["object"]
                    priority: 1
```

Tip: To check if your new state machine configuration is overriding the old one run: `$ php bin/console debug:winzou:state-machine` and check the configuration of `sylius_order_checkout`.

Adjust Checkout Resolver

The next step of customizing Checkout is to adjust the Checkout Resolver to match the changes you have made in the state machine. Make these changes in the `config.yml`.

```yaml
# app/config/config.yml
sylius_shop:
    checkout_resolver:
        pattern: /checkout/.+
        route_map:
            cart:
                route: sylius_shop_checkout_address
            addressed:
                route: sylius_shop_checkout_select_payment
            payment_selected:
                route: sylius_shop_checkout_complete
```

```
        payment_skipped:
            route: sylius_shop_checkout_complete
```

Adjust Checkout Templates

After you have got the resolver adjusted, modify the templates for checkout. You have to remove shipping from steps and disable the hardcoded ability to go back to the shipping step and the number of steps being displayed in the checkout navigation. You will achieve that by overriding two files:

- ShopBundle/Resources/views/Checkout/_steps.html.twig

- ShopBundle/Resources/views/Checkout/SelectPayment/_navigation.html.twig

```twig
{# app/Resources/SyliusShopBundle/views/Checkout/_steps.html.twig #}
{% if active is not defined or active == 'address' %}
    {% set steps = {'address': 'active', 'select_payment': 'disabled', 'complete':
↪'disabled'} %}
{% elseif active == 'select_payment' %}
    {% set steps = {'address': 'completed', 'select_payment': 'active', 'complete':
↪'disabled'} %}
{% else %}
    {% set steps = {'address': 'completed', 'select_payment': 'completed', 'complete
↪': 'active'} %}
{% endif %}

{% set order_requires_payment = sylius_is_payment_required(order) %}

{% set steps_count = 'three' %}
{% if not order_requires_payment %}
    {% set steps_count = 'two' %}
{% endif %}

<div class="ui {{ steps_count }} steps">
    <a class="{{ steps['address'] }} step" href="{{ path('sylius_shop_checkout_address
↪') }}">
        <i class="map icon"></i>
        <div class="content">
            <div class="title">{{ 'sylius.ui.address'|trans }}</div>
            <div class="description">{{ 'sylius.ui.fill_in_your_billing_and_shipping_
↪addresses'|trans }}</div>
        </div>
    </a>
    {% if order_requires_payment %}
    <a class="{{ steps['select_payment'] }} step" href="{{ path('sylius_shop_checkout_
↪select_payment') }}">
        <i class="payment icon"></i>
        <div class="content">
            <div class="title">{{ 'sylius.ui.payment'|trans }}</div>
            <div class="description">{{ 'sylius.ui.choose_how_you_will_pay'|trans }}</
↪div>
        </div>
    </a>
    {% endif %}
    <div class="{{ steps['complete'] }} step" href="{{ path('sylius_shop_checkout_
↪complete') }}">
```

```
        <i class="checkered flag icon"></i>
        <div class="content">
            <div class="title">{{ 'sylius.ui.complete'|trans }}</div>
            <div class="description">{{ 'sylius.ui.review_and_confirm_your_order
↪'|trans }}</div>
        </div>
    </div>
</div>
```

```
{# app/Resources/SyliusShopBundle/views/Checkout/SelectPayment/_navigation.html.twig
↪#}
{% set enabled = order.payments|length %}

<div class="ui two column grid">
    <div class="column">
        <a href="{{ path('sylius_shop_checkout_address') }}" class="ui large icon
↪labeled button"><i class="arrow left icon"></i> {{ 'sylius.ui.change_address'|trans
↪}}</a>
    </div>
    <div class="right aligned column">
        <button type="submit" id="next-step" class="ui large primary icon labeled{%
↪if not enabled %} disabled{% endif %} button">
            <i class="arrow right icon"></i>
            {{ 'sylius.ui.next'|trans }}
        </button>
    </div>
</div>
```

Overwrite routing for Checkout

Unfortunately there is no better way - you have to overwrite the whole routing for Checkout.
To do that copy the content of ShopBundle/Resources/config/routing/checkout.yml to the `app/`
`Resources/SyliusShopBundle/config/routing/checkout.yml` file. **Remove routing** of
`sylius_shop_checkout_select_shipping`. The rest should remain the same.

```yaml
# app/Resources/SyliusShopBundle/config/routing/checkout.yml
sylius_shop_checkout_start:
    path: /
    methods: [GET]
    defaults:
        _controller: FrameworkBundle:Redirect:redirect
        route: sylius_shop_checkout_address

sylius_shop_checkout_address:
    path: /address
    methods: [GET, PUT]
    defaults:
        _controller: sylius.controller.order:updateAction
        _sylius:
            event: address
            flash: false
            template: SyliusShopBundle:Checkout:address.html.twig
            form:
                type: Sylius\Bundle\CoreBundle\Form\Type\Checkout\AddressType
```

```yaml
                options:
                    customer: expr:service('sylius.context.customer').getCustomer()
            repository:
                method: find
                arguments:
                    - "expr:service('sylius.context.cart').getCart()"
            state_machine:
                graph: sylius_order_checkout
                transition: address

sylius_shop_checkout_select_payment:
    path: /select-payment
    methods: [GET, PUT]
    defaults:
        _controller: sylius.controller.order:updateAction
        _sylius:
            event: payment
            flash: false
            template: SyliusShopBundle:Checkout:selectPayment.html.twig
            form: Sylius\Bundle\CoreBundle\Form\Type\Checkout\SelectPaymentType
            repository:
                method: find
                arguments:
                    - "expr:service('sylius.context.cart').getCart()"
            state_machine:
                graph: sylius_order_checkout
                transition: select_payment

sylius_shop_checkout_complete:
    path: /complete
    methods: [GET, PUT]
    defaults:
        _controller: sylius.controller.order:updateAction
        _sylius:
            event: complete
            flash: false
            template: SyliusShopBundle:Checkout:complete.html.twig
            repository:
                method: find
                arguments:
                    - "expr:service('sylius.context.cart').getCart()"
            state_machine:
                graph: sylius_order_checkout
                transition: complete
            redirect:
                route: sylius_shop_order_pay
                parameters:
                    tokenValue: resource.tokenValue
            form:
                type: Sylius\Bundle\CoreBundle\Form\Type\Checkout\CompleteType
                options:
                    validation_groups: 'sylius_checkout_complete'
```

Tip: If you do not see any changes run `$ php bin/console cache:clear`.

Learn more

- *Checkout - concept Documentation*
- *State Machine - concept Documentation*
- *Customization Guide*

How to change a redirect after the add to cart action?

Currently **Sylius** by default is using route definition and **sylius-add-to-cart.js** script to handle redirect after successful add to cart action.

```yaml
sylius_shop_partial_cart_add_item:
    path: /add-item
    methods: [GET]
    defaults:
        _controller: sylius.controller.order_item:addAction
        _sylius:
            template: $template
            factory:
                method: createForProduct
                arguments: [expr:service('sylius.repository.product').find(
↪$productId)]
            form:
                type: Sylius\Bundle\CoreBundle\Form\Type\Order\AddToCartType
                options:
                    product: expr:service('sylius.repository.product').find(
↪$productId)
            redirect:
                route: sylius_shop_cart_summary
                parameters: {}
```

```javascript
$.fn.extend({
    addToCart: function () {
        var element = $(this);
        var href = $(element).attr('action');
        var redirectUrl = $(element).data('redirect');
        var validationElement = $('#sylius-cart-validation-error');

        $(element).api({
            method: 'POST',
            on: 'submit',
            cache: false,
            url: href,
            beforeSend: function (settings) {
                settings.data = $(this).serialize();

                return settings;
            },
            onSuccess: function (response) {
                validationElement.addClass('hidden');
                window.location.replace(redirectUrl);
            },
            onFailure: function (response) {
                validationElement.removeClass('hidden');
                var validationMessage = '';
```

(continues on next page)

```
                $.each(response.errors.errors, function (key, message) {
                    validationMessage += message;
                });
                validationElement.html(validationMessage);
                $(element).removeClass('loading');
            },
        });
    }
});
```

If you want to have custom logic after cart add action you can use **ResourceControllerEvent** to set your custom response.

Let's assume that you would like such a feature in your system:

```php
<?php

final class ChangeRedirectAfterAddingToCartListener
{
    /**
     * @var RouterInterface
     */
    private $router;

    /**
     * @param RouterInterface $router
     */
    public function __construct(RouterInterface $router)
    {
        $this->router = $router;
    }

    /**
     * @param ResourceControllerEvent $event
     */
    public function onSuccessfulAddToCart(ResourceControllerEvent $event)
    {
        if (!$event->getSubject() instanceof OrderItemInterface) {
            throw new \LogicException(
                sprintf('This listener operates only on order item, got "$s"', get_
→class($event->getSubject()))
            );
        }

        $newUrl = $this->router->generate('your_new_route_name', []);

        $event->setResponse(new RedirectResponse($newUrl));
    }
}
```

```xml
<service id="sylius.listener.change_redirect_after_adding_to_cart" class=
→"Sylius\Bundle\ShopBundle\EventListener\ChangeRedirectAfterAddingToCartListener">
    <argument type="service" id="router" />
    <tag name="kernel.event_listener" event="sylius.order_item.post_add" method=
→"onSuccessfulAddToCart" />
</service>
```

Next thing to do is handling it by your frontend application.

How to disable guest checkout?

Sometimes, depending on your use case, you may want to resign from the guest checkout feature provided by Sylius.

In order to require users to have an account and be logged in before they can make an order in your shop, you have to turn on the firewalls on the `/checkout` urls.

To achieve that simple add this path to `access_control` in the `security.yml` file.

```yaml
# app/config/security.yml
security:
    access_control:
        - { path: "%sylius.security.shop_regex%/checkout", role: ROLE_USER }
```

That will do the trick. Now, when a guest user tries to click the checkout button in the cart, they will be redirected to the login/registration page, where after they sign in/sign up they will be redirected to the checkout addressing step.

Learn more

- *Sylius Checkout*

How to disable localised URLs?

URLs in Sylius are localised, this means they contain the `/locale` prefix with the current locale. For example when the `English (United States)` locale is currently chosen in the channel, the URL of homepage will look like that `localhost:8000/en_US/`.

If you do not need localised URLs, this guide will help you to disable this feature.

1. Customise the application routing in the `app/config/routing.yml`.

Replace:

```yaml
# app/config/routing.yml

sylius_shop:
    resource: "@SyliusShopBundle/Resources/config/routing.yml"
    prefix: /{_locale}
    requirements:
        _locale: ^[a-z]{2}(?:_[A-Z]{2})?$

sylius_shop_default_locale:
    path: /
    methods: [GET]
    defaults:
        _controller: sylius.controller.shop.locale_switch:switchAction
```

With:

```yaml
# app/config/routing.yml

sylius_shop:
    resource: "@SyliusShopBundle/Resources/config/routing.yml"
```

2. Customise the security settings in the `app/config/security.yml`.

Replace:

```
# app/config/security.yml

parameters:
    # ...
    sylius.security.shop_regex: "^/(?!admin|api)[^/]++"
```

With:

```
# app/config/security.yml

parameters:
    # ...
    sylius.security.shop_regex: "^"
```

3. Customise SyliusShopBundle to use storage-based locale switching by adding the following lines at the end of the `app/config/config.yml`.

```
# app/config/config.yml

sylius_shop:
    locale_switcher: storage
```

How to render a menu of taxons (categories) in a view?

The way of rendering a menu of taxons is a supereasy reusable action, that you can adapt into any place you need.

How does it look like?

That's a menu that you will find on the default Sylius homepage:
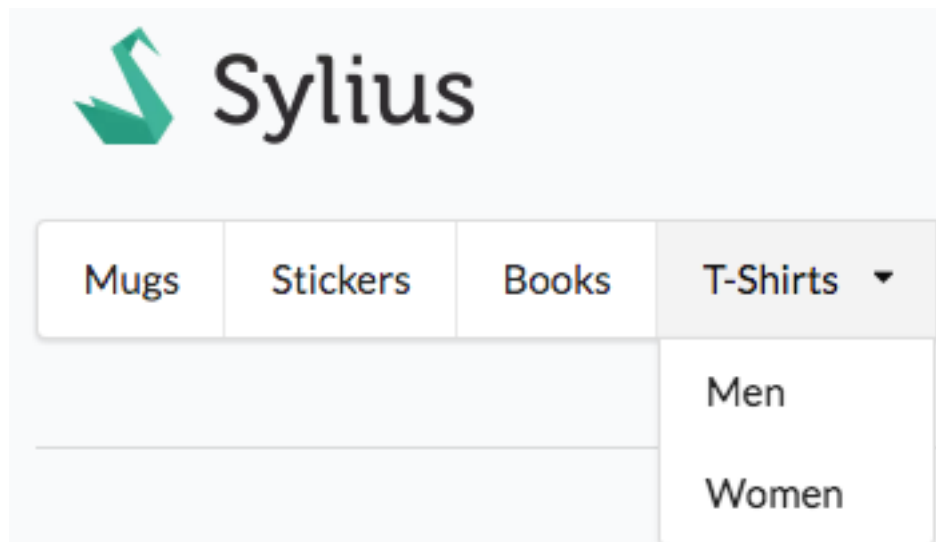
How to do it?

You can render such a menu wherever you have access to a `category` variable in the view, but also anywhere else.

The `findChildren` method of **TaxonRepository** takes a `parentCode` and nullable `locale`.

If `locale` parameter is not null the method returns also taxon's translation based on given `locale`.

To render a simple menu of categories in any twig template use:

```
{{ render(url('sylius_shop_partial_taxon_index_by_code', {'code': 'category',
↪'template': '@SyliusShop/Taxon/_horizontalMenu.html.twig'})) }}
```

You can of course customize the template or enclose the menu into html to make it look better.

That's all. Done!

Learn more

- *The Customization Guide*

How to embed a list of products into a view?

Let's imagine that you would like to render **a list of 5 latest products by a chosen taxon**. Such an action can take place on the category page. Here are the steps that you will need to take:

Create a new method for the product repository

To cover the usecase we have imagined we will need a new method on the product repository: `findLatestByChannelAndTaxonCode()`.

Tip: First learn how to customize repositories in *the customization docs here*.

The new repository method will take a channel object (retrieved from channel context), a taxon code and the count of items that you want to find.

Your extending repository class should look like that:

```php
<?php

namespace AppBundle\Repository;

use Sylius\Bundle\CoreBundle\Doctrine\ORM\ProductRepository as BaseProductRepository;
use Sylius\Component\Core\Model\ChannelInterface;

class ProductRepository extends BaseProductRepository
{
    /**
     * {@inheritdoc}
     */
    public function findLatestByChannelAndTaxonCode(ChannelInterface $channel, $code,
↪$count)
        {
```

(continues on next page)

```php
            return $this->createQueryBuilder('o')
                ->innerJoin('o.channels', 'channel')
                ->andWhere('o.enabled = true')
                ->andWhere('channel = :channel')
                ->innerJoin('o.productTaxons', 'productTaxons')
                ->addOrderBy('productTaxons.position', 'asc')
                ->innerJoin('productTaxons.taxon', 'taxon')
                ->andWhere('taxon.code = :code')
                ->setParameter('code', $code)
                ->setParameter('channel', $channel)
                ->setMaxResults($count)
                ->getQuery()
                ->getResult();
    }
}
```

And should be registered in the `app/config/config.yml` just like that:

```yaml
sylius_product:
    resources:
        product:
            classes:
                repository: AppBundle\Repository\ProductRepository
```

Configure routing for the action of products rendering

To be able to render a partial with the retrieved products configure routing for it in the `app/config/routing.yml`:

```yaml
# app/config/routing.yml
app_shop_partial_product_index_latest_by_taxon_code:
    path: /latest/{code}/{count} # configure a new path that has all the needed
↪variables
    methods: [GET]
    defaults:
        _controller: sylius.controller.product:indexAction # you make a call on the
↪Product Controller's index action
        _sylius:
            template: $template
            repository:
                method: findLatestByChannelAndTaxonCode # here use the new repository
↪method
                arguments:
                    - "expr:service('sylius.context.channel').getChannel()"
                    - $code
                    - $count
```

Render the result of your new path in a template

Having a new path, you can call it in a twig template that has acces to a taxon. Remember that you need to have your **taxon as a variable available there**. Render the list using a simple built-in template to try it out.

```
{{ render(url('app_shop_partial_product_index_latest_by_taxon_code', {'code': taxon.
↪code, 'count': 5, 'template': '@SyliusShop/Product/_horizontalList.html.twig'})) }}
```

Done. In the taxon view where you have rendered the new url you will see a simple list of 5 products from this taxon, ordered by position.

Learn more

- *The Customization Guide*

How to add Facebook login?

For integrating social login functionalities Sylius uses the HWIOAuthBundle. Here you will find the tutorial for integrating Facebook login into Sylius:

Set up the HWIOAuthBundle

- Add HWIOAuthBundle to your project:

```
$ composer require hwi/oauth-bundle
```

- Enable the bundle in the `AppKernel.php`:

```php
// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        // ...
        new HWI\Bundle\OAuthBundle\HWIOAuthBundle(),
    );
}
```

- Import the routing:

```yaml
# app/config/routing.yml
hwi_oauth_redirect:
    resource: "@HWIOAuthBundle/Resources/config/routing/redirect.xml"
    prefix:   /connect

hwi_oauth_connect:
    resource: "@HWIOAuthBundle/Resources/config/routing/connect.xml"
    prefix:   /connect

hwi_oauth_login:
    resource: "@HWIOAuthBundle/Resources/config/routing/login.xml"
    prefix:   /login

facebook:
    path: "/login/check-facebook"
```

Configure the connection to Facebook

Note: To properly connect to Facebook you will need a Facebook developer account. Having an account create a new app for your website. In your app dashboard you will have the client_id (App ID) and the client_secret (App Secret), which are needed for the configuration.

```yaml
# app/config/config.yml
hwi_oauth:
    firewall_names: [shop]
    resource_owners:
        facebook:
            type: facebook
            client_id: <client_id>
            client_secret: <client_secret>
            scope: "email"
```

Sylius uses email as the username, that's why we choose emails as scope for this connection.

Tip: If you cannot connect to your localhost with the Facebook app, configure its settings in such a way:

- **App Domain**: localhost

- Click +Add Platform and choose "Website" type.

- Provide the **Site URL** of the platform - your local server on which you run Sylius: http://localhost:8000

Configure the security layer

As Sylius already has a service that implements the **OAuthAwareUserProviderInterface** - sylius.oauth.user_provider - we can only configure the oauth firewall. Under the security: firewalls: shop: keys in the security.yml configure like below:

```yaml
# app/config/security.yml
security:
    firewalls:
        shop:
            oauth:
                resource_owners:
                    facebook: "/login/check-facebook"
                login_path: /login
                use_forward: false
                failure_path: /login

                oauth_user_provider:
                    service: sylius.oauth.user_provider
            anonymous: true
```

Add facebook login button

You can for instance override the login template (SyliusShopBundle/Resources/views/login.html.twig) in the app/Resources/SyliusShopBundle/views/login.html.twig and add these lines to be

able to login via Facebook.

```
<a href="{{ path('hwi_oauth_service_redirect', {'service': 'facebook' }) }}">
    <span>Login with Facebook</span>
</a>
```

Done!

Learn more

- [HWIOAuthBundle documentation](#)

How to manage content in Sylius?

Why do you need content management system?

Content management is one of the most important business aspects of modern eCommerce apps. Providing store updates like new blog pages, banners and promotion images is responsible for building the conversion rate either for new and existing clients.

Content management in Sylius

Sylius standard app does not come with a content management system. Our community has taken care of it. As Sylius does have a convenient dev oriented plugin environment, the developers from [BitBag](#) decided to develop their flexible CMS module. You can find it [here](#).

Tip: The whole plugin has its own [demo page](#) with specific use cases. You can access the [admin panel](#) with `login: sylius, password: sylius` credentials.

Inside the plugin, you will find:

- HTML, image and text blocks you can place in each Twig template
- Page resources
- Sections which you can use to create a blog, customer information, etc.
- FAQ module

A very handy feature of this plugin is that you can customize it for your specific needs like you do with each *Sylius model*.

Installation & usage

Find out more about how to install the plugin on [GitHub](#) in the README file.

Learn more

- *[How to create a plugin for Sylius?](#)*
- [BitBag plugins](#)

- FriendsOfSylius plugins

- *How to customize Sylius Checkout?*

- *How to disable guest checkout?*

- *How to add Facebook login?*

- *How to change a redirect after the add to cart action?*

- *How to render a menu of taxons (categories) in a view?*

- *How to embed a list of products into a view?*

- *How to disable localised URLs?*

- *How to manage content in Sylius?*

4.1.4 Payments

How to configure PayPal Express Checkout?

One of the most frequently used payment methods in e-commerce is PayPal. Its configuration in Sylius is really simple.

Add a payment method with the Paypal Expresss gateway in the Admin Panel

---

Note: To test this configuration properly you will need a developer account on Paypal.

- Create a new payment method choosing `Paypal Express Checkout` gateway from the gateways choice dropdown and enable it for chosen channels.

Go to the `http://localhost:8000/admin/payment-methods/new/paypal_express_checkout` url.

- Fill in the Paypal configuration form with your developer account data (`username`, `password` and `signature`).

- Save the new payment method.

Choosing Paypal Express method in Checkout

From now on Paypal Express will be available in Checkout in the channel you have created it for.

Done!

Learn more

- *Payments concept documentation*
- Payum - Project Documentation

How to configure Stripe Credit Card payment?

One of very important payment methods in e-commerce are credit cards. Payments via credit card are in Sylius supported by Stripe.

Install Stripe

Stripe is not available by default in Sylius, to have it you need to add its package via composer.

```
$ php composer require stripe/stripe-php:~4.1
```

Add a payment method with the Stripe gateway in the Admin Panel

---

Note: To test this configuration properly you will need a developer account on Stripe.

- Create a new payment method, choosing the `Stripe Credit Card` gateway from the gateways choice dropdown and enable it for chosen channels.

Go to the `http://localhost:8000/admin/payment-methods/new/stripe_checkout` url.

- Fill in the Stripe configuration form with your developer account data (`publishable_key` and `secret_key`).

• Save the new payment method.

Tip: If your are not sure how to do it check how we do it *for Paypal in this cookbook*.

Warning: When your project is behind a loadbalancer and uses https you probably need to configure trusted proxies. Otherwise the payment will not succeed and the user will endlessly loopback to the payment page without any notice.

Choosing Stripe Credit Card method in Checkout

From now on Stripe Credit Card will be available in Checkout in the channel you have added it to.

Done!

Learn more

• *Payments concept documentation*
• Payum - Project Documentation

How to encrypt gateway config stored in the database?

1. Add defuse/php-encryption to your project .. code-block:

```
composer require defuse/php-encryption
```

2. Generate your Defuse Secret Key by executing the following script:

```php
<?php

use Defuse\Crypto\Key;

require_once 'vendor/autoload.php';

var_dump(Key::createNewRandomKey()->saveToAsciiSafeString());
```

3. Store your generated key in a parameter in `app/config/parameters.yml`.

```yaml
# app/config/parameters.yml

parameters:
    # ...
    defuse_secret: "YOUR_GENERATED_KEY"
```

4. Add the following code to the application configuration in the `app/config/config.yml`.

```yaml
# app/config/config.yml

payum:
    dynamic_gateways:
```

```
        encryption:
            defuse_secret_key: "%defuse_secret%"
```

5. Existing gateway configs will be automatically encrypted when updated. New gateway configs will be encrypted by default.

- *How to configure PayPal Express Checkout?*
- *How to configure Stripe Credit Card payment?*
- *How to encrypt gateway config stored in the database?*

4.1.5 Emails

How to send a custom e-mail?

Note: This cookbook is suitable for a clean *sylius-standard installation*. For more general tips, while using *Sylius-MailerBundle* go to Sending configurable e-mails in Symfony Blogpost.

Currently **Sylius** is sending e-mails only in a few "must-have" cases - see *E-mails documentation*. Of course these cases may not be sufficient for your business needs. If so, you will need to create your own custom e-mails inside the system.

On a basic example we will now teach how to do it.

Let's assume that you would like such a feature in your system:

```
Feature: Sending a notification email to the administrator when a product is out of␣
→stock
    In order to be aware which products become out of stock
    As an Administrator
    I want to be notified via email when products become out of stock
```

To achieve that you will need to:

1. Create a new e-mail that will be sent:

- prepare a template for your email in the `app/Resources/views/Email`.

```twig
{# app/Resources/views/Email/out_of_stock.html.twig #}
{% block subject %}
    One of your products has become out of stock.
{% endblock %}

{% block body %}
    {% autoescape %}
        The {{ variant.name }} variant is out of stock!
    {% endautoescape %}
{% endblock %}
```

- configure the email under `sylius_mailer:` in the `app/config/emails.yml` included in `app/config/yml`.

```
# app/config/emails.yml
sylius_mailer:
    sender:
        name: Example.com
        address: no-reply@example.com
    emails:
        out_of_stock:
            subject: "A product has become out of stock!"
            template: "AppBundle:Email:out_of_stock.html.twig"
```

```
# app/config/config.yml
imports:
    - { resource: "emails.yml" }
```

2. Create an Email Manager class:

- It will need the **EmailSender**, the **AvailabilityChecker** and the **AdminUser Repository**.

- It will operate on the **Order** where it needs to check each OrderItem, get their **ProductVariants** and check if they are available.

```php
<?php

namespace AppBundle\EmailManager;

use Sylius\Component\Core\Model\OrderInterface;
use Sylius\Component\Inventory\Checker\AvailabilityCheckerInterface;
use Sylius\Component\Mailer\Sender\SenderInterface;
use Sylius\Component\Resource\Repository\RepositoryInterface;

class OutOfStockEmailManager
{
    /**
     * @var SenderInterface
     */
    private $emailSender;

    /**
     * @var AvailabilityCheckerInterface $availabilityChecker
     */
    private $availabilityChecker;

    /**
     * @var RepositoryInterface $adminUserRepository
     */
    private $adminUserRepository;

    /**
     * @param SenderInterface $emailSender
     * @param AvailabilityCheckerInterface $availabilityChecker
     * @param RepositoryInterface $adminUserRepository
     */
    public function __construct(
        SenderInterface $emailSender,
        AvailabilityCheckerInterface $availabilityChecker,
```

(continues on next page)

```php
        RepositoryInterface $adminUserRepository
    ) {
        $this->emailSender = $emailSender;
        $this->availabilityChecker = $availabilityChecker;
        $this->adminUserRepository = $adminUserRepository;
    }


    /**
     * @param OrderInterface $order
     */
    public function sendOutOfStockEmail(OrderInterface $order)
    {
        // get all admins, but remember to put them into an array
        $admins = $this->adminUserRepository->findAll()->toArray();

        foreach($order->getItems() as $item) {
            $variant = $item->getVariant();

            $stockIsSufficient = $this->availabilityChecker->isStockSufficient(
↪$variant, 1);

            if ($stockIsSufficient) {
                continue;
            }
            foreach($admins as $admin) {
                $this->emailSender->send('out_of_stock', [$admin->getEmail()], [
↪'variant' => $variant]);
            }
        }
    }
}
```

3. Register the manager as a service:

```yaml
# app/config/services.yml
services:
    app.email_manager.out_of_stock:
    class: AppBundle\EmailManager\OutOfStockEmailManager
    arguments: ['@sylius.email_sender', '@sylius.availability_checker', '@sylius.
↪repository.admin_user']
```

4. Customize the state machine callback of Order's Payment:

```yaml
# app/config/state_machine.yml
winzou_state_machine:
    sylius_order_payment:
        callbacks:
            after:
                app_out_of_stock_email:
                    on: ["pay"]
                    do: ["@app.email_manager.out_of_stock", "sendOutOfStockEmail"]
                    args: ["object"]
```

```
# app/config/config.yml
imports:
    - { resource: "state_machine.yml" }
```

Done!

Learn More

- *Emails Concept*
- *State Machine Concept*
- *Customization Guide - State Machine*
- Sending configurable e-mails in Symfony Blogpost

How to disable the order confirmation email?

In some usecases you may be wondering if it is possible to completely turn off the order confirmation email after the order complete.

This is a complicated situation, because we need to be precise what is our expected result:

- *to disable that email in the system completely*,
- *to send a different email on the complete action of an order instead of the order confirmation email*,

Below a few ways to disable that email are presented:

Disabling the email in the configuration

There is a pretty straightforward way to disable an e-mail using just a few lines of yaml:

```
# app/config/config.yml
sylius_mailer:
    emails:
        order_confirmation:
            enabled: false
```

That's all. With that configuration placed in your `app/config/config.yml` the order confirmation email will not be sent.

Disabling the listener responsible for that action

To easily turn off the sending of the order confirmation email you will need to disable the `OrderCompleteListener` service. This can be done via a CompilerPass.

```php
<?php

namespace AppBundle\DependencyInjection\Compiler;

use Symfony\Component\DependencyInjection\Compiler\CompilerPassInterface;
use Symfony\Component\DependencyInjection\ContainerBuilder;
```

```php
class MailPass implements CompilerPassInterface
{
    public function process(ContainerBuilder $container)
    {
        $container->removeDefinition('sylius.listener.order_complete');
    }
}
```

The above compiler pass needs to be added to your bundle in the `AppBundle/AppBundle.php` file:

```php
<?php

namespace AppBundle;

use AppBundle\DependencyInjection\Compiler\MailPass;
use Symfony\Component\HttpKernel\Bundle\Bundle;
use Symfony\Component\DependencyInjection\ContainerBuilder;

class AppBundle extends Bundle
{
    public function build(ContainerBuilder $container)
    {
        parent::build($container);

        $container->addCompilerPass(new MailPass());
    }
}
```

That's it, we have removed the definition of the listener that is responsible for sending the order confirmation email.

Learn more

- Compiler passes in the Symfony documentation

How to configure mailer?

There are many services used for sending transactional emails in web applications. You can find for instance Mailjet, Mandrill or SendGrid among them.

In Sylius emails are configured the Symfony way, so you can get inspired by the Symfony guides to those mailing services.

Basically to start sending emails via a mailing service you will need to:

1. Create an account on a mailing service. 2. In the *app/config/parameters.yml* **modify such parameters:**

```yaml
# By default it is set to "swiftmailer", we need to have "smtp" here
mailer_transport: smtp

# The mailer host may be called "SMTP Server" for some services. Copy its name from
↪your mailing provider and paste here.
mailer_host: ...

# These are Username and Password provided by the service for your account
mailer_user: ...
```

```
mailer_password: ~

# Here you provide a Mailing port suggested by your service. It can be 25, 465 or 587
# This parameter is not there by default, you need to add it.
mailer_port: 25
```

3. **Remember not to have the *disable_delivery: true* **parameter in the** *app/config/config_prod.yml* for your production environment.

Emails delivery is disable for *test*, *dev* and *stage* environments by default. The *prod* environment has delivery turned on by default, so there is nothing to worry about if you did not change anything about it.

That's pretty much all! All the other issues are dependent on the service you are using.

> **Warning:** Remember that the parameters like username or password must not be commited publicly to your repository. Save them as environment variables on your server.

Learn More

- *Emails Concept*
- Sending configurable e-mails in Symfony Blogpost
- *How to configure mailer?*
- *How to send a custom e-mail?*
- *How to disable the order confirmation email?*

4.1.6 Promotions

How to add a custom promotion rule?

Adding new, custom rules to your shop is a common usecase. You can imagine for instance, that you have some customers in your shop that you distinguish as premium. And for these premium customers you would like to give special promotions. For that you will need a new PromotionRule that will check if the customer is premium or not.

Create a new promotion rule

The new Rule needs a RuleChecker class:

```php
<?php

namespace AppBundle\Promotion\Checker\Rule;

use Sylius\Component\Promotion\Checker\Rule\RuleCheckerInterface;
use Sylius\Component\Promotion\Model\PromotionSubjectInterface;

class PremiumCustomerRuleChecker implements RuleCheckerInterface
{
    const TYPE = 'premium_customer';
```

```php
    /**
     * {@inheritdoc}
     */
    public function isEligible(PromotionSubjectInterface $subject, array
↪$configuration)
    {
        return $subject->getCustomer()->isPremium();
    }
}
```

Prepare a configuration form type for your new rule

To be able to configure a promotion with your new rule you will need a form type for the admin panel.

Create the configuration form type class:

```php
<?php

namespace AppBundle\Form\Type\Rule;

use Symfony\Component\Form\AbstractType;

class PremiumCustomerConfigurationType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix()
    {
        return 'app_promotion_rule_premium_customer_configuration';
    }
}
```

And configure it in the `app/config/services.yml`:

```yaml
# app/config/services.yml
app.form.type.promotion_rule.premium_customer_configuration:
    class: AppBundle\Form\Type\Rule\PremiumCustomerConfigurationType
    tags:
        - { name: form.type }
```

Register the new rule checker as a service in the `app/config/services.yml`:

```yaml
# apps/config/services.yml
app.promotion_rule_checker.premium_customer:
    class: AppBundle\Promotion\Checker\Rule\PremiumCustomerRuleChecker
    tags:
        - { name: sylius.promotion_rule_checker, type: premium_customer, form_type:
↪AppBundle\Form\Type\Rule\PremiumCustomerConfigurationType, label: Premium customer }
```

That's all. You will now be able to choose the new rule while creating a new promotion.

Tip: Depending on the type of rule that you would like to configure you may need to configure its form fields. See how we do it here for example.

Learn more

- *Customization Guide*
- *Promotions Concept Documentation*

How to add a custom promotion action?

Let's assume that you would like to have a promotion that gives **100% discount on the cheapest item in the cart**.

See what steps need to be taken to achieve that:

Create a new promotion action

You will need a new class `CheapestProductDiscountPromotionActionCommand`.

It will give a discount equal to the unit price of the cheapest item. That's why it needs to have the Proportional Distributor and the Adjustments Applicator. The `execute` method applies the discount and distributes it properly on the totals. This class needs also a `isConfigurationValid()` method which was omitted in the snippet below.

```php
<?php

namespace AppBundle\Promotion\Action;

use AppBundle\Promotion\Action\CheapestProductDiscountPromotionActionCommand;

class CheapestProductDiscountPromotionActionCommand extends
→DiscountPromotionActionCommand
{
    const TYPE = 'cheapest_item_discount';

    /**
     * @var ProportionalIntegerDistributorInterface
     */
    private $proportionalDistributor;

    /**
     * @var UnitsPromotionAdjustmentsApplicatorInterface
     */
    private $unitsPromotionAdjustmentsApplicator;

    /**
     * @param ProportionalIntegerDistributorInterface $proportionalIntegerDistributor
     * @param UnitsPromotionAdjustmentsApplicatorInterface
→$unitsPromotionAdjustmentsApplicator
     */
    public function __construct(
        ProportionalIntegerDistributorInterface $proportionalIntegerDistributor,
        UnitsPromotionAdjustmentsApplicatorInterface
→$unitsPromotionAdjustmentsApplicator
    ) {
        $this->proportionalDistributor = $proportionalIntegerDistributor;
        $this->unitsPromotionAdjustmentsApplicator =
→$unitsPromotionAdjustmentsApplicator;
    }
```

```php
    /**
     * {@inheritdoc}
     */
    public function execute(PromotionSubjectInterface $subject, array $configuration,
    ↪PromotionInterface $promotion)
    {
        if (!$subject instanceof OrderInterface) {
            throw new UnexpectedTypeException($subject, OrderInterface::class);
        }

        $items = $subject->getItems();

        $cheapestItem = $items->first();

        $itemsTotals = [];

        foreach ($items as $item) {
            $itemsTotals[] = $item->getTotal();

            $cheapestItem = ($item->getVariant()->getPrice() < $cheapestItem->
    ↪getVariant()->getPrice()) ? $item : $cheapestItem;
        }

        $splitPromotion = $this->proportionalDistributor->distribute($itemsTotals, -1
    ↪* $cheapestItem->getVariant()->getPrice());
        $this->unitsPromotionAdjustmentsApplicator->apply($subject, $promotion,
    ↪$splitPromotion);
    }

    /**
     * {@inheritdoc}
     */
    public function getConfigurationFormType()
    {
        return CheapestProductDiscountPromotionActionCommand::class;
    }
}
```

Prepare a configuration form type for the admin panel

The new action needs a form type to be available in the admin panel, while creating a new promotion.

```php
<?php

namespace AppBundle\Form\Type\Action;

use Symfony\Component\Form\AbstractType;

class CheapestProductDiscountConfigurationType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix()
    {
```

```
        return 'app_promotion_action_cheapest_product_discount_configuration';
    }
}
```

Register the action as a service

In the `app/config/services.yml` configure:

```yaml
# app/config/services.yml
app.promotion_action.cheapest_product_discount:
    class: AppBundle\Promotion\Action\CheapestProductDiscountPromotionActionCommand
    arguments: ['@sylius.proportional_integer_distributor', '@sylius.promotion.units_
↪promotion_adjustments_applicator']
    tags:
        - { name: sylius.promotion_action, type: cheapest_product_discount, form_
↪type: AppBundle\Form\Type\Action\CheapestProductDiscountConfigurationType, label:␣
↪Cheapest product discount }
```

Register the form type as a service

In the `app/config/services.yml` configure:

```yaml
# app/config/services.yml
app.form.type.promotion_action.cheapest_product_discount_configuration:
    class: AppBundle\Form\Type\Action\CheapestProductDiscountConfigurationType
    tags:
        - { name: form.type }
```

Create a new promotion with your action

Go to the admin panel of your system. On the `/admin/promotions/new` url you can create a new promotion.

In its configuration you can choose your new "Cheapest product discount" action.

That's all. **Done!**

Learn more

- *Customization Guide*
- *Promotions Concept Documentation*
- *How to add a custom promotion action?*
- *How to add a custom promotion rule?*

4.1.7 Images

How to resize images?

In Sylius we are using the LiipImagineBundle for handling images.

Tip: You will find a reference to the types of filters in the LiipImagineBundle in their documentation.

There are three places in the Sylius platform where the configuration for images can be found:

- AdminBundle

- ShopBundle

- CoreBundle

These configs provide you with a set of filters for resizing images to **thumbnails**.

`sylius_admin_product_tiny_thumbnail`	size: [64, 64]
`sylius_admin_product_thumbnail`	size: [50, 50]
`sylius_shop_product_tiny_thumbnail`	size: [64, 64]
`sylius_shop_product_small_thumbnail`	size: [150, 112]
`sylius_shop_product_thumbnail`	size: [260, 260]
`sylius_shop_product_large_thumbnail`	size: [550, 412]
`sylius_small`	size: [120, 90]
`sylius_medium`	size: [240, 180]
`sylius_large`	size: [640, 480]

How to resize images with filters?

Knowing that you have filters out of the box you need to also know how to use them with images in **Twig** templates.

The `imagine_filter('name')` is a twig filter. This is how you would get an image path for on object `item` with a thumbnail applied:

```
<img src="{{ object.path|imagine_filter('sylius_small') }}" />
```

Note: Sylius stores images on entities by saving a `path` to the file in the database. The imagine_filter root path is `/web/media/image`.

How to add custom image resizing filters?

If the filters we have in Sylius by deafult are not suitable for your needs, you can easily add your own.

All you need to do is to configure new filter in the `app/config/config.yml` file. For example you can create a filter for advertisement banners:

```
# app/config/config.yml
liip_imagine:
    filter_sets:
        advert_banner:
            filters:
                thumbnail: { size: [800, 200], mode: inset }
```

How to use your new filter in Twig?

```
<img src="{{ banner.path|imagine_filter('advert_banner') }}" />
```

- The LiipImagineBundle documentation

How to store images in MongoDB GridFS?

This guide will show you how to store product images in MongoDB GridFS using the DoctrineMongoDBBundle. We're assuming that you already enabled and configured the bundle accordingly.

Prerequisite: data structure

First of all a document class to store an image is required. To make GridFS files easily reusable, we introduce a mapped superclass which will store the basic file information.

```php
<?php

namespace AppBundle\Document;

use Doctrine\MongoDB\GridFSFile;

abstract class File
{
    /**
     * @var string
     */
    protected $id;

    /**
     * @var GridFSFile
     */
    protected $file;

    /**
     * @var int
     */
    protected $length;

    /**
     * @var int
     */
    protected $chunkSize;

    /**
     * @var \DateTime
     */
    protected $uploadDate;

    /**
     * @var string
     */
    protected $md5;

    /**
     * @var string
     */
```

(continued from previous page)

```php
    protected $filename;

    /**
     * @var string
     */
    protected $contentType;

    /**
     * @return string
     */
    public function getId()
    {
        return (string) $this->id;
    }

    /**
     * The file can either be a string if the document isn't persisted yet, or a
↪GridFSFile
     * if the document has already been persisted.
     *
     * @return GridFSFile|string
     */
    public function getFile()
    {
        return $this->file;
    }

    /**
     * @param string $file
     *
     * @return File
     */
    public function setFile($file)
    {
        $this->file = $file;

        if (!$this->contentType) {
            $this->contentType = mime_content_type($file);
        }

        return $this;
    }

    /**
     * @return int
     */
    public function getLength()
    {
        return (int) $this->length;
    }

    /**
     * @return int
     */
    public function getChunkSize()
    {
        return (int) $this->chunkSize;
```

(continues on next page)

```php
    }

    /**
     * @return \DateTime
     */
    public function getUploadDate()
    {
        return $this->uploadDate;
    }

    /**
     * @return string
     */
    public function getMd5()
    {
        return (string) $this->md5;
    }

    /**
     * @return string
     */
    public function getFilename()
    {
        return (string) $this->filename;
    }

    /**
     * @param string $filename
     *
     * @return File
     */
    public function setFilename($filename)
    {
        if ($filename === '') {
            $filename = null;
        }

        $this->filename = $filename;

        return $this;
    }

    /**
     * @return string
     */
    public function getContentType()
    {
        return (string) $this->contentType;
    }

    /**
     * @param string $contentType
     *
     * @return File
     */
    public function setContentType($contentType)
    {
```

```php
        if ($contentType === '') {
            $contentType = null;
        }

        $this->contentType = $contentType;

        return $this;
    }

    /**
     * @return string
     */
    public function __toString()
    {
        return $this->getFilename();
    }
}
```

```xml
<!-- @AppBundle/Resources/doctrine/model/File.odm.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<doctrine-mongo-mapping xmlns="http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://doctrine-project.org/schemas/odm/doctrine-
↪mongo-mapping
                http://doctrine-project.org/schemas/odm/doctrine-mongo-mapping.xsd">

    <mapped-superclass name="AppBundle\Document\File">
        <field fieldName="id" id="true" />
        <field fieldName="file" type="file" />
        <field fieldName="length" type="int" />
        <field fieldName="chunkSize" type="int" />
        <field fieldName="uploadDate" type="date" />
        <field fieldName="md5" type="string" />
        <field fieldName="filename" type="string" index="true" order="asc" />
        <field fieldName="contentType" type="string" />
    </mapped-superclass>
</doctrine-mongo-mapping>
```

After creating the base `File` class, we're able to create the concrete product image documents for the original and cached images. They will be stored in the same collection, but can be differentiated by it's `type` field (see `DiscriminatorField` and `DiscriminatorMap`).

```php
<?php

namespace AppBundle\Document\Product;

use AppBundle\Document;

class Image extends Document\File
{
}
```

```xml
<!-- @AppBundle/Resources/doctrine/model/ProductImage.odm.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<doctrine-mongo-mapping xmlns="http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping"
```

```xml
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://doctrine-project.org/schemas/odm/
↪doctrine-mongo-mapping
                http://doctrine-project.org/schemas/odm/doctrine-mongo-mapping.xsd
↪">

    <document name="AppBundle\Document\Product\Image" collection="product_image"␣
↪inheritance-type="SINGLE_COLLECTION">
        <discriminator-field name="type" />
        <discriminator-map>
            <discriminator-mapping value="image" class=
↪"AppBundle\Document\Product\Image" />
            <discriminator-mapping value="cache" class=
↪"AppBundle\Document\Product\Image\Cache" />
        </discriminator-map>
        <default-discriminator-value value="image" />
    </document>
</doctrine-mongo-mapping>
```

Note: The image cache file stores its `filter` property in an embedded metadata object.

```php
<?php

namespace AppBundle\Document\Product\Image;

use AppBundle\Document;

final class Cache extends Document\Product\Image
{
    /**
     * @var Cache\Metadata
     */
    private $metadata;

    /**
     * @param string $file
     * @param string $contentType
     * @param string $filename
     * @param string $filter
     */
    public function __construct($file, $contentType, $filename, $filter)
    {
        $this
            ->setContentType($contentType)
            ->setFile($file)
            ->setFilename($filename)
        ;

        $this->metadata = new Cache\Metadata($filter);
    }

    /**
     * @return Cache\Metadata
     */
```

```php
    public function getMetadata()
    {
        return $this->metadata;
    }
}
```

```xml
<!-- @AppBundle/Resources/doctrine/model/ProductImageCache.odm.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<doctrine-mongo-mapping xmlns="http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://doctrine-project.org/schemas/odm/
↪doctrine-mongo-mapping
                        http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping.xsd">

    <document name="AppBundle\Document\Product\Image\Cache">
        <embed-one field="metadata" target-document=
↪"AppBundle\Document\Product\Image\Cache\Metadata" />
    </document>
</doctrine-mongo-mapping>
```

```php
<?php

namespace AppBundle\Document\Product\Image\Cache;

final class Metadata
{
    /**
     * @var string
     */
    private $filter;

    /**
     * @param string $filter
     */
    public function __construct($filter)
    {
        $this->filter = $filter;
    }

    /**
     * @return string
     */
    public function getFilter()
    {
        return $this->filter;
    }
}
```

```xml
<!-- @AppBundle/Resources/doctrine/model/ProductImageCacheMetadata.odm.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<doctrine-mongo-mapping xmlns="http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```xml
                        xsi:schemaLocation="http://doctrine-project.org/schemas/odm/
↪doctrine-mongo-mapping
                        http://doctrine-project.org/schemas/odm/doctrine-mongo-
↪mapping.xsd">

    <embedded-document name="AppBundle\Document\Product\Image\Cache\Metadata">
        <field fieldName="filter" type="string" index="true" order="asc" />
    </embedded-document>
</doctrine-mongo-mapping>
```

How to store images in MongoDB GridFS?

In Sylius the KnpGaufretteBundle is used to store images. In order to store images in MongoDB GridFS, we have to create new GridFS loader services for Gaufrette:

First of all a new service is configured.

```xml
<!-- @AppBundle/Resources/config/services.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services
            http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="app.gaufrette_loader.doctrine_grid_fs" class=
↪"Doctrine\MongoDB\GridFS" public="false">
            <factory service="doctrine.odm.mongodb.document_manager" method=
↪"getDocumentCollection" />
            <argument>AppBundle\Document\Product\Image</argument>
        </service>

        <service id="app.gaufrette_loader.grid_fs" class="MongoGridFS">
            <factory service="app.gaufrette_loader.doctrine_grid_fs" method=
↪"getMongoCollection" />
        </service>

        <!-- ... -->
    </services>
</container>
```

Now we can override the Gaufrette configuration in `app/config/config.yml` to use the newly created loader service `app.gaufrette_loader.grid_fs`.

```yaml
knp_gaufrette:
    adapters:
        sylius_image:
            gridfs:
                mongogridfs_id: app.gaufrette_loader.grid_fs
```

Once this configuration is changed, newly uploaded images are already stored in MongoDB GridFS.

How to load images from MongoDB GridFS?

Loading images from MongoDB GridFS is a bit more complicated and requires some custom classes.

First of all we have to create a new `data_loader` for the `LiipImagineBundle`.

```php
<?php

namespace AppBundle\Imagine\Binary\Loader;

use Doctrine\ODM\MongoDB\DocumentManager;
use Liip\ImagineBundle\Binary\Loader\LoaderInterface ;
use Liip\ImagineBundle\Exception\Binary\Loader\NotLoadableException;

final class GridFSLoader implements LoaderInterface
{
    /**
     * @var DocumentManager
     */
    protected $dm;

    /**
     * @var string
     */
    protected $class;

    /**
     * @param DocumentManager $dm
     * @param string $class
     */
    public function __construct(DocumentManager $dm, string $class)
    {
        $this->dm = $dm;
        $this->class = $class;
    }

    /**
     * {@inheritdoc}
     */
    public function find($filename)
    {
        $image = $this->dm
            ->getRepository($this->class)
            ->findOneBy(['filename' => $filename]);

        if (!$image) {
            throw new NotLoadableException(sprintf('Source image was not found with␣
↪filename "%s"', $filename));
        }

        return $image->getFile()->getBytes();
    }
}
```

Now we can create the service definition for the data loader:

```xml
<!-- @AppBundle/Resources/config/services.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
```

<div align="right">(continues on next page)</div>

```xml
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
        http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="app.imagine_loader.grid_fs" class=
→"AppBundle\Imagine\Binary\Loader\GridFSLoader">
            <argument type="service" id="doctrine.odm.mongodb.document_manager" />
            <argument>AppBundle\Document\Product\Image</argument>
            <tag name="liip_imagine.binary.loader" loader="app.imagine_loader.grid_fs
→" />
        </service>

        <!-- ... -->
    </services>
</container>
```

The `LiipImagineBundle` still doesn't know that we're storing our images in GridFS, which is why we have to create a custom resolver class that can find an image for a given filename and store new cached filter types of an image.

Note: The route `product_cache_image` is defined via an annotation on the `ImagineController::imageAction` class method below this example.

```php
<?php

namespace AppBundle\Imagine\Cache\Resolver;

use AppBundle\Document\Product\Image\Cache;
use Doctrine\ODM\MongoDB\DocumentManager;
use Doctrine\ODM\MongoDB\DocumentRepository;
use Liip\ImagineBundle\Binary\BinaryInterface;
use Liip\ImagineBundle\Imagine\Cache\Resolver\ResolverInterface;
use Symfony\Component\Routing\RouterInterface;

final class GridFSResolver implements ResolverInterface
{
    /**
     * @var DocumentManager
     */
    private $documentManager;

    /**
     * @var string
     */
    private $class;

    /**
     * @var RouterInterface
     */
    private $router;

    /**
     * @param DocumentManager $documentManager
     * @param string $class
```

```php
     * @param RouterInterface $router
     */
    public function __construct(DocumentManager $documentManager, string $class,
→RouterInterface $router)
    {
        $this->documentManager = $documentManager;
        $this->class = $class;
        $this->router = $router;
    }

    /**
     * {@inheritdoc}
     */
    public function isStored($path, $filter)
    {
        return $this->findCacheFile($path, $filter) !== null;
    }

    /**
     * {@inheritdoc}
     */
    public function resolve($path, $filter)
    {
        $cache = $this->findCacheFile($path, $filter);

        return $this->router->generate('product_cache_image', ['id' => $cache->
→getId()], RouterInterface::ABSOLUTE_URL);
    }

    /**
     * {@inheritdoc}
     *
     * @throws GridFSException
     */
    public function store(BinaryInterface $binary, $path, $filter)
    {
        $file = tempnam(sys_get_temp_dir(), 'GridFSResolver');

        if (file_put_contents($file, $binary->getContent()) === false) {
            // We're using a custom exception to make it explicit catchable
            throw new GridFSException("Could not write cache file '{$file}' to disk");
        }

        try {
            $cache = new Cache($file, $binary->getMimeType(), $path, $filter);

            $this->documentManager->persist($cache);
            $this->documentManager->flush();
        } finally {
            @unlink($file);
        }
    }

    /**
     * {@inheritdoc}
     */
    public function remove(array $paths, array $filters)
```

```php
    {
        if (empty($paths) && empty($filters)) {
            return;
        }

        $queryBuilder = $this->getRepository()->createQueryBuilder();

        $queryBuilder
            ->remove()
            ->multiple()
            ->field('metadata.filter')
            ->in($filters)
        ;

        if (!empty($paths)) {
            $queryBuilder
                ->field('filename')
                ->in($paths)
            ;
        }

        $queryBuilder->getQuery()->execute();
    }

    /**
     * @param string $path
     * @param string $filter
     *
     * @return Cache|null
     */
    private function findCacheFile($path, string $filter)
    {
        return $this->getRepository()->findOneBy(['filename' => $path, 'metadata.
→filter' => $filter]);
    }

    /**
     * @return DocumentRepository
     */
    private function getRepository()
    {
        return $this->documentManager->getRepository($this->class);
    }
}
```

```php
<?php

namespace AppBundle\Imagine\Cache\Resolver;

class GridFSException extends \RuntimeException
{
}
```

Create the service definition for the resolver:

```xml
<!-- @AppBundle/Resources/config/services.xml -->
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
        http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="app.imagine_resolver.grid_fs" class=
→"AppBundle\Imagine\Cache\Resolver\GridFSResolver">
            <argument type="service" id="doctrine.odm.mongodb.document_manage" />
            <argument>AppBundle\Document\Product\Image\Cache</argument>
            <argument type="service" id="router" />
            <tag name="liip_imagine.cache.resolver" resolver="app.imagine_resolver.
→grid_fs" />
        </service>

        <!-- ... -->
    </services>
</container>
```

Last but not least we have to override the `liip_imagine` configuration in the `app/config/config.yml` file to use the new data loader and resolver.

```yaml
liip_imagine:
    data_loader: app.imagine_loader.grid_fs
    cache: app.imagine_resolver.grid_fs
```

Now we're going to add a new controller action which can resolve a cached product image and it's route.

Note: This implementation uses the Symfony Doctrine param converter.

```xml
<!-- app/config/routing.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<routes xmlns="http://symfony.com/schema/routing"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/routing
        http://symfony.com/schema/routing/routing-1.0.xsd">

    <route id="product_cache_image" path="/product/media/cache/{id}" />

    <!-- ... -->
</routes>
```

```php
<?php

namespace AppBundle\Controller;

use AppBundle\Document\Product\Image\Cache;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\ResponseHeaderBag;

class ImagineController extends Controller
```

```
{
    /**
     * @param Cache $cache
     * @param Request $request
     *
     * @return Response
     */
    public function imageAction(Cache $cache, Request $request)
    {
        $response = new Response();
        $response->setEtag($cache->getMd5());

        if ($response->isNotModified($request)) {
            return $response;
        }

        $disposition = $response->headers->
→makeDisposition(ResponseHeaderBag::DISPOSITION_INLINE, $cache->getId());
        $response->headers->set('Content-Disposition', $disposition);
        $response->headers->set('Content-Type', $cache->getContentType());

        $response->setContent($cache->getFile()->getBytes());

        return $response;
    }
}
```

Learn more

- The MongoDB GridFS documentation
- The Doctrine MongoDB ODM documentation
- The DoctrineMongoDBBundle documentation

How to add images to an entity?

Extending entities with an `images` field is quite a popular use case. In this cookbook we will present how to **add image to the Shipping Method entity**.

Instructions:

1. Extend the ShippingMethod class with the ImagesAwareInterface

In order to override the `ShippingMethod` that lives inside of the SyliusCoreBundle, you have to create your own ShippingMethod class that will extend it:

```php
<?php

declare(strict_types=1);

namespace AppBundle\Entity;
```

```php
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Sylius\Component\Core\Model\ImagesAwareInterface;
use Sylius\Component\Core\Model\ImageInterface;
use Sylius\Component\Core\Model\ShippingMethod as BaseShippingMethod;

class ShippingMethod extends BaseShippingMethod implements ImagesAwareInterface
{
    /**
     * @var Collection|ImageInterface[]
     */
    protected $images;

    public function __construct()
    {
        parent::__construct();

        $this->images = new ArrayCollection();
    }

    /**
     * {@inheritdoc}
     */
    public function getImages(): Collection
    {
        return $this->images;
    }

    /**
     * {@inheritdoc}
     */
    public function getImagesByType(string $type): Collection
    {
        return $this->images->filter(function (ImageInterface $image) use ($type) {
            return $type === $image->getType();
        });
    }

    /**
     * {@inheritdoc}
     */
    public function hasImages(): bool
    {
        return !$this->images->isEmpty();
    }

    /**
     * {@inheritdoc}
     */
    public function hasImage(ImageInterface $image): bool
    {
        return $this->images->contains($image);
    }

    /**
     * {@inheritdoc}
```

```
     */
    public function addImage(ImageInterface $image): void
    {
        $image->setOwner($this);
        $this->images->add($image);
    }

    /**
     * {@inheritdoc}
     */
    public function removeImage(ImageInterface $image): void
    {
        if ($this->hasImage($image)) {
            $image->setOwner(null);
            $this->images->removeElement($image);
        }
    }
}
```

Tip: Read more about customizing models in the docs *here*.

2. Register your extended ShippingMethod as a resource's model class

With such a configuration in the `config.yml` you will register your ShippingMethod class in order to override the default one:

```
# app/config/config.yml
sylius_shipping:
    resources:
        shipping_method:
            classes:
                model: AppBundle\Entity\ShippingMethod
```

3. Create the ShippingMethodImage class

In the `AppBundle\Entity` namespace place the `ShippingMethodImage` class which should look like this:

```php
<?php

declare(strict_types=1);

namespace AppBundle\Entity;

use Sylius\Component\Core\Model\Image;

class ShippingMethodImage extends Image
{
}
```

4. Add the mapping file for the ShippingMethodImage

Your new entity will be saved in the database, therefore it needs a mapping file, where you will set the `ShippingMethod` as the `owner` of the `ShippingMethodImage`.

```yaml
# AppBundle/Resources/config/doctrine/ShippingMethodImage.orm.yml
AppBundle\Entity\ShippingMethodImage:
    type: entity
    table: app_shipping_method_image
    manyToOne:
        owner:
            targetEntity: AppBundle\Entity\ShippingMethod
            inversedBy: images
            joinColumn:
                name: owner_id
                referencedColumnName: id
                nullable: false
                onDelete: CASCADE
```

5. Modify the ShippingMethod's mapping file

The newly added `images` field has to be added to the mapping, with a relation to the `ShippingMethodImage`:

```yaml
# AppBundle/Resources/config/doctrine/ShippingMethod.orm.yml
AppBundle\Entity\ShippingMethod:
    type: entity
    table: sylius_shipping_method
    oneToMany:
        images:
            targetEntity: AppBundle\Entity\ShippingMethodImage
            mappedBy: owner
            orphanRemoval: true
            cascade:
                - all
```

6. Register the ShippingMethodImage as a resource

The `ShippingMethodImage` class needs to be registered as a Sylius resource:

```yaml
# app/config/config.yml
sylius_resource:
    resources:
        app.shipping_method_image:
            classes:
                model: AppBundle\Entity\ShippingMethodImage
```

7. Create the ShippingMethodImageType class

This is how the class for `ShippingMethodImageType` should look like. Place it in the `AppBundle\Form\Type\` directory.

```php
<?php

declare(strict_types=1);

namespace AppBundle\Form\Type;

use Sylius\Bundle\CoreBundle\Form\Type\ImageType;

final class ShippingMethodImageType extends ImageType
{
    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix(): string
    {
        return 'app_shipping_method_image';
    }
}
```

8. Register the ShippingMethodImageType as a service

After creating the form type class, you need to register it as a `form.type` service like below:

```yaml
# services.yml
services:
    app.form.type.shipping_method_image:
        class: AppBundle\Form\Type\ShippingMethodImageType
        tags:
            - { name: form.type }
        arguments: ['%app.model.shipping_method_image.class%']
```

9. Add the ShippingMethodImageType to the resource form configuration

What is more the new form type needs to be configured as the resource form of the `ShippingMethodImage`:

```yaml
# app/config/config.yml
sylius_resource:
    resources:
        app.shipping_method_image:
            classes:
                form: AppBundle\Form\Type\ShippingMethodImageType
```

10. Extend the ShippingMethodType with the images field

Tip: Read more about *customizing forms via extensions in the dedicated guide*.

Create the form extension class for the `Sylius\Bundle\ShippingBundle\Form\Type\ShippingMethodType`:

It needs to have the images field as a CollectionType.

```php
<?php

declare(strict_types=1);

namespace AppBundle\Form\Extension;

use AppBundle\Form\Type\ShippingMethodImageType;
use Sylius\Bundle\ShippingBundle\Form\Type\ShippingMethodType;
use Symfony\Component\Form\AbstractTypeExtension;
use Symfony\Component\Form\Extension\Core\Type\CollectionType;
use Symfony\Component\Form\FormBuilderInterface;

final class ShippingMethodTypeExtension extends AbstractTypeExtension
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder->add('images', CollectionType::class, [
            'entry_type' => ShippingMethodImageType::class,
            'allow_add' => true,
            'allow_delete' => true,
            'by_reference' => false,
            'label' => 'sylius.form.shipping_method.images',
        ]);
    }

    /**
     * {@inheritdoc}
     */
    public function getExtendedType(): string
    {
        return ShippingMethodType::class;
    }
}
```

Tip: In case you need only a single image upload, this can be done in 2 very easy steps.

First, in the code for the form provided above set `allow_add` and `allow_delete` to `false`

Second, in the `__construct` method of the `ShippingMethod` entity you defined earlier add the following:

```php
public function __construct()
{
    parent::__construct();
    $this->images = new ArrayCollection();
    $this->addImage(new ShippingMethodImage());
}
```

Register the form extension as a service:

```yaml
# services.yml
services:
    app.form.extension.type.shipping_method:
        class: AppBundle\Form\Extension\ShippingMethodTypeExtension
```

(continues on next page)

```
        tags:
            - { name: form.type_extension, extended_type:
→Sylius\Bundle\ShippingBundle\Form\Type\ShippingMethodType }
```

11. Declare the ImagesUploadListener service

In order to handle the image upload you need to attach the `ImagesUploadListener` to the `ShippingMethod` entity events:

```
# services.yml
services:
    app.listener.images_upload:
        class: Sylius\Bundle\CoreBundle\EventListener\ImagesUploadListener
        parent: sylius.listener.images_upload
        autowire: true
        autoconfigure: false
        public: false
        tags:
            - { name: kernel.event_listener, event: sylius.shipping_method.pre_create,
→ method: uploadImages }
            - { name: kernel.event_listener, event: sylius.shipping_method.pre_update,
→ method: uploadImages }
```

12. Render the images field in the form view

In order to achieve that you will need to customize the form view from the `SyliusAdminBundle/views/ShippingMethod/_form.html.twig` file.

Copy and paste its contents into your own `app/Resources/SyliusAdminBundle/views/ShippingMethod/_form.html.twig` file, and render the `{{ form_row(form.images) }}` field.

```twig
{# app/Resources/SyliusAdminBundle/views/ShippingMethod/_form.html.twig #}

{% from '@SyliusAdmin/Macro/translationForm.html.twig' import translationForm %}

<div class="ui two column stackable grid">
    <div class="column">
        <div class="ui segment">
            {{ form_errors(form) }}
            <div class="three fields">
                {{ form_row(form.code) }}
                {{ form_row(form.zone) }}
                {{ form_row(form.position) }}
            </div>
            {{ form_row(form.enabled) }}
            <h4 class="ui dividing header">{{ 'sylius.ui.availability'|trans }}</h4>
            {{ form_row(form.channels) }}
            <h4 class="ui dividing header">{{ 'sylius.ui.category_requirements'|trans
→}}</h4>
            {{ form_row(form.category) }}
            {% for categoryRequirementChoiceForm in form.categoryRequirement %}
                {{ form_row(categoryRequirementChoiceForm) }}
            {% endfor %}
```

```twig
            <h4 class="ui dividing header">{{ 'sylius.ui.taxes'|trans }}</h4>
            {{ form_row(form.taxCategory) }}
            <h4 class="ui dividing header">{{ 'sylius.ui.shipping_charges'|trans }}</
↪h4>
            {{ form_row(form.calculator) }}
            {% for name, calculatorConfigurationPrototype in form.vars.prototypes %}
                <div id="{{ form.calculator.vars.id }}_{{ name }}" data-container=".
↪configuration"
                     data-prototype="{{ form_
↪widget(calculatorConfigurationPrototype)|e }}">
                </div>
            {% endfor %}

            {# Here you go! #}
            {{ form_row(form.images) }}

            <div class="ui segment configuration">
                {% if form.configuration is defined %}
                    {% for field in form.configuration %}
                        {{ form_row(field) }}
                    {% endfor %}
                {% endif %}
            </div>
        </div>
    </div>
    <div class="column">
        {{ translationForm(form.translations) }}
    </div>
</div>
```

Tip: Learn more about customizing templates *here*.

---

13. Validation

Your form so far is working fine, but don't forget about validation. The easiest way is using validation config files under the `AppBundle/Resources/config/validation` folder.

This could look like this e.g.:

```yaml
# AppBundle\Resources\config\validation\ShippingMethodImage.yml
AppBundle\Entity\ShippingMethodImage:
  properties:
    file:
      - Image:
          groups: [sylius]
          maxHeight: 1000
          maxSize: 10240000
          maxWidth: 1000
          mimeTypes:
            - "image/png"
            - "image/jpg"
            - "image/jpeg"
            - "image/gif"
```

```
        mimeTypesMessage: 'This file format is not allowed. Please use PNG, JPG or
→GIF files.'
        minHeight: 200
        minWidth: 200
```

This defines the validation constraints for each image entity. Now connecting the validation of the `ShippingMethod` to the validation of each single `Image Entity` is left:

```
# AppBundle\Resources\config\validation\ShippingMethod.yml
AppBundle\Entity\ShippingMethod:
  properties:
    ...
    images:
      - Valid: ~
```

Learn more

- *GridBundle documentation*
- *ResourceBundle documentation*
- *Customization Guide*
- *How to resize images?*
- *How to store images in MongoDB GridFS?*
- *How to add images to an entity?*

4.1.8 Deployment

How to deploy Sylius to Platform.sh?

Tip: Start with reading Platform.sh documentation. Also Symfony provides a guide on deploying projects to Platform.sh.

The process of deploying Sylius to Platform.sh is based on the guidelines prepared for Symfony projects in general. In this guide you will find sufficient instructions to have your application up and running on Platform.sh.

1. Prepare a Platform.sh project

If you do not have it yet, go to the Platform.sh store, choose development plan and go through checkout. Then, when you will have a project ready, give it a name and proceed to `Import an existing site`.

Tip: To investigate if Platform.sh suits your needs, you can use their **free trial**, which you can choose as a development plan.

---

2. Make the application ready to deploy

- Add the `.platform.app.yaml` file at the root of your project repository

This is how this file should look like for Sylius (tuned version of the default Platform.sh example):

```yaml
# .platform.app.yaml
name: app

type: "php:7.1"
build:
    flavor: composer

relationships:
    database: "mysql:mysql"
    redis: "redis:redis"

runtime:
    extensions:
        - msgpack
        - igbinary
        - memcached
        - redis

dependencies:
    nodejs:
        yarn: "*"
        gulp-cli: "*"

web:
    locations:
        '/':
            root: "web"
            passthru: "/app.php"
            allow: true
            expires: -1
            scripts: true
        '/assets/shop':
            expires: 2w
            passthru: true
            allow: false
            rules:
                # Only allow static files from the assets directories.
                '\.(css|js|jpe?g|png|gif|svgz?|ico|bmp|tiff?
→|wbmp|ico|jng|bmp|html|pdf|otf|woff2|woff|eot|ttf|jar|swf|ogx|avi|wmv|asf|asx|mng|flv|webm|mov|ogv
→g|mp4|3gpp|weba|ra|m4a|mp3|mp2|mpe?ga|midi?)$':
                    allow: true
        '/media/image':
            expires: 2w
            passthru: true
            allow: false
            rules:
                # Only allow static files from the assets directories.
                '\.(jpe?g|png|gif|svgz?)$':
                    allow: true
        '/media/cache/resolve':
            passthru: "/app.php"
            expires: -1
```

(continues on next page)

```yaml
            allow: true
            scripts: true
        '/media/cache':
            expires: 2w
            passthru: true
            allow: false
            rules:
                # Only allow static files from the assets directories.
                '\.(jpe?g|png|gif|svgz?)$':
                    allow: true

disk: 4096

mounts:
    "/var/cache": "shared:files/cache"
    "/var/logs": "shared:files/logs"
    "/web/uploads": "shared:files/uploads"
    "/web/media": "shared:files/media"

hooks:
    build: |
        rm web/app_dev.php
        rm web/app_test.php
        rm web/app_test_cached.php
        rm -rf var/cache/*
        php bin/console --env=prod --no-debug --ansi cache:clear --no-warmup
        php bin/console --env=prod --no-debug --ansi cache:warmup
        php bin/console --env=prod --no-debug --ansi assets:install
        # Next command is only needed if you are using themes
        php bin/console --env=prod --no-debug --ansi sylius:theme:assets:install
        yarn install
        GULP_ENV=prod yarn build
    deploy: |
        rm -rf var/cache/*
        php bin/console --env=prod doctrine:migrations:migrate --no-interaction
```

The above configuration includes tuned cache expiration headers for static files. The cache lifetimes can be adjusted for your site if desired.

- Add `.platform/routes.yaml` file:

```yaml
# .platform/routes.yaml
"http://{default}/":
    type: upstream
    upstream: "app:http"


"http://www.{default}/":
    type: redirect
    to: "http://{default}/"
```

- Add `.platform/services.yaml` file:

This file will load `mysql` and `redis` on your Platform.sh server.

```yaml
# .platform/services.yaml
mysql:
    type: mysql
```

```
    disk: 1024

redis:
    type: redis:3.0
```

- Configure the access to the database:

In the `app/config/parameters_platform.php` file, put the following code:

```php
// app/config/parameters_platform.php
<?php

$relationships = getenv("PLATFORM_RELATIONSHIPS");

if (!$relationships) {
    return;
}

$relationships = json_decode(base64_decode($relationships), true);

foreach ($relationships['database'] as $endpoint) {
    if (empty($endpoint['query']['is_master'])) {
        continue;
    }

    $container->setParameter('database_driver', 'pdo_' . $endpoint['scheme']);
    $container->setParameter('database_host', $endpoint['host']);
    $container->setParameter('database_port', $endpoint['port']);
    $container->setParameter('database_name', $endpoint['path']);
    $container->setParameter('database_user', $endpoint['username']);
    $container->setParameter('database_password', $endpoint['password']);
    $container->setParameter('database_path', '');
}
foreach ($relationships['redis'] as $endpoint) {
    $container->setParameter('redis_dsn', 'redis://'.$endpoint['host'].':'.$endpoint[
→'port']);
}

$container->setParameter('sylius.cache', array('type' => 'array'));

ini_set('session.save_path', '/tmp/sessions');

if (getenv('PLATFORM_PROJECT_ENTROPY')) {
    $container->setParameter('secret', getenv('PLATFORM_PROJECT_ENTROPY'));
}
```

Remember to have it imported in the config:

```yaml
# app/config/config.yml
imports:
    # - { resource: parameters.yml } <- Has to be placed before our new file
    - { resource: parameters_platform.php }
```

> **Warning:** It is important to place newly created file after importing regular parameters.yml file. Otherwise your database connection will not work. Also this will be the file where you should set your required parameters. Its value will be fetched from environmental variables.

The application secret is used in several places in Sylius and Symfony. Platform.sh allows you to deploy an environment for each branch you have, and therefore it makes sense to have a secret automatically generated by the Platform.sh system. The last 3 lines in the sample above will use the Platform.sh-provided random value as the application secret.

3. Add Platform.sh as a remote to your repository:

Use the below command to add your Platform.sh project as the `platform` remote:

```
$ git remote add platform [PROJECT-ID]@git.[CLUSTER].platform.sh:[PROJECT-ID].git
```

The `PROJECT-ID` is the unique identifier of your project, and `CLUSTER` can be `eu` or `us` - depending on where are you deploying your project.

4. Commit the Platform.sh specific files:

```
$ git add .platform.app.yaml
$ git add .platform/*
$ git add app/config/parameters_platform.php
$ git add app/config/config.yml
$ git commit -m "Platform.sh deploy configuration files."
```

5. Push your project to the platform remote:

```
$ git push platform master
```

The output of this command shows you on which URL your online store can be accessed.

6. Connect to the project via SSH and install Sylius

The SSH command can be found in your project data on Platform.sh. Alternatively use the Platform CLI tool.

When you get connected please run:

```
$ php bin/console sylius:install --env prod
```

> **Warning:** By default platform.sh creates only one instance of a database with the *main* name. Platform.sh works with the concept of an environment per branch if activated. The idea is to mimic production settings per each branch.

7. Dive deeper

Learn some more specific topics related to Sylius & Symfony on our *Advanced Platform.sh Cookbook*

Learn more

- Platform.sh documentation: Configuring Symfony projects for Platform.sh

---

- Symfony documentation: Deploying Symfony to Platform.sh

- *Installation Guide*

Advanced Platform.sh configurations

The basic set-up let's you easily set-up a Platform.sh project running your Sylius application. It should give you an environment suitable for testing Platform.sh in combination with Sylius.

In this guide additional tips will be given in order to benefit in a production environment.

Keep sessions between deployments

The default configuration saves PHP sessions into `/tmp/sessions`. Platform.sh functions in such way that each deployment spins up a new container instance and therefore the temporary folder holding sessions will be gone.

In order to save the PHP sessions on disk, the following steps need to be followed:

- In `platform.app.yml` add the following under the mount property:

```
mount:
    "/app/sessions": "shared:files/sessions"
```

- In the `app/config/parameters_platform.php` replace the session path:

```
ini_set('session.save_path', '/app/app/sessions');
```

Alternatively you can use a `php.ini`` file in the root of your project:

```
session.save_path = "/app/app/sessions"
```

Use Redis for Doctrine caching:

Want to use the metacache, query cache or result cache Symfony and Doctrine have to offer? It comes with a caveat. Platform.sh doesn't allow you to connect to all your services yet from inside the *build* hook. The following tutorial will guide you through this and make use of Redis. In the default example Redis is already activated.

- In your `app/config/parameters.yml.dist` add:

```
parameters:
    metacache_driver: []
    querycache_driver: []
    resultcache_driver: []
    redis_dsn: ~
    redis_host: ~
    redis_port: ~
```

- In the `app/config/parameters_platform.php` file, under the part where the database credentials are set, add:

```
foreach ($relationships['redis'] as $endpoint) {
    $container->setParameter('metacache_driver', 'redis');
    $container->setParameter('querycache_driver', 'redis');
    $container->setParameter('resultcache_driver', 'redis');
```

(continues on next page)

```
    $container->setParameter('redis_dsn', 'redis://'.$endpoint['host'].':'.$endpoint[
↪'port']);
    $container->setParameter('redis_host', $endpoint['host']);
    $container->setParameter('redis_port', $endpoint['port']);
}
```

Tip: Your Redis connection credentials are now available, which you can also use for the default Symfony cache.

- In your `app/config/config_prod.yml` file add:

```
doctrine:
    orm:
        metadata_cache_driver:
            type: "%metacache_driver%"
            database: 1
            host: "%redis_host%"
            port: "%redis_port%"
        query_cache_driver:
            type: "%querycache_driver%"
            database: 2
            host: "%redis_host%"
            port: "%redis_port%"
        result_cache_driver:
            type: "%resultcache_driver%"
            database: 3
            host: "%redis_host%"
            port: "%redis_port%"
```

- If you want to empty the cache on deployment, adjust the deploy hook in `.platform.app.yaml`:

```
hooks:
    deploy: |
        rm -rf var/cache/*
        php bin/console --env=prod doctrine:cache:clear-metadata
        php bin/console --env=prod doctrine:cache:clear-query
        php bin/console --env=prod doctrine:cache:clear-result
        php bin/console --env=prod doctrine:migrations:migrate --no-interaction
```

Add default Sylius cronjobs:

Add the example below to your `.platform.app.yaml` file. This runs these cronjobs every 6 hours.

```
crons:
    cleanup_cart:
        spec: '0 */6 * * *'
        cmd: '/usr/bin/flock -n /tmp/lock.app.cleanup_cart bin/console sylius:remove-
↪expired-carts --env=prod --verbose'
    cleanup_order:
        spec: '0 */6 * * *'
        cmd: '/usr/bin/flock -n /tmp/lock.app.cleanup_order bin/console sylius:cancel-
↪unpaid-orders --env=prod --verbose'
```

Additional tips:

- **Platform.sh can serve gzipped versions of your static assets. Make sure to save your assets in the same folder, but with** a .gz suffix. The gulp-gzip node package comes very helpful integrating saving of .gz versions of your assets.
- Platform.sh comes with a New Relic integration.
- Platform.sh comes with a Blackfire.io integration

How to deploy Sylius to Cloudways PHP Hosting?

Cloudways is a managed hosting platform for custom PHP apps and PHP frameworks such as Symfony, Laravel, Codeigniter, Yii, CakePHP and many more. You can launch the servers on any of the five providers including DigitalOcean, Vultr, AWS, GCE and KYUP containers.

The deployment process of Sylius on Cloudways is pretty much straightforward and easy.

Now to install Sylius you need to go through series of few steps:

1. Launch Server with Custom PHP App

You should signup at Cloudways to buy the PHP servers from the above mentioned providers. Simply go to the pricing page and choose your required plan. You then need to go through the verification process. Once it done login to platform and launch your first Custom PHP application. You can follow the Gif too.

Now let's start the process of installing Sylius on Cloudways.

2. Install the latest version of Sylius via SSH

Open the SSH terminal from the **Server Management tab**. You can also use PuTTY for this purpose. Find the SSH credentials under the **Master Credentials** heading and login to the SSH terminal:
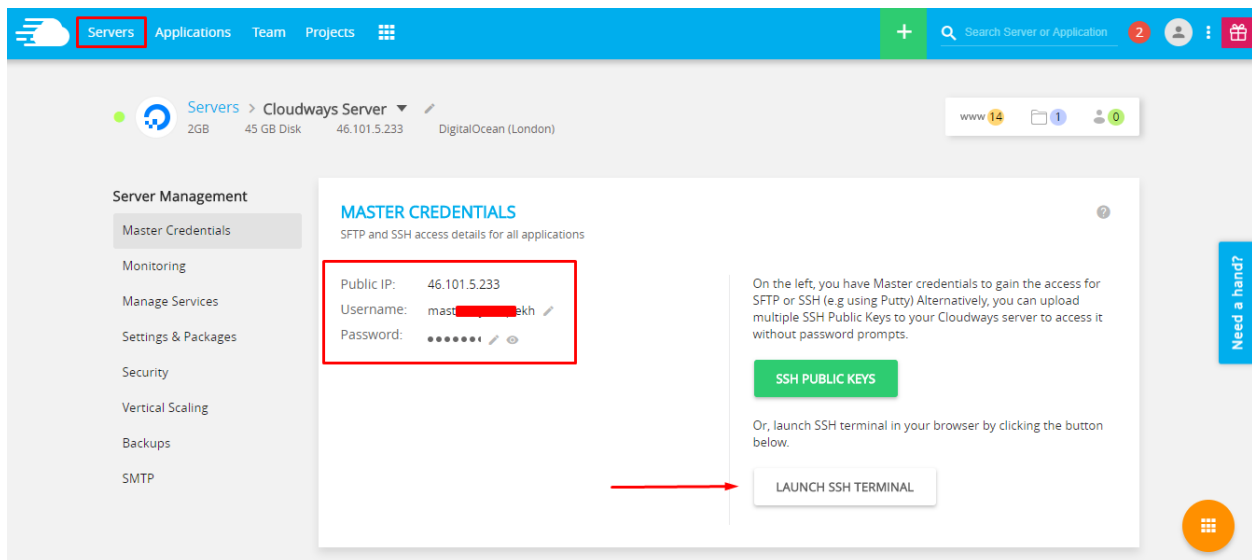
After the login, move to the application folder using the `cd` command and run the following command to start installing Sylius:

```
$ composer create-project sylius/sylius-standard myshop
```

The command will start installing the long list of dependencies for Sylius. Once the installation finishes, Sylius will ask for the database credentials. You can find the database username and password in the Application Access Details.
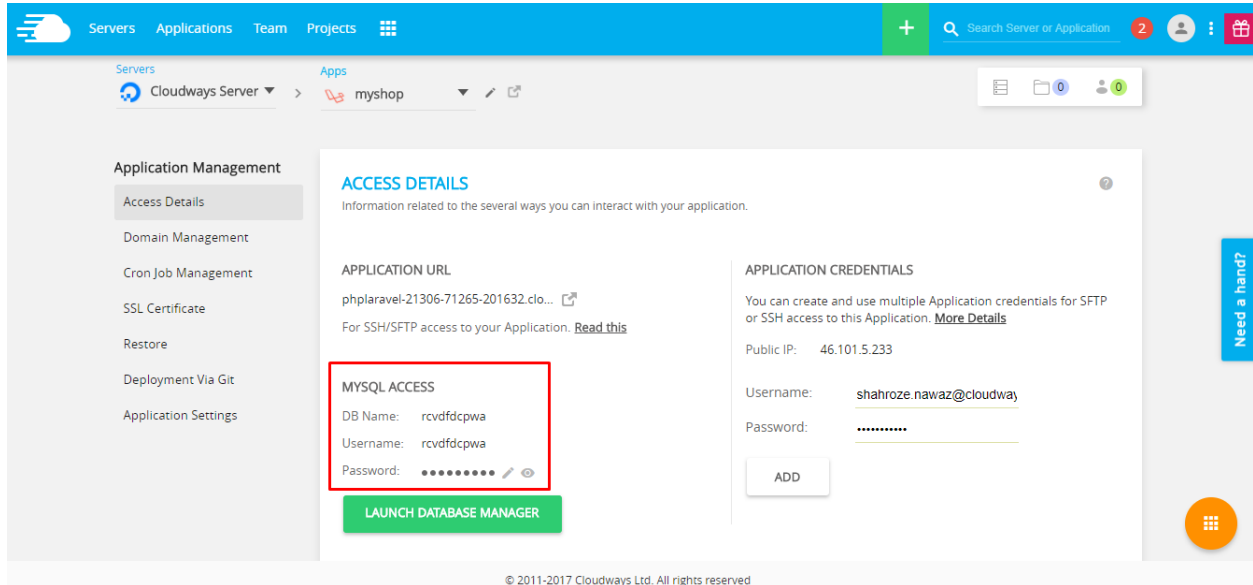
Enter the database details in the SSH terminal:
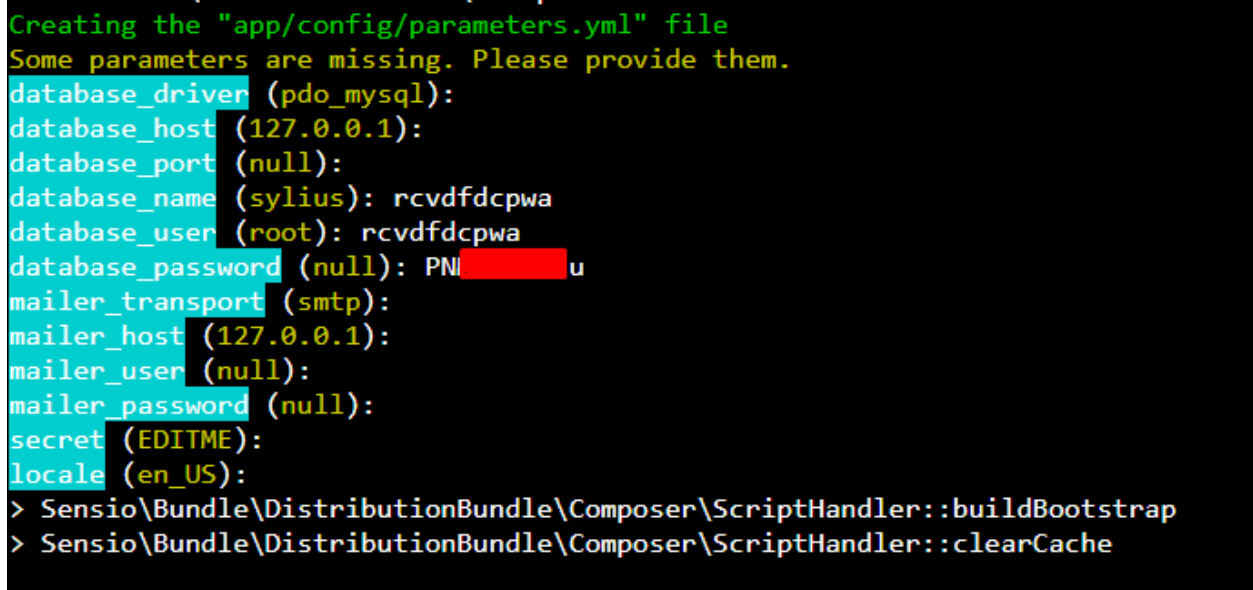
Keep the rest of the values to default so that the config file will have the defaults Sylius settings. If the need arises, you can obviously change these settings later.

3. Install Node Dependencies

Sylius requires several Node packages, which also needs to be installed and updated before setting up the shop. In addition, I also need to start and setup Gulp.

Now move to the myshop folder by using `cd myshop` and run the following command `yarn install`. Once the command finishes, run the next command, `yarn build`.

4. Install Sylius for the production environment

Now run the following command:

```
$ bin/console sylius:install -e prod
```

5. Update The Webroot of the Application

Finally, the last step is to update the webroot of the application in the Platform. Move to the **Application Settings** tab and update it.
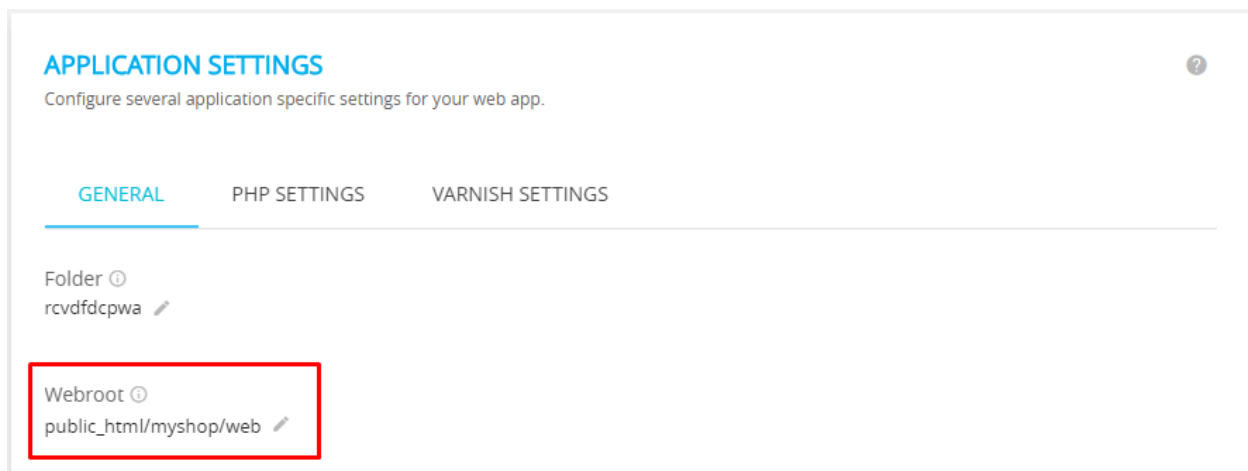
Now open the application URL as shown in the Access Details tab.

Learn more

- Cloudways PHP Hosting documentation: How to host PHP applications on DigitalOcean via Cloudways
- PHP FAQs And Features: Know more about PHP Hosting
- What You As A User Can Do With Cloudways PHP Stack

How to prepare simple CRON jobs?

What are CRON jobs?

This is what we call scheduling repetitive task on the server. In web applications this will be mainly repetitively running specific commands.

CRON jobs in Sylius

Sylius has two vital, predefined commands designed to be run as cron jobs on your server.

- `sylius:remove-expired-carts` - to remove carts that have expired after desired time
- `sylius:cancel-unpaid-orders` - to cancel orders that are still unpaid after desired time

How to configure a CRON job ?

Tip: Learn more here: Cron and Crontab usage and examples.

- *How to deploy Sylius to Platform.sh?*
- *Advanced Platform.sh configurations*
- *How to deploy Sylius to Cloudways PHP Hosting?*
- *How to prepare simple CRON jobs?*

4.1.9 Configuration

How to disable default shop, admin or API of Sylius?

When you are using Sylius as a whole you may be needing to remove some of its parts. It is possible to remove for example Sylius shop to have only administration panel and API. Or the other way, remove API if you do not need it.

Therefore you have this guide that will help you when wanting to disable shop, admin or API of Sylius.

How to disable Sylius shop?

1. Remove SyliusShopBundle from `app/AppKernel`.

```php
// # app/AppKernel.php

public function registerBundles()
{
    $bundles = [
        new \Sylius\Bundle\AdminBundle\SyliusAdminBundle(),
        // new \Sylius\Bundle\ShopBundle\SyliusShopBundle(), // - remove or leave
this line commented

        new \FOS\OAuthServerBundle\FOSOAuthServerBundle(),
        new \Sylius\Bundle\AdminApiBundle\SyliusAdminApiBundle(),

        new \AppBundle\AppBundle(),
    ];

    return array_merge(parent::registerBundles(), $bundles);
}
```

2. Remove SyliusShopBundle's config import from `app/config/config.yml`.

Here you've got the line that should disappear from imports:

```
imports:
#    - { resource: "@SyliusShopBundle/Resources/config/app/config.yml" } # remove or
↪leave this line commented
```

3. Remove SyliusShopBundle routing configuration from `app/config/routing.yml`.

```
# sylius_shop:
#     resource: "@SyliusShopBundle/Resources/config/routing.yml" # remove or leave
↪these lines commented
```

4. Remove security configuration from `app/config/security.yml`.

The part that has to be removed from this file is shown below:

```
parameters:
    # sylius.security.shop_regex: "^/(?!admin|api/.*|api$)[^/]++"

security:
    firewalls:
# Delete or leave this part commented
#        shop:
#            switch_user: { role: ROLE_ALLOWED_TO_SWITCH }
#            context: shop
#            pattern: "%sylius.security.shop_regex%"
#            form_login:
#                success_handler: sylius.authentication.success_handler
#                failure_handler: sylius.authentication.failure_handler
#                provider: sylius_shop_user_provider
#                login_path: sylius_shop_login
#                check_path: sylius_shop_login_check
#                failure_path: sylius_shop_login
#                default_target_path: sylius_shop_homepage
#                use_forward: false
#                use_referer: true
#                csrf_token_generator: security.csrf.token_manager
#                csrf_parameter: _csrf_shop_security_token
#                csrf_token_id: shop_authenticate
#            remember_me:
#                secret: "%secret%"
#                name: APP_SHOP_REMEMBER_ME
#                lifetime: 31536000
#                remember_me_parameter: _remember_me
#            logout:
#                path: sylius_shop_logout
#                target: sylius_shop_login
#                invalidate_session: false
#                success_handler: sylius.handler.shop_user_logout
#            anonymous: true

access_control:
#    - { path: "%sylius.security.shop_regex%/_partial", role: IS_AUTHENTICATED_
↪ANONYMOUSLY, ips: [127.0.0.1, ::1] }
#    - { path: "%sylius.security.shop_regex%/_partial", role: ROLE_NO_ACCESS }

#    - { path: "%sylius.security.shop_regex%/login", role: IS_AUTHENTICATED_
↪ANONYMOUSLY }

#    - { path: "%sylius.security.shop_regex%/register", role: IS_AUTHENTICATED_
↪ANONYMOUSLY }
```

```
#    - { path: "%sylius.security.shop_regex%/verify", role: IS_AUTHENTICATED_
↪ANONYMOUSLY }

#    - { path: "%sylius.security.shop_regex%/account", role: ROLE_USER }
#    - { path: "%sylius.security.shop_regex%/seller/register", role: ROLE_USER }
```

Done! There is no shop in Sylius now, just admin and API.

How to disable Sylius Admin?

1. Remove SyliusAdminBundle from `app/AppKernel`.

```php
// # app/AppKernel.php

public function registerBundles()
{
    $bundles = [
        // new \Sylius\Bundle\AdminBundle\SyliusAdminBundle(), // - remove or leave
↪this line commented
        new \Sylius\Bundle\ShopBundle\SyliusShopBundle(),

        new \FOS\OAuthServerBundle\FOSOAuthServerBundle(),
        new \Sylius\Bundle\AdminApiBundle\SyliusAdminApiBundle(),

        new \AppBundle\AppBundle(),
    ];

    return array_merge(parent::registerBundles(), $bundles);
}
```

2. Remove SyliusAdminBundle's config import from `app/config/config.yml`.

Here you've got the line that should disappear from imports:

```yaml
imports:
#    - { resource: "@SyliusAdminBundle/Resources/config/app/config.yml" } # remove or
↪leave this line commented
```

3. Remove SyliusAdminBundle routing configuration from `app/config/routing.yml`.

```yaml
#    sylius_admin:
#        resource: "@SyliusAdminBundle/Resources/config/routing.yml"
```

4. Remove security configuration from `app/config/security.yml`.

The part that has to be removed from this file is shown below:

```yaml
parameters:
# Delete or leave this part commented
#    sylius.security.admin_regex: "^/admin"
    sylius.security.shop_regex: "^/(?!api/.*|api$)[^/]++" # Remove `admin|` from the
↪pattern

security:
    firewalls:
# Delete or leave this part commented
```

```
#        admin:
#            switch_user: true
#            context: admin
#            pattern: "%sylius.security.admin_regex%"
#            form_login:
#                provider: sylius_admin_user_provider
#                login_path: sylius_admin_login
#                check_path: sylius_admin_login_check
#                failure_path: sylius_admin_login
#                default_target_path: sylius_admin_dashboard
#                use_forward: false
#                use_referer: true
#                csrf_token_generator: security.csrf.token_manager
#                csrf_parameter: _csrf_admin_security_token
#                csrf_token_id: admin_authenticate
#            remember_me:
#                secret: "%secret%"
#                path: /admin
#                name: APP_ADMIN_REMEMBER_ME
#                lifetime: 31536000
#                remember_me_parameter: _remember_me
#            logout:
#                path: sylius_admin_logout
#                target: sylius_admin_login
#            anonymous: true


access_control:
# Delete or leave this part commented
#    - { path: "%sylius.security.admin_regex%/_partial", role: IS_AUTHENTICATED_
↪ANONYMOUSLY, ips: [127.0.0.1, ::1] }
#    - { path: "%sylius.security.admin_regex%/_partial", role: ROLE_NO_ACCESS }

#    - { path: "%sylius.security.admin_regex%/login", role: IS_AUTHENTICATED_
↪ANONYMOUSLY }

#    - { path: "%sylius.security.admin_regex%", role: ROLE_ADMINISTRATION_ACCESS }
```

Done! There is no admin in Sylius now, just api and shop.

How to disable Sylius API?

1. Remove SyliusAdminApiBundle & FOSOAuthServerBundle from `app/AppKernel`.

```
// # app/AppKernel.php

public function registerBundles()
{
    $bundles = [
        new \Sylius\Bundle\AdminBundle\SyliusAdminBundle(),
        new \Sylius\Bundle\ShopBundle\SyliusShopBundle(),

        // new \FOS\OAuthServerBundle\FOSOAuthServerBundle(),
        // new \Sylius\Bundle\AdminApiBundle\SyliusAdminApiBundle(), // - remove or
↪leave this line commented
```

```
        new \AppBundle\AppBundle(),
    ];

    return array_merge(parent::registerBundles(), $bundles);
}
```

2. Remove SyliusAdminApiBundle's config import from `app/config/config.yml`.

Here you've got the line that should disappear from imports:

```
imports:
#    - { resource: "@SyliusAdminApiBundle/Resources/config/app/config.yml" } # remove␣
↪or leave this line commented
```

3. Remove SyliusAdminApiBundle routing configuration from `app/config/routing.yml`.

```
# sylius_api:
#    resource: "@SyliusAdminApiBundle/Resources/config/routing.yml" # remove or leave␣
↪these lines commented
```

4. Remove security configuration from `app/config/security.yml`.

The part that has to be removed from this file is shown below:

```
parameters:
# Delete or leave this part commented
#   sylius.security.api_regex: "^/api"
    sylius.security.shop_regex: "^/(?!admin$)[^/]++" # Remove `|api/.*|api` from the␣
↪pattern

security:
    firewalls:
# Delete or leave this part commented
#       oauth_token:
#           pattern: "%sylius.security.api_regex%/oauth/v2/token"
#           security: false
#       api:
#           pattern:    "%sylius.security.api_regex%/.*"
#           fos_oauth:  true
#           stateless:  true
#           anonymous:  true

access_control:
# Delete or leave this part commented
#    - { path: "%sylius.security.api_regex%/login", role: IS_AUTHENTICATED_
↪ANONYMOUSLY }

#    - { path: "%sylius.security.api_regex%/.*", role: ROLE_API_ACCESS }
```

5. Remove fos_rest config from `app/config/config.yml`.

```
fos_rest:
    format_listener:
        rules:
        #    - { path: '^/api', priorities: ['json', 'xml'], fallback_format: json,␣
↪prefer_extension: true } # remove or leave this line commented
```

Done! There is no API in Sylius now, just admin and shop.

Learn more

- *Architecture: Division into Core, Shop, Admin and API*

How to use installer commands?

Sylius platform ships with the `sylius:install` command, which takes care of creating the database, schema, dumping the assets and basic store configuration.

This command actually uses several other commands behind the scenes and each of those is available for you:

Checking system requirements

You can quickly check all your system requirements and possible recommendations by calling the following command:

```
$ php bin/console sylius:install:check-requirements
```

Database configuration

Sylius can create or even reset the database/schema for you, simply call:

```
$ php bin/console sylius:install:database
```

The command will check if your database schema exists. If yes, you may decide to recreate it from scratch, otherwise Sylius will take care of this automatically. It also allows you to load sample data.

Loading sample data

You can load sample data by calling the following command:

```
$ php bin/console sylius:install:sample-data
```

Basic store configuration

To configure your store, use this command and answer all questions:

```
$ php bin/console sylius:install:setup
```

Installing assets

You can reinstall all web assets by simply calling:

```
$ php bin/console sylius:install:assets
```

How to extend SyliusBundles and link those to an existing Sylius's Project Database?

In some cases and from another Symfony project (non Sylius/Sylius-Standard || Sylius/Sylius installation) you may be needing to be able to access/manipulate existing Sylius's data.

To be able to achieve this, you will need to install the desired Sylius's bundles, extend those, and link those to another Entity Manager which point to your existing Sylius's database.

For the sake of this guide, let's assume we want to access Sylius's users and addresses. In this way we are going to use SyliusCustomerBundle, SyliusUserBundle, SyliusAddressingBundle.

Installing the SyliusBundles

- Install SyliusCustomerBundle
- Install SyliusUserBundle
- Install SyliusAddressingBundle

Tip: Read more about how to install SyliusCustomerBundle *here*.

Tip: Read more about how to install SyliusUserBundle *here*.

Tip: Read more about how to install SyliusAddressingBundle *here*.

Extending the SyliusBundles

1. Generating our own bundles:

- Generate CustomerBundle
- Generate UserBundle
- Generate AddressingBundle

Tip: Read more about how to generate your own Symfony's bundle here.

2. Extending the Sylius's bundles:

```php
<?php

namespace CustomerBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class CustomerBundle extends Bundle
{
    public function getParent()
    {
        return 'SyliusCustomerBundle';
```

(continues on next page)

```
    }
}
```

```php
<?php

namespace UserBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class UserBundle extends Bundle
{
    public function getParent()
    {
        return 'SyliusUserBundle';
    }
}
```

```php
<?php

namespace AddressingBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class AddressingBundle extends Bundle
{
    public function getParent()
    {
        return 'SyliusAddressingBundle';
    }
}
```

3. Override the Sylius's bundles config and link our models to some_other_em:

```php
<?php

namespace CustomerBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

final class Configuration implements ConfigurationInterface
{
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('sylius_customer');

        return $treeBuilder;
    }
}
```

```php
<?php

namespace CustomerBundle\DependencyInjection;
```

```php
use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Loader;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;

class CustomerExtension extends Extension
{
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $loader = new Loader\YamlFileLoader($container, new FileLocator(__DIR__.'/../
→Resources/config'));
        $loader->load('services.yml');
    }
}
```

```yaml
# src/CustomerBundle/Resources/config/config.yml

sylius_customer:
    driver: doctrine/orm
    resources:
        customer:
            options:
                object_manager: some_other_em
            classes:
                model: CustomerBundle\Entity\Customer
                interface: Sylius\Component\Customer\Model\CustomerInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
        customer_group:
            options:
                object_manager: some_other_em
            classes:
                model: Sylius\Component\Customer\Model\CustomerGroup
                interface: Sylius\Component\Customer\Model\CustomerGroupInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
```

```php
<?php

namespace UserBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

final class Configuration implements ConfigurationInterface
{
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('sylius_user');

        return $treeBuilder;
    }
```

```
}
```

```php
<?php

namespace UserBundle\DependencyInjection;

use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Loader;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;

class UserExtension extends Extension
{
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $loader = new Loader\YamlFileLoader($container, new FileLocator(__DIR__.'/../
→Resources/config'));
        $loader->load('services.yml');
    }
}
```

```yaml
# src/UserBundle/Resources/config/config.yml

sylius_user:
    driver: doctrine/orm
    resources:
        shop:
            user:
                options:
                    object_manager: some_other_em
                classes:
                    model: UserBundle\Entity\ShopUser
                    repository: Sylius\Bundle\UserBundle\Doctrine\ORM\UserRepository
                    interface: Sylius\Component\User\Model\UserInterface
                    controller: Sylius\Bundle\UserBundle\Controller\UserController
                    factory: Sylius\Component\Resource\Factory\Factory
                templates: 'SyliusUserBundle:User'
                resetting:
                    token:
                        ttl: P1D
                        length: 16
                        field_name: passwordResetToken
                    pin:
                        length: 4
                        field_name: passwordResetToken
                verification:
                    token:
                        length: 16
                        field_name: emailVerificationToken
        oauth:
            user:
                options:
                    object_manager: some_other_em
```

```yaml
                    classes:
                        model: Sylius\Component\User\Model\UserOAuth
                        interface: Sylius\Component\User\Model\UserOAuthInterface
                        controller:␣
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                        factory: Sylius\Component\Resource\Factory\Factory
                        form: Sylius\Bundle\UserBundle\Form\Type\UserType
                    templates: 'SyliusUserBundle:User'
                    resetting:
                        token:
                            ttl: P1D
                            length: 16
                            field_name: passwordResetToken
                        pin:
                            length: 4
                            field_name: passwordResetToken
                    verification:
                        token:
                            length: 16
                            field_name: emailVerificationToken
```

```php
<?php

namespace AddressingBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

final class Configuration implements ConfigurationInterface
{
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('sylius_addressing');

        return $treeBuilder;
    }
}
```

```php
<?php

namespace AddressingBundle\DependencyInjection;

use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Loader;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;

class AddressingExtension extends Extension
{
    public function load(array $configs, ContainerBuilder $container)
    {
        $configuration = new Configuration();
        $config = $this->processConfiguration($configuration, $configs);

        $loader = new Loader\YamlFileLoader($container, new FileLocator(__DIR__.'/../
→Resources/config'));
```

```
        $loader->load('services.yml');
    }
}
```

```yaml
# src/AddressingBundle/Resources/config/config.yml

sylius_addressing:
    driver: doctrine/orm
    resources:
        address:
            options:
                object_manager: some_other_em
            classes:
                model: AddressingBundle\Entity\Address
                interface: Sylius\Component\Addressing\Model\AddressInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\AddressType
        country:
            options:
                object_manager: some_other_em
            classes:
                model: Sylius\Component\Addressing\Model\Country
                interface: Sylius\Component\Addressing\Model\CountryInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\CountryType
        province:
            options:
                object_manager: some_other_em
            classes:
                model: Sylius\Component\Addressing\Model\Province
                interface: Sylius\Component\Addressing\Model\ProvinceInterface
                controller:
→Sylius\Bundle\AddressingBundle\Controller\ProvinceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\ProvinceType
        zone:
            options:
                object_manager: some_other_em
            classes:
                model: Sylius\Component\Addressing\Model\Zone
                interface: Sylius\Component\Addressing\Model\ZoneInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\ZoneType
        zone_member:
            options:
                object_manager: some_other_em
            classes:
                model: Sylius\Component\Addressing\Model\ZoneMember
                interface: Sylius\Component\Addressing\Model\ZoneMemberInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\ZoneMemberType
```

4. Import our new config files to the global config

```
# app/config/config.yml

imports:
    - { resource: "@CustomerBundle/Resources/config/config.yml" }
    - { resource: "@UserBundle/Resources/config/config.yml" }
    - { resource: "@AddressingBundle/Resources/config/config.yml" }
```

5. Add the proper ORM mapping in the global config

```
# app/config/config.yml

# Doctrine Configuration
doctrine:
    orm:
        auto_generate_proxy_classes: '%kernel.debug%'
        default_entity_manager: default
        resolve_target_entities:
            Sylius\Component\User\Model\CustomerInterface:
→CustomerBundle\Entity\Customer
            Sylius\Component\User\Model\UserInterface: UserBundle\Entity\ShopUser
            Sylius\Component\Addressing\Model\AddressInterface:
→AddressingBundle\Entity\Address
        entity_managers:
            default:
                ...

            some_other_em:
                naming_strategy: doctrine.orm.naming_strategy.underscore
                connection: some_other_connexion
                auto_mapping: false
                mappings:
                    SyliusCustomerBundle:
                        type: xml
                        dir: "%kernel.project_dir%/vendor/sylius/customer-bundle/
→Resources/config/doctrine/model"
                        prefix: Sylius\Component\Customer\Model
                        is_bundle: false
                    CustomerBundle: ~
                    SyliusUserBundle:
                        type: xml
                        dir: "%kernel.project_dir%/vendor/sylius/user-bundle/
→Resources/config/doctrine/model"
                        prefix: Sylius\Component\User\Model
                        is_bundle: false
                    UserBundle: ~
                    SyliusAddressingBundle:
                        type: xml
                        dir: "%kernel.project_dir%/vendor/sylius/addressing-bundle/
→Resources/config/doctrine/model"
                        prefix: Sylius\Component\Addressing\Model
                        is_bundle: false
                    AddressingBundle: ~
```

6. Override the Sylius's models and add the missing relations:

As the Sylius's models which hold the declaration and the mapping of the relations between, in our case, SyliusCustomer, SyliusUser and SyliusAddressing are provided by the SyliusCoreBundle and as we don't have access to it we need to redefine the relations and their related mapping on our bundles.

```php
<?php

namespace CustomerBundle\Entity;

use Sylius\Component\Customer\Model\Customer as BaseCustomer;
use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use AddressingBundle\Entity\Address;
use UserBundle\Entity\ShopUser;

class Customer extends BaseCustomer
{
    private $defaultAddress;
    private $user;
    private $addresses;

    public function __construct()
    {
        parent::__construct();

        $this->addresses = new ArrayCollection();
    }

    /**
     * Set defaultAddress
     *
     * @param Address $defaultAddress
     *
     * @return Customer
     */
    public function setDefaultAddress(Address $defaultAddress = null)
    {
        $this->defaultAddress = $defaultAddress;

        if (null !== $defaultAddress)
        {
            $this->addAddress($defaultAddress);
        }

        return $this;
    }

    /**
     * Get defaultAddress
     *
     * @return Address
     */
    public function getDefaultAddress()
    {
        return $this->defaultAddress;
    }

    /**
     * Set user
     *
     * @param ShopUser $user
     *
```

(continues on next page)

```php
     * @return Customer
     */
    public function setUser(ShopUser $user = null)
    {
        $this->user = $user;

        return $this;
    }

    /**
     * Get user
     *
     * @return ShopUser
     */
    public function getUser()
    {
        return $this->user;
    }

    /**
     * Add address
     *
     * @param Address $address
     *
     * @return Customer
     */
    public function addAddress(Address $address)
    {
        if (!$this->hasAddress($address))
        {
            $this->addresses[] = $address;
            $address->setCustomer($this);
        }

        return $this;
    }

    /**
     * Remove address
     *
     * @param Address $address
     */
    public function removeAddress(Address $address)
    {
        $this->addresses->removeElement($address);
        $address->setCustomer(null);
    }

    /**
     * Get addresses
     *
     * @return \Doctrine\Common\Collections\Collection
     */
    public function getAddresses()
    {
        return $this->addresses;
    }
```

```php
    public function hasAddress(Address $address)
    {
        return $this->addresses->contains($address);
    }
}
```

```php
<?php

namespace UserBundle\Entity;

use Sylius\Component\User\Model\User as BaseUser;

class ShopUser extends BaseUser
{
    private $customer;

    /**
     * Get customer
     * @return
     */
    public function getCustomer()
    {
        return $this->customer;
    }

    /**
     * Set customer
     * @return $this
     */
    public function setCustomer($customer)
    {
        $this->customer = $customer;
        return $this;
    }
}
```

```php
<?php

namespace AddressingBundle\Entity;

use Sylius\Component\Addressing\Model\Address as BaseAddress;
use CustomerBundle\Entity\Customer;

class Address extends BaseAddress
{
    private $customer;

    /**
     * Set customer
     *
     * @param Customer $customer
     *
     * @return Address
     */
    public function setCustomer(Customer $customer = null)
```

```
    {
        $this->customer = $customer;

        return $this;
    }

    /**
     * Get customer
     *
     * @return Customer
     */
    public function getCustomer()
    {
        return $this->customer;
    }
}
```

7. Add the proper ORM mapping to our models:

```yaml
# src/CustomerBundle/Resources/config/doctrine/Customer.orm.yml

CustomerBundle\Entity\Customer:
    type: entity
    table: sylius_customer
    oneToOne:
        defaultAddress:
            targetEntity: AddressingBundle\Entity\Address
            joinColumn:
                name: default_address_id
                onDelete: SET NULL
            cascade: ["persist"]
        user:
            targetEntity: UserBundle\Entity\ShopUser
            mappedBy: customer
            cascade: ["persist"]
    oneToMany:
        addresses:
            targetEntity: AddressingBundle\Entity\Address
            mappedBy: customer
            cascade: ["all"]
```

```yaml
# src/UserBundle/Resources/config/doctrine/ShopUser.orm.yml

UserBundle\Entity\ShopUser:
    type: entity
    table: sylius_shop_user
    oneToOne:
        customer:
            targetEntity: CustomerBundle\Entity\Customer
            inversedBy: user
            joinColumn:
                name: customer_id
                referencedColumnName: id
                nullable: false
            cascade: ["persist"]
```

```
# src/AddressingBundle/Resources/config/doctrine/Address.orm.yml

AddressingBundle\Entity\Address:
    type: entity
    table: sylius_address
    manyToOne:
        customer:
            targetEntity: CustomerBundle\Entity\Customer
            inversedBy: addresses
            joinColumn:
                name: customer_id
                referencedColumnName: id
                nullable: true
                onDelete: CASCADE
```

8. Final steps:

- Clear both caches

At this point you should be able to test the ORM mapping of our "some_other_em" entity manager by calling:

```
$ php bin/console doctrine:schema:update --dump-sql --em=some_other_em
```

It should returns(as we did not add any new property to our models):

```
Nothing to update - your database is already in sync with the current entity metadata.
```

9. An "issue":

If you try in another hand to call a schema update on the default EM:

```
$ php bin/console doctrine:schema:update --dump-sql
```

It should returns:

```
[Doctrine\Common\Persistence\Mapping\MappingException]
The class 'UserBundle\Entity\ShopUser' was not found in the chain configured␣
↪namespaces AppBundle\Entity, Sylius\Component\Customer\Model,␣
↪Sylius\Component\User\Model, Sylius\Component\Ad
dressing\Model
```

This seems to be a "known issue" related to the shema-tool CLI command, as obviously this command uses all the metadata collected across all mapping drivers.

To fix this I overriden the UpdateSchemaDoctrineCommand and excluded all the Sylius metadatas when the default entity manager is specified.

```php
<?php

namespace AppBundle\Command;

use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Doctrine\ORM\Tools\SchemaTool;
use Doctrine\Bundle\DoctrineBundle\Command\Proxy\UpdateSchemaDoctrineCommand;
```

(continues on next page)

```php
class DoctrineUpdateCommand extends UpdateSchemaDoctrineCommand
{

    protected function executeSchemaCommand(InputInterface $input, OutputInterface
↪$output, SchemaTool $schemaTool, array $metadatas)
    {
        $newMetadatas = array();
        foreach ($metadatas as $metadata)
        {
            if (empty($input->getOption('em')) || $input->getOption('em') == 'default
↪')
            {
                if (explode('\\', $metadata->getName())[0] != 'Sylius')
                {
                    array_push($newMetadatas, $metadata);
                }
            }
            else
            {
                array_push($newMetadatas, $metadata);
            }
        }

        parent::executeSchemaCommand($input, $output, $schemaTool, $newMetadatas);
    }

}
```

How to disable admin version notifications?

By default Sylius sends checks from the admin whether you are running the latest version. In case you are not running the latest version, a notification will be shown in the admin panel (top right).

This guide will instruct you how to disable this check & notification.

How to disable notifications?

Add the following configuration to `app/config/config.yml`.

```yaml
sylius_admin:
    notifications:
        enabled: false
```

- *How to use installer commands?*
- *How to disable default shop, admin or API of Sylius?*
- *How to extend SyliusBundles and link those to an existing Sylius's Project Database?*
- *How to disable admin version notifications?*

4.1.10 Frontend

How to customize Admin JS & CSS?

It is sometimes required to add your own JSS and CSS files for Sylius Admin. Achieving that is really straightforward.
We will now teach you how to do it!

How to add custom JS to Admin?

1. Prepare your own JS file:

As an example we will use a popup window script, it is easy for manual testing.

```
// web/assets/admin/js/custom.js
window.confirm("Your custom JS was loaded correctly!");
```

2. Prepare a file with your JS include, you can use the include template from SyliusUiBundle:

```
{# src/AppBundle/Resources/views/Admin/_javascripts.html.twig #}
{% include 'SyliusUiBundle::_javascripts.html.twig' with {'path': 'assets/admin/js/
↪custom.js'} %}
```

3. Use the Sonata block event to insert your javascripts:

Tip: Learn more about customizing templates via events in the customization guide *here*.

```
# src/AppBundle/Resources/config/services.yml
services:
    app.block_event_listener.admin.layout.javascripts:
        class: Sylius\Bundle\UiBundle\Block\BlockEventListener
        arguments:
            - '@@App/Admin/_javascripts.html.twig'
        tags:
            - { name: kernel.event_listener, event: sonata.block.event.sylius.admin.
↪layout.javascripts, method: onBlockEvent }
```

4. Additionally, to make sure everything is loaded run gulp:

```
$ yarn build
```

5. Go to Sylius Admin and check the results!

How to add custom CSS to Admin?

1. Prepare your own CSS file:

As an example we will change the sidebar menu background color, what is clearly visible at first sight.

```
// web/assets/admin/css/custom.css
#sidebar {
    background-color: #1abb9c;
}
```

2. Prepare a file with your CSS include, you can use the include template from SyliusUiBundle:

```
{# src/AppBundle/Resources/views/Admin/_stylesheets.html.twig #}
{% include 'SyliusUiBundle::_stylesheets.html.twig' with {'path': 'assets/admin/css/
↪custom.css'} %}
```

3. Use the Sonata block event to insert your stylesheets:

Tip: Learn more about customizing templates via events in the customization guide *here*.

```
# src/AppBundle/Resources/config/services.yml
services:
    app.block_event_listener.admin.layout.stylesheets:
        class: Sylius\Bundle\UiBundle\Block\BlockEventListener
        arguments:
            - '@@App/Admin/_stylesheets.html.twig'
        tags:
            - { name: kernel.event_listener, event: sonata.block.event.sylius.admin.
↪layout.stylesheets, method: onBlockEvent }
```

4. Additionally, to make sure everything is loaded run gulp:

```
$ yarn build
```

5. Go to Sylius Admin and check the results!

Learn more

- *Templates customizing*
- *How to customize Admin JS & CSS?*

The REST API Reference

The API guide covers the REST API of Sylius platform.

5.1 The REST API Reference

5.1.1 Introduction to Sylius REST API

This part of the documentation is about RESTful JSON/XML API for the Sylius platform.

Note: This documentation assumes you have at least some experience with REST APIs.

Tip: We strongly recommend starting with our basic guide to Sylius API in the Cookbook: "*How to use Sylius API?*".

5.1.2 Authorization

This part of documentation is about authorization to Sylius platform through API. In order to check this configuration, please set up your local copy of Sylius platform and change *sylius.test* to your address.

OAuth2

Sylius has the OAuth2 authorization configured. The authorization process is a standard procedure. Authorize as admin and enjoy the API!

Note: User has to have the ROLE_API_ACCESS role in order to access `/api` resources

---

Create OAuth client

Use Sylius command:

```
php bin/console sylius:oauth-server:create-client \
    --grant-type="password" \
    --grant-type="refresh_token" \
    --grant-type="token"
```

You will receive client public id and client secret

Exemplary Result

```
A new client with public id 3e2iqilq2ygwk0ccgogkcwco8oosckkkk4gkoc0k4s8s044wss,
↪secret 44ectenmudus8g88w4wkws84044ckw0k4w4kg0sokoss84oko8 has been added
```

Tip: If you use Guzzle check out OAuth2 plugin and use Password Credentials.

Obtain access token

Send the request with the following parameters:

Definition

```
GET /api/oauth/v2/token
```

Parameter	Parameter type	Description
client_id	query	Client public id generated in the previous step
client_secret	query	Client secret generated in the previous step
grant_type	query	We will use 'password' to authorize as user. Other available options are token and refresh-token
username	query	User name
password	query	User password

Note: This action can be done by POST method as well.

Example

```
curl http://sylius.test/api/oauth/v2/token \
    -d "client_id"=demo_client \
    -d "client_secret"=secret_demo_client \
    -d "grant_type"=password \
    -d "username"=api@example.com \
    -d "password"=sylius-api
```

Tip: In a developer environment there is a default API user and client data. To use this credentials you have to load data fixtures. Otherwise you have to use your user data and replace client id and client secret with data generated in a previous step.

Exemplary Response

```
{
    "access_token":
→"NzFiYTM4ZTEwMjcwZTcyZWIzZTA0NmY3NjE3MTIyMjM1Y2NlMmNlNWEyMTAzY2UzYmY0YWIxYmUzNTkyMDcyNQ
→",
    "expires_in": 3600,
    "token_type": "bearer",
    "scope": null,
    "refresh_token":
→"MDk2ZmIwODBkYmE3YjNjZWQ4ZTk2NTk2N2JmNjkyZDQ4NzA3YzhiZDQzMjJjODI5MmQ4ZmYxZjlkZmU1ZDNkMQ
→"
}
```

Request for a resource

Put access token in the request header:

```
Authorization: Bearer␣
→NzFiYTM4ZTEwMjcwZTcyZWIzZTA0NmY3NjE3MTIyMjM1Y2NlMmNlNWEyMTAzY2UzYmY0YWIxYmUzNTkyMDcyNQ
```

You can now access any resource you want under /api prefix.

Example

```
curl http://sylius.test/api/v1/users/
    -H "Authorization: Bearer␣
→NzFiYTM4ZTEwMjcwZTcyZWIzZTA0NmY3NjE3MTIyMjM1Y2NlMmNlNWEyMTAzY2UzYmY0YWIxYmUzNTkyMDcyNQ
→"
```

Note: You have to refresh your token after it expires.

Refresh Token

Send request with the following parameters

Definition

```
GET /api/oauth/v2/token
```

Parameter	Parameter type	Description
client_id	query	Public client id
client_secret	query	Client secret
grant_type	query	We will use 'refresh_token' to authorize as user
refresh_token	query	Refresh token generated during authorization

Example

```
curl http://sylius.test/api/oauth/v2/token \
    -d "client_id"=demo_client \
    -d "client_secret"=secret_demo_client \
    -d "grant_type"=refresh_token \
    -d "refresh_token
↪"=MDk2ZmIwODBkYmE3YjNjZWQ4ZTk2NTk2N2JmNjkyZDQ4NzA3YzhiZDQzMjJjODI5MmQ4ZmYxZjlkZmU1ZDNkMQ
```

Exemplary Response

You can now use new token to send requests

```
{
    "access_token":
↪"MWExMWM0NzE1NmUyZDgyZDJiMjEzMmFlMjQ4MzgwMmE4ZTkxYzM0YjdlN2U2YzliNDIyMTk1ZDhlNDYxYWE4Ng
↪",
    "expires_in": 3600,
    "token_type": "bearer",
    "scope": null,
    "refresh_token":
↪"MWI4NzVkNThjZDc2Y2M1N2JiNzBmOTQ0MDFmY2U0YzVjYzllMDE1OTU5OWFiMzJiZTY5NGU4NzYyODU1N2ZjYZ
↪"
}
```

Default values in dev environment

In a developer environment there are default client id, client secret and default access token provided to allow you to test our API just out-of-the-box. In order to access them, please use the following values:

Parameter	Value
client_id	demo_client
client_secret	secret_demo_client
grant_type	password
access_token	SampleToken

These values will be used later on to make it easier for you to check, how our API works.

5.1.3 Admin Users API

These endpoints will allow you to easily manage admin users. Base URI is *api/v1/users*.

Admin User API response structure

If you request an admin user via API, you will receive an object with the following fields:

Field	Description
id	Admin user's id
username	Admin user's name
email	Admin user's email
enabled	Flag set if the user is enabled

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Admin user's id
username	Admin user's name
email	Admin user's email
enabled	Flag set if the user is enabled
usernameCanonical	Username of the admin user in canonical form
emailCanonical	Email of the admin user in canonical form
roles	Roles of the admin user
firstName	The admin user's first name
lastName	The admin user's last name
localeCode	Code of the language, which is used by the admin user

---

Note: Read more about *User model in the component docs*.

Creating an Admin User

To create a new admin user you will need to call the `/api/v1/users/` endpoint with the `POST` method.

Definition

```
POST /api/v1/users/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
username	request	Admin user name
email	request	Admin user email
plainPassword	request	Admin user password
localeCode	request	Code of the language, which is used by the admin user

Example

To create a new admin user use the below method:

```
$ curl http://demo.sylius.com/api/v1/users/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "username": "Balrog",
            "email": "teamEvil@middleearth.com",
            "plainPassword": "youShallNotPass",
            "localeCode": "en_US"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 7,
    "username": "Balrog",
    "usernameCanonical": "balrog",
    "roles": [
        "ROLE_ADMINISTRATION_ACCESS"
    ],
    "email": "teamEvil@middleearth.com",
    "emailCanonical": "teamevil@middleearth.com",
    "enabled": false
}
```

> **Warning:** If you try to create an admin user without username, email, password or locale's code, you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/users/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
  -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
```

```
            "username": {
                "errors": [
                    "Please enter your name."
                ]
            },
            "email": {
                "errors": [
                    "Please enter your email."
                ]
            },
            "plainPassword": {
                "errors": [
                    "Please enter your password."
                ]
            },
            "enabled": {},
            "firstName": {},
            "lastName": {},
            "localeCode": {
                "errors": [
                    "Please choose a locale."
                ]
            }
        }
    }
}
```

You can also create an admin user with additional (not required) fields:

Parameter	Parameter type	Description
enabled	request	Flag set if the user is enabled
firstName	request	The admin user's first name
lastName	request	The admin user's last name

Example

```
$ curl http://demo.sylius.com/api/v1/users/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "firstName": "Balrog",
            "lastName": "of Morgoth",
            "username": "Balrog",
            "email": "teamEvil@middleearth.com",
            "plainPassword": "youShallNotPass",
            "localeCode": "en_US",
            "enabled": "true"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```json
{
    "id": 9,
    "username": "Balrog",
    "usernameCanonical": "balrog",
    "roles": [
        "ROLE_ADMINISTRATION_ACCESS"
    ],
    "email": "teamEvil@middleearth.com",
    "emailCanonical": "teamevil@middleearth.com",
    "enabled": true,
    "firstName": "Balrog",
    "lastName": "of Morgoth"
}
```

Getting a Single Admin User

To retrieve the details of an admin user you will need to call the `/api/v1/users/{id}` endpoint with the `GET` method.

Definition

```
GET /api/v1/users/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the admin user

Example

To see the details for the admin user with `id` = `9` use the below method:

```
$ curl http://demo.sylius.com/api/v1/users/9 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *9* id is an exemplary value. Your value can be different. Check in the list of all admin users if you are not sure which id should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 9,
    "username": "Balrog",
    "usernameCanonical": "balrog",
    "roles": [
        "ROLE_ADMINISTRATION_ACCESS"
    ],
    "email": "teamEvil@middleearth.com",
    "emailCanonical": "teamevil@middleearth.com",
    "enabled": true,
    "firstName": "Balrog",
    "lastName": "of Morgoth"
}
```

Collection of Admin Users

To retrieve a paginated list of admin users you will need to call the `/api/v1/users/` endpoint with the `GET` method.

Definition

```
GET /api/v1/users/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all admin users use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/users/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 1,
    "total": 3,
    "_links": {
        "self": {
            "href": "\/api\/v1\/users\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
```

(continues on next page)

```json
        "first": {
            "href": "\/api\/v1\/users\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
        "last": {
            "href": "\/api\/v1\/users\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 5,
                "username": "sylius",
                "email": "sylius@example.com",
                "enabled": true
            },
            {
                "id": 6,
                "username": "api",
                "email": "api@example.com",
                "enabled": true
            },
            {
                "id": 9,
                "username": "Balrog",
                "email": "teamEvil@middleearth.com",
                "enabled": true
            }
        ]
    }
}
```

Updating an Admin User

To fully update an admin user you will need to call the `/api/v1/users/{id}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/users/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the admin user
username	request	Admin user name
email	request	Admin user email
plainPassword	request	Admin user password
localeCode	request	Code of the language, which is used by the admin user

Example

To fully update the admin user with `id` = `9` use the below method:

```
$ curl http://demo.sylius.com/api/v1/users/9 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "firstName": "Gollum",
            "lastName": "Gollum!",
            "username": "Smeagol",
            "email": "smeagol@middleearth.com",
            "plainPassword": "myPrecious",
            "localeCode": "en_US"
        }
'
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform a full admin user update without all the required fields specified, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/users/9 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "username": {
                "errors": [
                    "Please enter your name."
                ]
            },
            "email": {
                "errors": [
                    "Please enter your email."
                ]
            },
            "plainPassword": {},
            "enabled": {},
            "firstName": {},
```

(continues on next page)

```
            "lastName": {},
            "localeCode": {
                "errors": [
                    "Please choose a locale."
                ]
            }
        }
    }
}
```

To update an admin user partially you will need to call the `/api/v1/users/{id}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/users/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the admin user

Example

To partially update the admin user with `id = 9` use the below method:

```
$ curl http://demo.sylius.com/api/v1/users/9 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "email": "smeagol@ring.com"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting an Admin User

To delete an admin user you will need to call the `/api/v1/users/{id}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/users/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the admin user

Example

To delete the admin user with `id = 9` use the below method:

```
$ curl http://demo.sylius.com/api/v1/users/9 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

> **Warning:**   If you try to delete the admin user which is currently logged in, you will receive a `422 Unprocessable Entity` error.

Example

```
$ curl http://demo.sylius.com/api/v1/users/6 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 422 Unprocessable Entity
```

```
{
    "code": 422,
    "message": "Cannot remove currently logged in user."
}
```

5.1.4 Carts API

These endpoints will allow you to easily manage cart and cart items. Base URI is */api/v1/carts/*.

Note: Remember that a **Cart** in Sylius is an **Order** in the state `cart`.

If you don't understand the difference between Cart and Order concepts in Sylius yet, please read *this article* carefully.

Cart API response structure

If you request a cart via API, you will receive an object with the following fields:

Field	Description
id	Id of the cart
items	List of items in the cart
itemsTotal	Sum of all items prices
adjustments	List of adjustments related to the cart
adjustmentsTotal	Sum of all cart adjustments values
total	Sum of items total and adjustments total
customer	*The customer object serialized with the default data* for cart
channel	*The channel object serialized with the default data*
currencyCode	Currency of the cart
localeCode	Locale of the cart
checkoutState	State of the checkout process of the cart

CartItem API response structure

Each CartItem in an API response will be build as follows:

Field	Description
id	Id of the cart item
quantity	Quantity of item units
unitPrice	Price of each item unit
total	Sum of units total and adjustments total of that cart item
units	A collection of units related to the cart item
unitsTotal	Sum of all units prices of the cart item
adjustments	List of adjustments related to the cart item
adjustmentsTotal	Sum of all item adjustments related to that cart item
variant	*The product variant object serialized with the default data*
_link[product]	Relative link to product
_link[variant]	Relative link to variant
_link[order]	Relative link to order

CartItemUnit API response structure

Each CartItemUnit API response will be build as follows:

Field	Description
id	Id of the cart item unit
adjustments	List of adjustments related to the unit
adjustmentsTotal	Sum of all units adjustments of the unit

Adjustment API response structure

And each Adjustment will be build as follows:

Field	Description
id	Id of the adjustment
type	Type of the adjustment (E.g. *order_promotion* or *tax*)
label	Label of the adjustment
amount	Amount of the adjustment (value)

Note: If it is confusing to you, learn more about *Carts (Orders) in the component docs* and *Adjustments concept*.

Creating a Cart

To create a new cart you will need to call the `/api/v1/carts/` endpoint with the `POST` method.

Definition

```
POST /api/v1/carts/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
customer	request	Email of the related customer
channel	request	Code of the related channel
localeCode	request	Code of the locale in which the cart should be created

Example

To create a new cart for the `shop@example.com` user in the `US_WEB` channel with the `en_US` locale use the below method:

Warning: Remember, that it doesn't replicate the environment of shop usage. It is more like an admin part of cart creation, which will allow you to manage the cart from the admin perspective. ShopAPI is still an experimental concept.

```
$ curl http://demo.sylius.com/api/v1/carts/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "customer": "shop@example.com",
            "channel": "US_WEB",
            "localeCode": "en_US"
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id":21,
    "items":[

    ],
    "itemsTotal":0,
    "adjustments":[

    ],
    "adjustmentsTotal":0,
    "total":0,
    "customer":{
        "id":1,
        "email":"shop@example.com",
        "firstName":"John",
        "lastName":"Doe",
        "user":{
            "id":1,
            "username":"shop@example.com",
            "usernameCanonical":"shop@example.com"
        },
        "_links":{
            "self":{
                "href":"\/api\/v1\/customers\/1"
            }
        }
    },
    "channel":{
        "code":"US_WEB",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/US_WEB"
            }
        }
    },
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"cart"
}
```

Note: A currency code will be added automatically based on the channel settings. Read more about channels *here*.

Warning: If you try to create a resource without localeCode, channel or customer, you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/carts/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code":400,
    "message":"Validation Failed",
    "errors":{
        "children":{
            "customer":{
                "errors":[
                    "This value should not be blank."
                ]
            },
            "localeCode":{
                "errors":[
                    "This value should not be blank."
                ]
            },
            "channel":{
                "errors":[
                    "This value should not be blank."
                ]
            }
        }
    }
}
```

Collection of Carts

To retrieve a paginated list of carts you will need to call the `/api/v1/carts/` endpoint with the `GET` method.

Definition

```
GET /api/v1/carts/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of carts displayed per page, by default = 10

Example

To see the first page of the paginated carts collection use the below method:

```
$ curl http://demo.sylius.com/api/v1/carts/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page":1,
    "limit":10,
    "pages":1,
    "total":1,
    "_links":{
        "self":{
            "href":"\/api\/v1\/carts\/?page=1&limit=10"
        },
        "first":{
            "href":"\/api\/v1\/carts\/?page=1&limit=10"
        },
        "last":{
            "href":"\/api\/v1\/carts\/?page=1&limit=10"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":21,
                "items":[

                ],
                "itemsTotal":0,
                "adjustments":[

                ],
                "adjustmentsTotal":0,
                "total":0,
                "customer":{
                    "id":1,
                    "email":"shop@example.com",
                    "firstName":"John",
                    "lastName":"Doe",
                    "user":{
                        "id":1,
                        "username":"shop@example.com",
                        "enabled":true
                    },
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/customers\/1"
                        }
```

```
                }
            },
            "channel":{
                "id":1,
                "code":"US_WEB",
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/channels\/US_WEB"
                    }
                }
            },
            "currencyCode":"USD",
            "localeCode":"en_US",
            "checkoutState":"cart"
        }
    ]
  }
}
```

Getting a Single Cart

To retrieve details of the cart you will need to call the `/api/v1/carts/{id}` endpoint with `GET` method.

Definition

```
GET /api/v1/carts/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart

Example

To see details of the cart with `id = 21` use the below method:

```
$ curl http://demo.sylius.com/api/v1/carts/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *21* value was taken from the previous create response. Your value can be different. Check in the list of all carts if you are not sure which id should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id":21,
    "items":[

    ],
    "itemsTotal":0,
    "adjustments":[

    ],
    "adjustmentsTotal":0,
    "total":0,
    "customer":{
        "id":1,
        "email":"shop@example.com",
        "firstName":"John",
        "lastName":"Doe",
        "user":{
            "id":1,
            "username":"shop@example.com",
            "usernameCanonical":"shop@example.com"
        },
        "_links":{
            "self":{
                "href":"\/api\/v1\/customers\/1"
            }
        }
    },
    "channel":{
        "code":"US_WEB",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/US_WEB"
            }
        }
    },
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"cart"
}
```

Deleting a Cart

To delete a cart you will need to call the `/api/v1/carts/{id}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/carts/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart

Example

To delete the cart with `id = 21` use the below method:

```
$ curl http://demo.sylius.com/api/v1/carts/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Note: Remember the *21* value comes from the previous example. Here we are deleting a previously fetched cart, so it is the same id.

Exemplary Response

```
STATUS: 204 No Content
```

Creating a Cart Item

To add a new cart item to an existing cart you will need to call the `/api/v1/carts/{cartId}/items/` endpoint with `POST` method.

Definition

```
POST /api/v1/carts/{cartId}/items/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
cartId	url attribute	Id of the requested cart
variant	request	Code of the item you want to add to the cart
quantity	request	Amount of variants you want to add to the cart (cannot be < 1)

Example

To add a new item of a variant with code `MEDIUM_MUG_CUP` to the cart with id = 21 (assuming, that we didn't remove it in the previous example) use the below method:

```
$ curl http://demo.sylius.com/api/v1/carts/21/items/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "variant": "MEDIUM_MUG_CUP",
            "quantity": 1
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```json
{
    "id":57,
    "quantity":1,
    "unitPrice":250,
    "total":250,
    "units":[
        {
            "id":165,
            "adjustments":[

            ],
            "adjustmentsTotal":0
        }
    ],
    "unitsTotal":250,
    "adjustments":[

    ],
    "adjustmentsTotal":0,
    "variant":{
        "id":331,
        "code":"MEDIUM_MUG_CUP",
        "optionValues":[
            {
                "code":"mug_type_medium",
                "translations":{
                    "en_US":{
                        "id":1,
                        "value":"Medium mug"
                    }
                }
            }
        ],
        "position":2,
        "translations":{
            "en_US":{
                "id":331,
                "name":"Medium Mug"
            }
        },
        "tracked":false,
        "channelPricings":{
            "US_WEB": {
                "channelCode": "US_WEB",
                "price":250
            }
        }
    },
    "_links":{
        "order":{
            "href":"\/api\/v1\/orders\/21"
        },
        "product":{
```

```
            "href":"\/api\/v1\/products\/07f2044a-855d-3c56-9274-b5167c2d5809"
        },
        "variant":{
            "href":"\/api\/v1\/products\/07f2044a-855d-3c56-9274-b5167c2d5809\/
↪variants\/MEDIUM_MUG_CUP"
        }
    }
}
```

Tip: In Sylius the prices are stored as an integers (`1059` represents `10.59`$). So in order to present a proper amount to the end user, you should divide price by 100 by default.

Updating a Cart Item

To change the quantity of a cart item you will need to call the `/api/v1/carts/{cartId}/items/{cartItemId}` endpoint with the `PUT` or `PATCH` method.

Definition

```
PUT /api/v1/carts/{cartId}/items/{cartItemId}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
cartId	url attribute	Id of the requested cart
cartItemId	url attribute	Id of the requested cart item
quantity	request	Amount of items you want to have in the cart (cannot be < 1)

Example

To change the quantity of the cart item with `id = 57` in the cart of `id = 21` to 3 use the below method:

```
$ curl http://demo.sylius.com/api/v1/carts/21/items/57 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '{"quantity": 3}'
```

Tip: If you are not sure where does the value **58** come from, check the previous response, and look for the cart item id.

Exemplary Response

```
STATUS: 204 No Content
```

Now we can check how does the cart look like after changing the quantity of a cart item.

```
$ curl http://demo.sylius.com/api/v1/carts/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id":21,
    "items":[
        {
            "id":57,
            "quantity":3,
            "unitPrice":250,
            "total":750,
            "units":[
                {
                    "id":165,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0
                },
                {
                    "id":166,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0
                },
                {
                    "id":167,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0
                }
            ],
            "unitsTotal":750,
            "adjustments":[

            ],
            "adjustmentsTotal":0,
            "variant":{
                "id":331,
                "code":"MEDIUM_MUG_CUP",
                "optionValues":[
                    {
                        "code":"mug_type_medium",
                        "translations":{
                            "en_US":{
                                "id":1,
                                "value":"Medium mug"
                            }
```

```
                }
            }
        ],
        "position":2,
        "translations":{
            "en_US":{
                "id":331,
                "name":"Medium Mug"
            }
        },
        "tracked":false,
        "channelPricings":{
            "US_WEB": {
                "channelCode": "US_WEB",
                "price":250
            }
        }
    },
    "_links":{
        "order":{
            "href":"\/api\/v1\/orders\/21"
        },
        "product":{
            "href":"\/api\/v1\/products\/07f2044a-855d-3c56-9274-b5167c2d5809"
        },
        "variant":{
            "href":"\/api\/v1\/products\/07f2044a-855d-3c56-9274-
→b5167c2d5809\/variants\/MEDIUM_MUG_CUP"
        }
    }
    }
    }
    ],
    "itemsTotal":750,
    "adjustments":[
        {
            "id":181,
            "type":"shipping",
            "label":"UPS",
            "amount":157
        }
    ],
    "adjustmentsTotal":157,
    "total":907,
    "customer":{
        "id":1,
        "email":"shop@example.com",
        "firstName":"John",
        "lastName":"Doe",
        "user":{
            "id":1,
            "username":"shop@example.com",
            "usernameCanonical":"shop@example.com"
        },
        "_links":{
            "self":{
                "href":"\/api\/v1\/customers\/1"
            }
```

```
        }
    },
    "channel":{
        "code":"US_WEB",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/US_WEB"
            }
        }
    },
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"cart"
}
```

Tip: In this response you can see that promotion and shipping have been taken into account to calculate the appropriate price.

Deleting a Cart Item

To delete a cart item from a cart you will need to call the `/api/v1/carts/{cartId}/items/{cartItemId}` endpoint with the `DELETE` method.

Definition

To delete the cart item with `id = 58` from the cart with `id = 21` use the below method:

```
DELETE /api/v1/carts/{cartId}/items/{cartItemId}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
cartId	url attribute	Id of the requested cart
cartItemId	url attribute	Id of the requested cart item

Example

```
$ curl http://demo.sylius.com/api/v1/carts/21/items/58 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.5 Channels API

These endpoints will allow you to easily manage channels. Base URI is */api/v1/channels*.

Channel API response structure

If you request a channel via API, you will receive an object with the following fields:

Field	Description
id	Id of the channel
code	Unique channel identifier

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the channel
code	Unique channel identifier
taxCalculationStrategy	Strategy which will be applied during processing orders in the channel
name	Name of the channel
hostname	Name of the host for the channel
enabled	Gives an information about channel availability
description	Description of the channel
color	Allows to recognize orders made in the channel
createdAt	The channel's creation date
updatedAt	The channel's last updating date

Note: Read more about *Channels docs*.

Getting a Single Channel

To retrieve the details of a specific channel you will need to call the `/api/v1/channels/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/channels/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested channel

Example

To see the details of the channel with `code = US_WEB` use the below method:

```
$ curl http://demo.sylius.com/api/v1/channels/US_WEB \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *US_WEB* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "code": "US_WEB",
    "name": "US Web Store",
    "hostname": "localhost",
    "color": "Wheat",
    "createdAt": "2017-02-10T13:14:20+0100",
    "updatedAt": "2017-02-10T13:14:20+0100",
    "enabled": true,
    "taxCalculationStrategy": "order_items_based",
    "_links": {
        "self": {
            "href": "\/api\/v1\/channels\/US_WEB"
        }
    }
}
```

5.1.6 Checkout API

These endpoints will allow you to go through the order checkout from the admin perspective. It can be useful for integrations with tools like Twillo or an inspiration for your custom Shop API. Base URI is */api/v1/checkouts/*.

After you create a cart (an empty order) and add some items to it, you can start the checkout via API. This basically means updating the order with concrete information, step by step, in a correct order.

Sylius checkout flow is built from 4 steps, which have to be done in a certain order (unless you will customize it).

Step	Description
addressing	Shipping and billing addresses are assigned to the cart
shipping	Choosing a shipping method from the available ones
payment	Choosing a payment method from the available ones
finalize	The order is built and its data can be confirmed

Tip: If you are not familiar with the concept of checkout in Sylius, please carefully read *this article* first.

Note: We do not present the order serialization in this chapter, because it is the same order serialization as described in *the article about orders*.

Addressing step

After you added some items to the cart, to start the checkout you simply need to provide a shipping address. You can also specify a different billing address if needed.

Definition

```
PUT /api/v1/checkouts/addressing/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart
differentBillingAd-dress	request	If false, the billing address fields are not required and data from the shipping address is copied
shippingAd-dress[firstName]	request	First name for the shipping address
shippingAd-dress[lastName]	request	Last name for the shipping address
shippingAd-dress[city]	request	City name
shippingAd-dress[postcode]	request	Postcode
shippingAd-dress[street]	request	Street
shippingAd-dress[country]	request	Id of the country
shippingAd-dress[province]	request	*(optional)* Id of the province
billingAd-dress[firstName]	request	*(optional)* First name for the billing address
billingAd-dress[lastName]	request	*(optional)* Last name for the billing address
billingAddress[city]	request	*(optional)* City name
billingAd-dress[postcode]	request	*(optional)* Postcode
billingAddress[street]	request	*(optional)* Street
billingAd-dress[country]	request	*(optional)* Id of the country
billingAd-dress[province]	request	*(optional)* Id of the province

Note: Remember a cart with *id = 21 for the Cart API documentation*? We will take the same cart as an exemplary cart for checkout process.

Example

To address the cart for a user that lives in `Los Angeles` in the United States, the following snippet can be used:

```
$ curl http://demo.sylius.com/api/v1/checkouts/addressing/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "shippingAddress": {
                "firstName": "Elon",
                "lastName": "Musk",
                "street": "10941 Savona Rd",
                "countryCode": "US",
                "city": "'Los Angeles",
                "postcode": "CA 90077"
            },
            "differentBillingAddress": false
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Now you can check the state of the order, by asking for the checkout summary:

Example

To check the checkout process state for the cart with *id = 21*, we need to execute this command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 Ok
```

```
{
    "id":21,
    "items":[
        {
            "id":74,
            "quantity":1,
            "unitPrice":100000,
            "total":100000,
            "units":[
                {
                    "id":228,
                    "adjustments":[
                    ],
                    "adjustmentsTotal":0
                }
```

(continues on next page)

```
            ],
            "unitsTotal":100000,
            "adjustments":[
            ],
            "adjustmentsTotal":0,
            "variant":{
                "id":331,
                "code":"MEDIUM_MUG_CUP",
                "optionValues":[
                    {
                        "code":"mug_type_medium"
                    }
                ],
                "position":2,
                "translations":{
                    "en_US":{
                    }
                },
                "onHold":0,
                "onHand":10,
                "tracked":false,
                "channelPricings":{
                    "US_WEB":{
                        "channelCode": "US_WEB",
                        "price":100000
                    }
                }
            },
            "_links":{
                "product":{
                    "href":"\/api\/v1\/products\/5"
                },
                "variant":{
                    "href":"\/api\/v1\/products\/5\/variants\/331"
                }
            }
        }
    ],
    "itemsTotal":100000,
    "adjustments":[
        {
            "id":249,
            "type":"shipping",
            "label":"UPS",
            "amount":8787
        }
    ],
    "adjustmentsTotal":8787,
    "total":108787,
    "state":"cart",
    "customer":{
        "id":1,
        "email":"shop@example.com",
        "emailCanonical":"shop@example.com",
        "firstName":"John",
        "lastName":"Doe",
        "gender":"u",
```

```
        "user":{
            "id":1,
            "username":"shop@example.com",
            "usernameCanonical":"shop@example.com",
            "roles":[
                "ROLE_USER"
            ],
            "enabled":true
        },
        "_links":{
            "self":{
                "href":"\/api\/v1\/customers\/1"
            }
        }
    },
    "channel":{
        "id":1,
        "code":"US_WEB",
        "name":"US Web Store",
        "hostname":"localhost",
        "color":"MediumPurple",
        "createdAt":"2017-02-14T11:10:02+0100",
        "updatedAt":"2017-02-14T11:10:02+0100",
        "enabled":true,
        "taxCalculationStrategy":"order_items_based",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/1"
            }
        }
    },
    "shippingAddress":{
        "firstName":"Elon",
        "lastName":"Musk",
        "countryCode":"US",
        "street":"10941 Savona Rd",
        "city":"\u2019Los Angeles",
        "postcode":"CA 90077"
    },
    "billingAddress":{
        "firstName":"Elon",
        "lastName":"Musk",
        "countryCode":"US",
        "street":"10941 Savona Rd",
        "city":"\u2019Los Angeles",
        "postcode":"CA 90077"
    },
    "payments":[
        {
            "id":21,
            "method":{
                "id":1,
                "code":"cash_on_delivery"
            },
            "amount":108787,
            "state":"cart"
        }
```

```
    ],
    "shipments":[
        {
            "id":21,
            "state":"cart",
            "method":{
                "code":"ups",
                "enabled":true
            }
        }
    ],
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"addressed"
}
```

Of course, you can specify different shipping and billing addresses. If our user Elon would like to send a gift to the NASA administrator, Frederick D. Gregory, he could send the following request:

```
$ curl http://demo.sylius.com/api/v1/checkouts/addressing/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "shippingAddress": {
                "firstName": " Frederick D.",
                "lastName": "Gregory",
                "street": "300 E St SW",
                "countryCode": "US",
                "city": "'Washington",
                "postcode": "DC 20546"
            },
            "differentBillingAddress": true,
            "billingAddress": {
                "firstName": "Elon",
                "lastName": "Musk",
                "street": "10941 Savona Rd",
                "countryCode": "US",
                "city": "'Los Angeles",
                "postcode": "CA 90077"
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Shipping step

When the order contains the address information, we are able to determine the available shipping methods. First, we need to get the available shipping methods to have our choice list:

Definition

```
GET /api/v1/checkouts/select-shipping/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart

Example

To check available shipping methods for the previously addressed cart, you can use the following command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/select-shipping/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json"
```

```
STATUS: 200 OK
```

```json
{
    "shipments":[
        {
            "methods":[
                {
                    "id":1,
                    "code":"ups",
                    "name":"UPS",
                    "description":"Dolorem consequatur itaque neque non voluptas
→dolor.",
                    "price":8787
                },
                {
                    "id":2,
                    "code":"dhl_express",
                    "name":"DHL Express",
                    "description":"Voluptatem ipsum dolor vitae corrupti eum repellat.
→",
                    "price":3549
                },
                {
                    "id":3,
                    "code":"fedex",
                    "name":"FedEx",
                    "description":"Qui nostrum minus accusantium molestiae voluptatem
→eaque.",
                    "price":3775
                }
            ]
        }
    ]
}
```

The response contains proposed shipments and for each of them, it has a list of the available shipping methods alongside their calculated prices.

> **Warning:** Because of the custom calculation logic, the regular rules of overriding do not apply for this endpoint. In order to have a different response, you have to provide a custom controller and build the message on your own. Exemplary implementation can be found here

Next step is updating the order with the types of shipping methods that have been selected. A PUT request has to be send for each available shipment.

Definition

```
PUT /api/v1/checkouts/select-shipping/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart
ship-ments[X]['method']	request	Code of the chosen shipping method (Where X is the number of shipment in the returned array)

Example

To choose the *DHL Express* method for our shipment (the cheapest one), we can use the following snippet:

```
$ curl http://demo.sylius.com/api/v1/checkouts/select-shipping/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "shipments": [
                {
                    "method": "dhl_express"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

While checking for the checkout process state of the cart with *id = 21*, you will get the following response:

Exemplary Response

```
STATUS: 200 OK

{
```

(continues on next page)

```
    "id":21,
    "items":[
        {
            "id":74,
            "quantity":1,
            "unitPrice":100000,
            "total":100000,
            "units":[
                {
                    "id":228,
                    "adjustments":[
                    ],
                    "adjustmentsTotal":0
                }
            ],
            "unitsTotal":100000,
            "adjustments":[
            ],
            "adjustmentsTotal":0,
            "variant":{
                "id":331,
                "code":"MEDIUM_MUG_CUP",
                "optionValues":[
                    {
                        "code":"mug_type_medium"
                    }
                ],
                "position":2,
                "translations":{
                    "en_US":{
                    }
                },
                "onHold":0,
                "onHand":10,
                "tracked":false,
                "channelPricings":{
                    "US_WEB":{
                        "channelCode": "US_WEB",
                        "price":100000
                    }
                }
            },
            "_links":{
                "product":{
                    "href":"\/api\/v1\/products\/5"
                },
                "variant":{
                    "href":"\/api\/v1\/products\/5\/variants\/331"
                }
            }
        }
    ],
    "itemsTotal":100000,
    "adjustments":[
        {
            "id":251,
            "type":"shipping",
```

```
                    "label":"DHL Express",
                    "amount":3549
                }
            ],
            "adjustmentsTotal":3549,
            "total":103549,
            "state":"cart",
            "customer":{
                "id":1,
                "email":"shop@example.com",
                "emailCanonical":"shop@example.com",
                "firstName":"John",
                "lastName":"Doe",
                "gender":"u",
                "user":{
                    "id":1,
                    "username":"shop@example.com",
                    "usernameCanonical":"shop@example.com",
                    "roles":[
                        "ROLE_USER"
                    ],
                    "enabled":true
                },
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/customers\/1"
                    }
                }
            },
            "channel":{
                "id":1,
                "code":"US_WEB",
                "name":"US Web Store",
                "hostname":"localhost",
                "color":"MediumPurple",
                "createdAt":"2017-02-14T11:10:02+0100",
                "updatedAt":"2017-02-14T11:10:02+0100",
                "enabled":true,
                "taxCalculationStrategy":"order_items_based",
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/channels\/1"
                    }
                }
            },
            "shippingAddress":{
                "firstName":"Frederick D.",
                "lastName":"Gregory",
                "countryCode":"US",
                "street":"300 E St SW",
                "city":"\u2019Washington",
                "postcode":"DC 20546"
            },
            "billingAddress":{
                "firstName":"Frederick D.",
                "lastName":"Gregory",
                "countryCode":"US",
```

```
        "street":"300 E St SW",
        "city":"\u2019Washington",
        "postcode":"DC 20546"
    },
    "payments":[
        {
            "id":21,
            "method":{
                "id":1,
                "code":"cash_on_delivery"
            },
            "amount":103549,
            "state":"cart"
        }
    ],
    "shipments":[
        {
            "id":21,
            "state":"cart",
            "method":{
                "code":"dhl_express",
                "enabled":true
            }
        }
    ],
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"shipping_selected"
}
```

Payment step

When we are done with shipping choices and we know the final price of an order, we can select a payment method.

Definition

```
GET /api/v1/checkouts/select-payment/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart

Warning: Similar to the shipping step, this one has its own controller, which has to be replaced if you want to make some changes. Exemplary implementation can be found *here*

Example

To check available payment methods for the cart that has a shipping methods assigned, we need to execute this curl command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/select-payment/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json"
```

```
STATUS: 200 OK
```

```json
{
    "payments":[
        {
            "methods":[
                {
                    "id":1,
                    "code":"cash_on_delivery",
                    "name":"Cash on delivery",
                    "description":"Ipsum dolor non esse quia sit."
                },
                {
                    "id":2,
                    "code":"bank_transfer",
                    "name":"Bank transfer",
                    "description":"Perspiciatis itaque earum quisquam ut dolor."
                }
            ]
        }
    ]
}
```

With that information, another PUT request with the id of payment method is enough to proceed:

Definition

```
PUT /api/v1/checkouts/select-payment/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart
payment[X]['method']	request	Code of chosen payment method

Example

To choose the Bank transfer method for our shipment, simply use the following code:

```
$ curl http://demo.sylius.com/api/v1/checkouts/select-payment/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "payments": [
                {
                    "method": "bank_transfer"
                }
```

```
            ]
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Finalize step

After choosing the payment method we are ready to finalize the cart and make an order. Now, you can get its snapshot by calling a `GET` request:

Tip: The same definition has been used over this chapter, to see the current state of the order.

Definition

```
GET /api/v1/checkouts/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart

Example

To check the fully constructed cart with *id = 21*, use the following command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json"
```

```
STATUS: 200 OK
```

```json
{
    "id":21,
    "items":[
        {
            "id":74,
            "quantity":1,
            "unitPrice":100000,
            "total":100000,
            "units":[
                {
                    "id":228,
                    "adjustments":[
```

```
                    ],
                    "adjustmentsTotal":0
                }
            ],
            "unitsTotal":100000,
            "adjustments":[
            ],
            "adjustmentsTotal":0,
            "variant":{
                "id":331,
                "code":"MEDIUM_MUG_CUP",
                "optionValues":[
                    {
                        "code":"mug_type_medium"
                    }
                ],
                "position":2,
                "translations":{
                    "en_US":{
                    }
                },
                "onHold":0,
                "onHand":10,
                "tracked":false,
                "channelPricings":{
                    "US_WEB":{
                        "channelCode":"US_WEB",
                        "price":100000
                    }
                }
            },
            "_links":{
                "product":{
                    "href":"\/api\/v1\/products\/5"
                },
                "variant":{
                    "href":"\/api\/v1\/products\/5\/variants\/331"
                }
            }
        }
    ],
    "itemsTotal":100000,
    "adjustments":[
        {
            "id":252,
            "type":"shipping",
            "label":"DHL Express",
            "amount":3549
        }
    ],
    "adjustmentsTotal":3549,
    "total":103549,
    "state":"cart",
    "customer":{
        "id":1,
        "email":"shop@example.com",
        "emailCanonical":"shop@example.com",
```

```json
            "firstName":"John",
            "lastName":"Doe",
            "gender":"u",
            "user":{
                "id":1,
                "username":"shop@example.com",
                "usernameCanonical":"shop@example.com",
                "roles":[
                    "ROLE_USER"
                ],
                "enabled":true
            },
            "_links":{
                "self":{
                    "href":"\/api\/v1\/customers\/1"
                }
            }
        },
        "channel":{
            "id":1,
            "code":"US_WEB",
            "name":"US Web Store",
            "hostname":"localhost",
            "color":"MediumPurple",
            "createdAt":"2017-02-14T11:10:02+0100",
            "updatedAt":"2017-02-14T11:10:02+0100",
            "enabled":true,
            "taxCalculationStrategy":"order_items_based",
            "_links":{
                "self":{
                    "href":"\/api\/v1\/channels\/1"
                }
            }
        },
        "shippingAddress":{
            "firstName":"Frederick D.",
            "lastName":"Gregory",
            "countryCode":"US",
            "street":"300 E St SW",
            "city":"\u2019Washington",
            "postcode":"DC 20546"
        },
        "billingAddress":{
            "firstName":"Frederick D.",
            "lastName":"Gregory",
            "countryCode":"US",
            "street":"300 E St SW",
            "city":"\u2019Washington",
            "postcode":"DC 20546"
        },
        "payments":[
            {
                "id":21,
                "method":{
                    "id":2,
                    "code":"bank_transfer"
                },
```

```
                "amount":103549,
                "state":"cart"
            }
        ],
        "shipments":[
            {
                "id":21,
                "state":"cart",
                "method":{
                    "code":"dhl_express",
                    "enabled":true
                }
            }
        ],
        "currencyCode":"USD",
        "localeCode":"en_US",
        "checkoutState":"payment_selected"
}
```

This is how your final order will look like. If you are satisfied with that response, simply call another `PUT` request to confirm the checkout, which will become a real order and appear in the backend.

Definition

```
PUT /api/v1/checkouts/complete/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested cart
notes	request	*(optional)* Notes that should be attached to the order

Example

To finalize the previously built order, execute the following command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/complete/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 204 No Content
```

The order has been placed, from now on you can manage it only via orders endpoint.

Of course the same result can be achieved when the order will be completed with some additional notes:

Example

To finalize the previously built order (assuming that, the previous example has not been executed), try the following command:

```
$ curl http://demo.sylius.com/api/v1/checkouts/complete/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "notes": "Please, call me before delivery"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.7 Countries API

These endpoints will allow you to easily manage countries. Base URI is */api/v1/countries*.

Country API response structure

If you request a country via API, you will receive an object with the following fields:

Field	Description
id	Id of the country
code	Unique country identifier

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the country
code	Unique country identifier
enabled	Information says if the country is enabled (default: false)
provinces	Collection of the country's provinces

Note: Read more about *Countries in the component docs*.

Creating a Country

To create a new country you will need to call the `/api/v1/countries/` endpoint with the `POST` method.

Definition

```
POST /api/v1/countries/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Country identifier

Example

```
$ curl http://demo.sylius.com/api/v1/countries/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "PL"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 2,
    "code": "PL",
    "provinces": [],
    "enabled": false,
    "_links": {
        "self": {
            "href": "\/api\/v1\/countries\/PL"
        }
    }
}
```

If you try to create a country without code you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/countries/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "provinces": {},
            "enabled": {},
            "code": {
                "errors": [
                    "Please enter country ISO code."
                ]
            }
        }
    }
}
```

You can also create a country with additional (not required) fields:

Parameter	Parameter type	Description
enabled	request	Information says if the country is enabled (default: false)
provinces	request	Collection of the country's provinces

Example

```
$ curl http://demo.sylius.com/api/v1/countries/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code":"PL",
            "enabled": true,
            "provinces": [
                {
                    "name": "mazowieckie",
                    "code": "PL-MZ"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 4,
```

```json
    "code": "PL",
    "provinces": [
        {
            "id": 1,
            "code": "PL-MZ",
            "name": "mazowieckie",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/countries\/PL\/provinces\/PL-MZ"
                },
                "country": {
                    "href": "\/api\/v1\/countries\/PL"
                }
            }
        }
    ],
    "enabled": true,
    "_links": {
        "self": {
            "href": "\/api\/v1\/countries\/PL"
        }
    }
}
```

Getting a Single Country

To retrieve the details of a country you will need to call the `/api/v1/countries/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/countries/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested country

Example

To see the details of the country with `code = US` use the below method:

```
$ curl http://demo.sylius.com/api/v1/countries/US \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *US* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "code": "US",
    "provinces": [],
    "enabled": true,
    "_links": {
        "self": {
            "href": "\/api\/v1\/countries\/US"
        }
    }
}
```

Collection of Countries

To retrieve a paginated list of countries you will need to call the `/api/v1/countries/` endpoint with the `GET`
method.

Definition

```
GET /api/v1/countries/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all countries use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/countries/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 2,
```

(continues on next page)

```json
    "_links": {
        "self": {
            "href": "\/api\/v1\/countries\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/countries\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/countries\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "US",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/countries\/US"
                    }
                }
            },
            {
                "id": 4,
                "code": "PL",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/countries\/PL"
                    }
                }
            }
        ]
    }
}
```

Deleting a Country

To delete a country you will need to call the `/api/v1/countries/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/countries/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the removed country

Example

```
$ curl http://demo.sylius.com/api/v1/countries/PL \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.8 Currencies API

These endpoints will allow you to easily manage currencies. Base URI is */api/v1/currencies*.

Currency API response structure

If you request a currency via API, you will receive an object with the following fields:

Field	Description
id	Id of the currency
code	Unique currency identifier

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the currency
code	Unique currency identifier
updatedAt	Last update date of the currency
createdAt	Creation date of the currency

Note: Read more about *Currencies in the component docs*.

Creating a Currency

To create a new currency you will need to call the `/api/v1/currencies/` endpoint with the `POST` method.

Definition

```
POST /api/v1/currencies/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Currency identifier

Example

```
$ curl http://demo.sylius.com/api/v1/currencies/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "PLN"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 4,
    "code": "PLN",
    "createdAt": "2017-02-14T11:38:40+0100",
    "updatedAt": "2017-02-14T11:38:41+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/currencies\/PLN"
        }
    }
}
```

If you try to create a currency without code you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/currencies/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "code": {
                "errors": [
                    "Please choose currency code."
                ]
```

(continues on next page)

```
                }
            }
        }
}
```

Getting a Single Currency

To retrieve the details of a currency you will need to call the `/api/v1/currencies/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/currencies/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested currency

Example

To see the details of the currency with `code = PLN` use the below method:

```
$ curl http://demo.sylius.com/api/v1/currencies/PLN \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *PLN* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 4,
    "code": "PLN",
    "createdAt": "2017-02-14T11:38:40+0100",
    "updatedAt": "2017-02-14T11:38:41+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/currencies\/PLN"
        }
    }
}
```

Collection of Currencies

To retrieve a paginated list of currencies you will need to call the `/api/v1/currencies/` endpoint with the `GET` method.

Definition

```
GET /api/v1/currencies/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all currencies use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/currencies/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 3,
    "_links": {
        "self": {
            "href": "\/api\/v1\/currencies\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/currencies\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/currencies\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 3,
                "code": "USD",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/currencies\/USD"
```

```
                }
            }
        },
        {
            "id": 4,
            "code": "PLN",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/currencies\/PLN"
                }
            }
        },
        {
            "id": 5,
            "code": "EUR",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/currencies\/EUR"
                }
            }
        }
    ]
  }
}
```

Deleting a Currency

To delete a currency you will need to call the `/api/v1/currencies/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/currencies/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the removed currency

Example

```
$ curl http://demo.sylius.com/api/v1/currencies/PLN \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.9 Customers API

These endpoints will allow you to easily manage customers. Base URI is */api/v1/customers*. The Customer class is strongly coupled with the User class. Because of that we recommend these endpoints to manage all related to user actions.

When you get a collection of resources, "Default" serialization group will be used and the following fields will be exposed:

Field	Description
id	Id of customer
user[id]	*(optional)* Id of related user
user[username]	*(optional)* Users username
user[enabled]	*(optional)* Flag set if user is enabled
email	Customers email
firstName	Customers first name
lastName	Customers last name

If you request for a more detailed data, you will receive an object with following fields:

Field	Description
id	Id of customer
user[id]	*(optional)* Id of related user
user[username]	*(optional)* Users username
user[usernameCanonical]	*(optional)* Canonicalized users username
user[roles]	*(optional)* Array of users roles
user[enabled]	*(optional)* Flag set if user is enabled
email	Customers email
emailCanonical	Canonicalized customers email
firstName	Customers first name
lastName	Customers last name
gender	Customers gender
birthday	Customers birthday
group	Customer group code

Note: Read more about *Customers and Users*.

Creating a Customer

Definition

```
POST /api/v1/customers/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
email	request	**(unique)** Customer's email
firstName	request	Customer's first name
lastName	request	Customer's last name
group	request	*(optional)* Customer group code
gender	request	Customer's gender
birthday	request	*(optional)* Customer's birthday
user[plainPassword]	request	*(optional)* Users plain password. Required if user account should be created together with customer
user[authorizationRoles]	request	*(optional)* Array of users roles
user[enabled]	request	*(optional)* Flag set if user is enabled

Example

```
$ curl http://demo.sylius.com/api/v1/customers/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "firstName": "John",
            "lastName": "Diggle",
            "email": "john.diggle@yahoo.com",
            "gender": "m",
            "user": {
                "plainPassword" : "testPassword"
            }
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id":409,
    "user":{
        "id":405,
        "username":"john.diggle@yahoo.com",
        "roles":[
            "ROLE_USER"
        ],
        "enabled":false
    },
    "email":"john.diggle@yahoo.com",
    "emailCanonical":"john.diggle@yahoo.com",
    "firstName":"John",
    "lastName":"Diggle",
    "gender":"m",
```

```
    "group":{}
}
```

If you try to create a customer without email or gender, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/customers/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "firstName": {},
            "lastName": {},
            "email": {
                "errors": [
                    "Please enter your email."
                ]
            },
            "birthday": {},
            "gender": {
                "errors": [
                    "Please choose your gender."
                ]
            },
            "phoneNumber": {},
            "subscribedToNewsletter": {},
            "group": {}
        }
    }
}
```

Getting a Single Customer

You can request detailed customer information by executing the following request:

Definition

```
GET /api/v1/customers/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested resource

Example

```
$ curl http://demo.sylius.com/api/v1/customers/399 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id":399,
    "user":{
        "id":398,
        "username":"cgulgowski@example.com",
        "usernameCanonical":"cgulgowski@example.com",
        "roles":[
            "ROLE_USER"
        ],
        "enabled":false
    },
    "email":"cgulgowski@example.com",
    "emailCanonical":"cgulgowski@example.com",
    "firstName":"Levi",
    "lastName":"Friesen",
    "gender":"u",
    "group":{}
}
```

Collection of Customers

You can retrieve the full customers list by making the following request:

Definition

```
GET /api/v1/customers/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
limit	query	*(optional)* Number of items to display per page, by default = 10

Example

```
$ curl http://demo.sylius.com/api/v1/customers/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page":1,
    "limit":10,
    "pages":21,
    "total":205,
    "_links":{
        "self":{
            "href":"\/api\/customers\/?page=1&limit=10"
        },
        "first":{
            "href":"\/api\/customers\/?page=1&limit=10"
        },
        "last":{
            "href":"\/api\/customers\/?page=21&limit=10"
        },
        "next":{
            "href":"\/api\/customers\/?page=2&limit=10"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":407,
                "email":"random@gmail.com",
                "firstName":"Random",
                "lastName":"Doe"
            },
            {
                "id":406,
                "email":"customer@email.com",
                "firstName":"Alexanne",
                "lastName":"Blick"
            },
            {
                "id":405,
                "user":{
                    "id":404,
                    "username":"gaylord.bins@example.com",
                    "enabled":true
                },
                "email":"gaylord.bins@example.com",
                "firstName":"Dereck",
                "lastName":"McDermott"
            },
            {
```

(continues on next page)

```json
            "id":404,
            "user":{
                "id":403,
                "username":"lehner.gerhard@example.com",
                "enabled":false
            },
            "email":"lehner.gerhard@example.com",
            "firstName":"Benton",
            "lastName":"Satterfield"
        },
        {
            "id":403,
            "user":{
                "id":402,
                "username":"raheem.ratke@example.com",
                "enabled":false
            },
            "email":"raheem.ratke@example.com",
            "firstName":"Rusty",
            "lastName":"Jerde"
        },
        {
            "id":402,
            "user":{
                "id":401,
                "username":"litzy.morissette@example.com",
                "enabled":false
            },
            "email":"litzy.morissette@example.com",
            "firstName":"Omer",
            "lastName":"Schaden"
        },
        {
            "id":401,
            "user":{
                "id":400,
                "username":"bbeer@example.com",
                "enabled":true
            },
            "email":"bbeer@example.com",
            "firstName":"Willard",
            "lastName":"Hand"
        },
        {
            "id":400,
            "user":{
                "id":399,
                "username":"qtrantow@example.com",
                "enabled":false
            },
            "email":"qtrantow@example.com",
            "firstName":"Caterina",
            "lastName":"Koelpin"
        },
        {
            "id":399,
            "user":{
```

```
                              "id":398,
                              "username":"cgulgowski@example.com",
                              "enabled":false
                          },
                          "email":"cgulgowski@example.com",
                          "firstName":"Levi",
                          "lastName":"Friesen"
                      }
              ]
          }
}
```

Updating a Customer

You can request full or partial update of resource. For full customer update, you should use PUT method.

Definition

```
PUT /api/v1/customers/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested resource
email	request	**(unique)** Customers email
firstName	request	Customers first name
lastName	request	Customers last name
group	request	*(optional)* Customer group code
gender	request	Customers gender
birthday	request	*(optional)* Customers birthday
user[plainPassword]	request	*(optional)* Users plain password. Required if any of user fields is defined
user[authorizationRoles]	request	*(optional)* Array of users roles.
user[enabled]	request	*(optional)* Flag set if user is enabled.

Example

```
$ curl http://demo.sylius.com/api/v1/customers/399 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "firstName": "John",
            "lastName": "Diggle",
            "email": "john.diggle@example.com",
            "gender": "m"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform full customer update without all required fields specified, you will receive a 400 error.

Example

```
$ curl http://demo.sylius.com/api/v1/customers/399 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "firstName": {},
            "lastName": {},
            "email": {
                "errors": [
                    "Please enter your email."
                ]
            },
            "birthday": {},
            "gender": {
                "errors": [
                    "Please choose your gender."
                ]
            },
            "phoneNumber": {},
            "subscribedToNewsletter": {},
            "group": {}
        }
    }
}
```

In order to perform a partial update, you should use a PATCH method.

Definition

```
PATCH /api/v1/customers/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested resource
email	request	*(optional)* **(unique)** Customers email
firstName	request	*(optional)* Customers first name
lastName	request	*(optional)* Customers last name
group	request	*(optional)* Customer group code
gender	request	*(optional)* Customers gender
birthday	request	*(optional)* Customers birthday
user[plainPassword]	request	*(optional)* Users plain password.
user[authorizationRoles]	request	*(optional)* Array of users roles.
user[enabled]	request	*(optional)* Flag set if user is enabled.

Example

```
$ curl http://demo.sylius.com/api/v1/customers/399 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '{"firstName": "Joe"}'
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Customer

Definition

```
DELETE /api/v1/customers/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested resource

Example

```
$ curl http://demo.sylius.com/api/v1/customers/399 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

Collection of all customer orders

To browse all orders for specific customer, you can do the following call:

Definition

```
GET /api/v1/customers/{id}/orders/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

Example

```
$ curl http://demo.sylius.com/api/v1/customers/7/orders/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK

{
    "page":1,
    "limit":10,
    "pages":1,
    "total":1,
    "_links":{
        "self":{
            "href":"\/api\/v1\/customers\/2\/orders\/?page=1&limit=10"
        },
        "first":{
            "href":"\/api\/v1\/customers\/2\/orders\/?page=1&limit=10"
        },
        "last":{
            "href":"\/api\/v1\/customers\/2\/orders\/?page=1&limit=10"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":2,
                "checkoutCompletedAt":"2017-02-23T14:53:11+0100",
```

```
            "number":"000000002",
            "items":[
                {
                    "id":4,
                    "quantity":2,
                    "unitPrice":101,
                    "total":123,
                    "units":[
                        {
                            "id":11,
                            "adjustments":[
                                {
                                    "id":12,
                                    "type":"order_promotion",
                                    "label":"Christmas",
                                    "amount":-40
                                }
                            ],
                            "adjustmentsTotal":-40
                        },
                        {
                            "id":12,
                            "adjustments":[
                                {
                                    "id":13,
                                    "type":"order_promotion",
                                    "label":"Christmas",
                                    "amount":-39
                                }
                            ],
                            "adjustmentsTotal":-39
                        }
                    ],
                    "unitsTotal":123,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0,
                    "variant":{
                        "id":181,
                        "code":"MEDIUM_MUG_CUP",
                        "optionValues":[
                            {
                                "code":"t_shirt_color_red",
                                "translations":{
                                    "en_US":{
                                        "locale":"en_US",
                                        "id":7,
                                        "value":"Red"
                                    }
                                }
                            },
                            {
                                "code":"t_shirt_size_s",
                                "translations":{
                                    "en_US":{
                                        "locale":"en_US",
```

```
                                        "id":10,
                                        "value":"S"
                                    }
                                }
                            }
                        ],
                        "position":0,
                        "translations":{
                            "en_US":{
                                "locale":"en_US",
                                "id":181,
                                "name":"tempore"
                            }
                        },
                        "onHold":0,
                        "onHand":6,
                        "tracked":false,
                        "channelPricings":{
                            "US_WEB": {
                                "channelCode": "US_WEB",
                                "price":101
                            }
                        },
                        "_links":{
                            "self":{
                                "href":"\/api\/v1\/products\/MUG\/variants\/
→MEDIUM_MUG_CUP"
                            },
                            "product":{
                                "href":"\/api\/v1\/products\/MUG"
                            }
                        }
                    },
                    "_links":{
                        "order":{
                            "href":"\/api\/v1\/orders\/2"
                        },
                        "product":{
                            "href":"\/api\/v1\/products\/MUG"
                        },
                        "variant":{
                            "href":"\/api\/v1\/products\/MUG\/variants\/MEDIUM_
→MUG_CUP"
                        }
                    }
                },
                {
                    "id":5,
                    "quantity":4,
                    "unitPrice":840,
                    "total":2050,
                    "units":[
                        {
                            "id":13,
                            "adjustments":[
                                {
                                    "id":14,
```

```
                    "type":"order_promotion",
                    "label":"Christmas",
                    "amount":-328
                }
            ],
            "adjustmentsTotal":-328
        },
        {
            "id":14,
            "adjustments":[
                {
                    "id":15,
                    "type":"order_promotion",
                    "label":"Christmas",
                    "amount":-328
                }
            ],
            "adjustmentsTotal":-328
        },
        {
            "id":15,
            "adjustments":[
                {
                    "id":16,
                    "type":"order_promotion",
                    "label":"Christmas",
                    "amount":-327
                }
            ],
            "adjustmentsTotal":-327
        },
        {
            "id":16,
            "adjustments":[
                {
                    "id":17,
                    "type":"order_promotion",
                    "label":"Christmas",
                    "amount":-327
                }
            ],
            "adjustmentsTotal":-327
        }
    ],
    "unitsTotal":2050,
    "adjustments":[

    ],
    "adjustmentsTotal":0,
    "variant":{
        "id":97,
        "code":"cd843634-6c85-3be0-9c84-7ce7786a394d-variant-0",
        "optionValues":[

        ],
        "position":0,
        "translations":{
```

```
                            "en_US":{
                                "locale":"en_US",
                                "id":97,
                                "name":"sequi"
                            }
                        },
                        "onHold":0,
                        "onHand":5,
                        "tracked":false,
                        "channelPricings":{
                            "US_WEB": {
                                "channelCode": "US_WEB",
                                "price":840
                            }
                        },
                        "_links":{
                            "self":{
                                "href":"\/api\/v1\/products\/cd843634-6c85-3be0-
→9c84-7ce7786a394d\/variants\/cd843634-6c85-3be0-9c84-7ce7786a394d-variant-0"
                            },
                            "product":{
                                "href":"\/api\/v1\/products\/cd843634-6c85-3be0-
→9c84-7ce7786a394d"
                            }
                        }
                    },
                    "_links":{
                        "order":{
                            "href":"\/api\/v1\/orders\/2"
                        },
                        "product":{
                            "href":"\/api\/v1\/products\/cd843634-6c85-3be0-9c84-
→7ce7786a394d"
                        },
                        "variant":{
                            "href":"\/api\/v1\/products\/cd843634-6c85-3be0-9c84-
→7ce7786a394d\/variants\/cd843634-6c85-3be0-9c84-7ce7786a394d-variant-0"
                        }
                    }
                },
                {
                    "id":6,
                    "quantity":4,
                    "unitPrice":660,
                    "total":1610,
                    "units":[
                        {
                            "id":17,
                            "adjustments":[
                                {
                                    "id":18,
                                    "type":"order_promotion",
                                    "label":"Christmas",
                                    "amount":-258
                                }
                            ],
                            "adjustmentsTotal":-258
```

```
            },
            {
                "id":18,
                "adjustments":[
                    {
                        "id":19,
                        "type":"order_promotion",
                        "label":"Christmas",
                        "amount":-258
                    }
                ],
                "adjustmentsTotal":-258
            },
            {
                "id":19,
                "adjustments":[
                    {
                        "id":20,
                        "type":"order_promotion",
                        "label":"Christmas",
                        "amount":-257
                    }
                ],
                "adjustmentsTotal":-257
            },
            {
                "id":20,
                "adjustments":[
                    {
                        "id":21,
                        "type":"order_promotion",
                        "label":"Christmas",
                        "amount":-257
                    }
                ],
                "adjustmentsTotal":-257
            }
        ],
        "unitsTotal":1610,
        "adjustments":[

        ],
        "adjustmentsTotal":0,
        "variant":{
            "id":45,
            "code":"c38fef5d-ddf9-31e2-8e05-71618605f381-variant-2",
            "optionValues":[
                {
                    "code":"mug_type_monster",
                    "translations":{
                        "en_US":{
                            "locale":"en_US",
                            "id":3,
                            "value":"Monster mug"
                        }
                    }
                }
```

```
                            ],
                            "position":2,
                            "translations":{
                                "en_US":{
                                    "locale":"en_US",
                                    "id":45,
                                    "name":"quod"
                                }
                            },
                            "onHold":0,
                            "onHand":7,
                            "tracked":false,
                            "channelPricings":{
                                "US_WEB": {
                                    "channelCode":"US_WEB"
                                    "price":660
                                }
                            },
                            "_links":{
                                "self":{
                                    "href":"\/api\/v1\/products\/c38fef5d-ddf9-31e2-
→8e05-71618605f381\/variants\/c38fef5d-ddf9-31e2-8e05-71618605f381-variant-2"
                                },
                                "product":{
                                    "href":"\/api\/v1\/products\/c38fef5d-ddf9-31e2-
→8e05-71618605f381"
                                }
                            }
                        },
                        "_links":{
                            "order":{
                                "href":"\/api\/v1\/orders\/2"
                            },
                            "product":{
                                "href":"\/api\/v1\/products\/c38fef5d-ddf9-31e2-8e05-
→71618605f381"
                            },
                            "variant":{
                                "href":"\/api\/v1\/products\/c38fef5d-ddf9-31e2-8e05-
→71618605f381\/variants\/c38fef5d-ddf9-31e2-8e05-71618605f381-variant-2"
                            }
                        }
                    },
                    {
                        "id":7,
                        "quantity":1,
                        "unitPrice":430,
                        "total":262,
                        "units":[
                            {
                                "id":21,
                                "adjustments":[
                                    {
                                        "id":22,
                                        "type":"order_promotion",
                                        "label":"Christmas",
                                        "amount":-168
```

```
                                }
                            ],
                            "adjustmentsTotal":-168
                        }
                    ],
                    "unitsTotal":262,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0,
                    "variant":{
                        "id":20,
                        "code":"4d4ba2e2-7138-3256-a88f-0caa5dc3bb81-variant-1",
                        "optionValues":[
                            {
                                "code":"mug_type_double",
                                "translations":{
                                    "en_US":{
                                        "locale":"en_US",
                                        "id":2,
                                        "value":"Double mug"
                                    }
                                }
                            }
                        ],
                        "position":1,
                        "translations":{
                            "en_US":{
                                "locale":"en_US",
                                "id":20,
                                "name":"nisi"
                            }
                        },
                        "onHold":0,
                        "onHand":2,
                        "tracked":false,
                        "channelPricings":{
                            "US_WEB": {
                                "channelCode":"US_WEB",
                                "price":430
                            }
                        },
                        "_links":{
                            "self":{
                                "href":"\/api\/v1\/products\/4d4ba2e2-7138-3256-
→a88f-0caa5dc3bb81\/variants\/4d4ba2e2-7138-3256-a88f-0caa5dc3bb81-variant-1"
                            },
                            "product":{
                                "href":"\/api\/v1\/products\/4d4ba2e2-7138-3256-
→a88f-0caa5dc3bb81"
                            }
                        }
                    },
                    "_links":{
                        "order":{
                            "href":"\/api\/v1\/orders\/2"
                        },
```

```
                              "product":{
                                  "href":"\/api\/v1\/products\/4d4ba2e2-7138-3256-a88f-
→0caa5dc3bb81"
                              },
                              "variant":{
                                  "href":"\/api\/v1\/products\/4d4ba2e2-7138-3256-a88f-
→0caa5dc3bb81\/variants\/4d4ba2e2-7138-3256-a88f-0caa5dc3bb81-variant-1"
                              }
                          }
                      },
                      {
                          "id":8,
                          "quantity":4,
                          "unitPrice":665,
                          "total":1623,
                          "units":[
                              {
                                  "id":22,
                                  "adjustments":[
                                      {
                                          "id":23,
                                          "type":"order_promotion",
                                          "label":"Christmas",
                                          "amount":-260
                                      }
                                  ],
                                  "adjustmentsTotal":-260
                              },
                              {
                                  "id":23,
                                  "adjustments":[
                                      {
                                          "id":24,
                                          "type":"order_promotion",
                                          "label":"Christmas",
                                          "amount":-259
                                      }
                                  ],
                                  "adjustmentsTotal":-259
                              },
                              {
                                  "id":24,
                                  "adjustments":[
                                      {
                                          "id":25,
                                          "type":"order_promotion",
                                          "label":"Christmas",
                                          "amount":-259
                                      }
                                  ],
                                  "adjustmentsTotal":-259
                              },
                              {
                                  "id":25,
                                  "adjustments":[
                                      {
                                          "id":26,
```

```
                                "type":"order_promotion",
                                "label":"Christmas",
                                "amount":-259
                            }
                        ],
                        "adjustmentsTotal":-259
                    }
                ],
                "unitsTotal":1623,
                "adjustments":[

                ],
                "adjustmentsTotal":0,
                "variant":{
                    "id":91,
                    "code":"6864f798-e0e5-339d-91c9-e6036befa414-variant-0",
                    "optionValues":[

                    ],
                    "position":0,
                    "translations":{
                        "en_US":{
                            "locale":"en_US",
                            "id":91,
                            "name":"maiores"
                        }
                    },
                    "onHold":0,
                    "onHand":7,
                    "tracked":false,
                    "channelPricings":{
                        "US_WEB": {
                            "channelCode":"US_WEB",
                            "price":665
                        }
                    },
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/products\/6864f798-e0e5-339d-
→91c9-e6036befa414\/variants\/6864f798-e0e5-339d-91c9-e6036befa414-variant-0"
                        },
                        "product":{
                            "href":"\/api\/v1\/products\/6864f798-e0e5-339d-
→91c9-e6036befa414"
                        }
                    }
                },
                "_links":{
                    "order":{
                        "href":"\/api\/v1\/orders\/2"
                    },
                    "product":{
                        "href":"\/api\/v1\/products\/6864f798-e0e5-339d-91c9-
→e6036befa414"
                    },
                    "variant":{
                        "href":"\/api\/v1\/products\/6864f798-e0e5-339d-91c9-
→e6036befa414\/variants\/6864f798-e0e5-339d-91c9-e6036befa414-variant-0"
```

```
                }
            }
        }
    ],
    "itemsTotal":5668,
    "adjustments":[
        {
            "id":27,
            "type":"shipping",
            "label":"FedEx",
            "amount":1530
        }
    ],
    "adjustmentsTotal":1530,
    "total":7198,
    "state":"new",
    "customer":{
        "id":2,
        "email":"metz.ted@beer.com",
        "emailCanonical":"metz.ted@beer.com",
        "firstName":"Dangelo",
        "lastName":"Graham",
        "gender":"u",
        "user":{
            "id":2,
            "username":"metz.ted@beer.com",
            "usernameCanonical":"metz.ted@beer.com",
            "roles":[
                "ROLE_USER"
            ],
            "enabled":true
        },
        "_links":{
            "self":{
                "href":"\/api\/v1\/customers\/2"
            }
        }
    },
    "channel":{
        "id":1,
        "code":"US_WEB",
        "name":"US Web Store",
        "hostname":"localhost",
        "color":"Plum",
        "createdAt":"2017-02-23T14:53:04+0100",
        "updatedAt":"2017-02-23T14:53:04+0100",
        "enabled":true,
        "taxCalculationStrategy":"order_items_based",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/US_WEB"
            }
        }
    },
    "shippingAddress":{
        "id":4,
        "firstName":"Kay",
```

```
                        "lastName":"Abbott",
                        "countryCode":"US",
                        "street":"Walsh Ford",
                        "city":"New Devante",
                        "postcode":"39325"
                },
                "billingAddress":{
                        "id":5,
                        "firstName":"Kay",
                        "lastName":"Abbott",
                        "countryCode":"US",
                        "street":"Walsh Ford",
                        "city":"New Devante",
                        "postcode":"39325"
                },
                "payments":[
                        {
                                "id":2,
                                "method":{
                                        "id":1,
                                        "code":"cash_on_delivery",
                                        "channels":[
                                                {
                                                        "id":1,
                                                        "code":"US_WEB",
                                                        "name":"US Web Store",
                                                        "hostname":"localhost",
                                                        "color":"Plum",
                                                        "createdAt":"2017-02-23T14:53:04+0100",
                                                        "updatedAt":"2017-02-23T14:53:04+0100",
                                                        "enabled":true,
                                                        "taxCalculationStrategy":"order_items_based",
                                                        "_links":{
                                                                "self":{
                                                                        "href":"\/api\/v1\/channels\/US_WEB"
                                                                }
                                                        }
                                                }
                                        ],
                                        "_links":{
                                                "self":{
                                                        "href":"\/api\/v1\/payment-methods\/cash_on_
→delivery"
                                                }
                                        }
                                },
                                "amount":7198,
                                "state":"new",
                                "_links":{
                                        "self":{
                                                "href":"\/api\/v1\/payments\/2"
                                        },
                                        "payment-method":{
                                                "href":"\/api\/v1\/payment-methods\/cash_on_delivery"
                                        },
                                        "order":{
                                                "href":"\/api\/v1\/orders\/2"
```

```
                }
            }
        }
    ],
    "shipments":[
        {
            "id":2,
            "state":"ready",
            "method":{
                "id":3,
                "code":"fedex",
                "enabled":true,
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/shipping-methods\/fedex"
                    },
                    "zone":{
                        "href":"\/api\/v1\/zones\/US"
                    }
                }
            },
            "_links":{
                "self":{
                    "href":"\/api\/v1\/shipments\/2"
                },
                "method":{
                    "href":"\/api\/v1\/shipping-methods\/fedex"
                },
                "order":{
                    "href":"\/api\/v1\/orders\/2"
                }
            }
        }
    ],
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"completed"
        }
    ]
    }
}
```

5.1.10 Exchange Rates API

These endpoints will allow you to easily manage exchange rates. Base URI is */api/v1/exchange-rates*.

Exchange Rate API response structure

If you request an exchange rate via API, you will receive an object with the following fields:

Field	Description
id	Id of the exchange rate
ratio	Exchange rate's ratio
sourceCurrency	*The currency object serialized with the default data*
targetCurrency	*The currency object serialized with the default data*
updatedAt	Last update date of the exchange rate

If you request for more detailed data, you will receive the default data with the additional field:

Field	Description
createdAt	Creation date date of the exchange rate

Creating an Exchange Rate

To create a new exchange rate you will need to call the `/api/v1/exchange-rates/` endpoint with the `POST` method.

Definition

```
POST /api/v1/exchange-rates/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
ratio	request	Ratio of the Exchange Rate
sourceCurrency	request	The source currency
targetCurrency	request	The target currency

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "ratio": "0,8515706",
            "sourceCurrency": "EUR",
            "targetCurrency": "GBP"
        }
    '
```

Tip: Remember that before you will be able to add a new exchange rate, both currencies have to be already defined.

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id":1,
    "ratio":0.85157,
    "sourceCurrency":{
        "id":2,
        "code":"EUR",
        "_links":{
            "self":{
                "href":"\/api\/v1\/currencies\/EUR"
            }
        }
    },
    "targetCurrency":{
        "id":3,
        "code":"GBP",
        "_links":{
            "self":{
                "href":"\/api\/v1\/currencies\/GBP"
            }
        }
    },
    "updatedAt":"2017-02-23T15:00:53+0100",
    "_links":{
        "self":{
            "href":"\/api\/v1\/exchange-rates\/EUR-GBP"
        }
    }
}
```

If you try to create an exchange rate without required fields you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code":400,
    "message":"Validation Failed",
    "errors":{
        "errors":[
            "The source and target currencies must differ."
```

```json
        ],
        "children":{
            "ratio":{
                "errors":[
                    "Please enter exchange rate ratio."
                ]
            },
            "sourceCurrency":{
                "errors":[
                    "This value is not valid."
                ]
            },
            "targetCurrency":{
                "errors":[
                    "This value is not valid."
                ]
            }
        }
    }
}
```

Getting a Single Exchange Rate

To retrieve the details of an exchange rate you will need to call the `/api/v1/exchange-rates/` `{firstCurrencyCode}-{secondCurrencyCode}` endpoint with the `GET` method.

Definition

```
GET /api/v1/exchange-rates/{firstCurrencyCode}-{secondCurrencyCode}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
firstCurrencyCode	url attribute	First currency code
secondCurrencyCode	url attribute	Second currency code

Example

To see the details of the exchange rate between `Euro (code = EUR)` and `British Pound (code = GBP)` use the below method:

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/EUR-GBP \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *EUR* and *GBP* codes are just an example.

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id":1,
    "ratio":0.85157,
    "sourceCurrency":{
        "id":2,
        "code":"EUR",
        "_links":{
            "self":{
                "href":"\/api\/v1\/currencies\/EUR"
            }
        }
    },
    "targetCurrency":{
        "id":3,
        "code":"GBP",
        "_links":{
            "self":{
                "href":"\/api\/v1\/currencies\/GBP"
            }
        }
    },
    "updatedAt":"2017-02-23T15:00:53+0100",
    "_links":{
        "self":{
            "href":"\/api\/v1\/exchange-rates\/EUR-GBP"
        }
    }
}
```

> **Warning:** The order of currencies in a request is not important. It doesn't matter if you will request the exchange rate for `EUR-GBP` or `GBP-EUR` the response will always be the same (including source and target currencies).

Collection of Currencies

To retrieve a paginated list of exchange rates you will need to call the `/api/v1/exchange-rates/` endpoint with the `GET` method.

Definition

```
GET /api/v1/exchange-rates/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all exchange rates use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page":1,
    "limit":10,
    "pages":1,
    "total":1,
    "_links":{
        "self":{
            "href":"\/api\/v1\/exchange-rates\/?page=1&limit=10"
        },
        "first":{
            "href":"\/api\/v1\/exchange-rates\/?page=1&limit=10"
        },
        "last":{
            "href":"\/api\/v1\/exchange-rates\/?page=1&limit=10"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":1,
                "ratio":0.85157,
                "sourceCurrency":{
                    "id":2,
                    "code":"EUR",
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/currencies\/EUR"
                        }
                    }
                },
                "targetCurrency":{
                    "id":3,
                    "code":"GBP",
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/currencies\/GBP"
                        }
                    }
                },
                "updatedAt":"2017-02-23T15:00:53+0100",
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/exchange-rates\/EUR-GBP"
                    }
                }
            }
```

```
            }
        ]
    }
}
```

Updating an Exchange Rate

To update an exchange rate you will need to call the `/api/v1/exchange-rates/`
`firstCurrencyCode-secondCurrencyCode` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/exchange-rates/{firstCurrencyCode}-{secondCurrencyCode}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
firstCurrencyCode	url attribute	First currency code
secondCurrencyCode	url attribute	Second currency code
ratio	request	Ratio of the Exchange Rate

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/EUR-GBP \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "ratio": "0,9515706"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to update an exchange rate without the required fields you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/EUR-GBP \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```json
{
    "code":400,
    "message":"Validation Failed",
    "errors":{
        "children":{
            "ratio":{
                "errors":[
                    "Please enter exchange rate ratio."
                ]
            },
            "sourceCurrency":{

            },
            "targetCurrency":{

            }
        }
    }
}
```

Deleting an Exchange Rate

To delete an exchange rate you will need to call the `/api/v1/exchange-rates/`
`firstCurrencyCode-secondCurrencyCode` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/exchange-rates/{firstCurrencyCode}-{secondCurrencyCode}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
firstCurrencyCode	url attribute	First currency code
secondCurrencyCode	url attribute	Second currency code

Example

```
$ curl http://demo.sylius.com/api/v1/exchange-rates/EUR-GBP \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.11 Locales API

These endpoints will allow you to easily manage locales. Base URI is */api/v1/locales*.

Locale API response structure

If you request a locale via API, you will receive an object with the following fields:

Field	Description
id	Id of the locale
code	Unique locale identifier

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the locale
code	Unique locale identifier
updatedAt	Last update date of the locale
createdAt	Creation date of the locale

Note: Read more about *Locales in the component docs*.

Creating a Locale

To create a new locale you will need to call the `/api/v1/locales/` endpoint with the `POST` method.

Definition

```
POST /api/v1/locales/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Locale identifier

Example

```
$ curl http://demo.sylius.com/api/v1/locales/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
```

(continues on next page)

```
        {
            "code": "pl"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 4,
    "code": "pl",
    "createdAt": "2017-02-14T12:49:38+0100",
    "updatedAt": "2017-02-14T12:49:39+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/locales\/pl"
        }
    }
}
```

If you try to create a locale without code you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/locales/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "code": {
                "errors": [
                    "Please enter locale code."
                ]
            }
        }
    }
}
```

Getting a Single Locale

To retrieve the details of a locale you will need to call the `/api/v1/locales/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/locales/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested locale

Example

To see the details of the locale with `code = pl` use the below method:

```
$ curl http://demo.sylius.com/api/v1/locales/pl \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *pl* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 4,
    "code": "pl",
    "createdAt": "2017-02-14T12:49:38+0100",
    "updatedAt": "2017-02-14T12:49:39+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/locales\/pl"
        }
    }
}
```

Collection of Locales

To retrieve a paginated list of locales you will need to call the `/api/v1/locales/` endpoint with the `GET` method.

Definition

```
GET /api/v1/locales/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all locales use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/locales/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 3,
    "_links": {
        "self": {
            "href": "\/api\/v1\/locales\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/locales\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/locales\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 2,
                "code": "en_US",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/locales\/en_US"
                    }
                }
            },
            {
                "id": 3,
                "code": "af",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/locales\/af"
                    }
                }
            }
```

```
        },
        {
            "id": 4,
            "code": "pl",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/locales\/pl"
                }
            }
        }
    ]
    }
}
```

Deleting a Locale

To delete a locale you will need to call the `/api/v1/locales/code` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/locales/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the removed locale

Example

```
$ curl http://demo.sylius.com/api/v1/locales/pl \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.12 Orders API

Sylius orders API endpoint is */api/v1/orders*.

If you request an order via API, you will receive an object with the following fields:

Field	Description
id	Id of the order
items	List of items related to the order
itemsTotal	Sum of all items prices
adjustments	List of adjustments related to the order
adjustmentsTotal	Sum of all order adjustments
total	Sum of items total and adjustments total
customer	*Customer detailed serialization* for order
channel	*Default channel serialization*
currencyCode	Currency of the order
checkoutState	*State of the checkout process*
state	*State of the order*
checkoutCompletedAt	Date when the checkout has been completed
number	Serial number of the order
shippingAddress	Detailed address serialization
billingAddress	Detailed address serialization
shipments	Detailed serialization of all related shipments
payments	Detailed serialization of all related payments

Orders endpoint gives an access point to finalized carts, so to the orders that have been placed. At this moment only certain actions are allowed:

Action	Description
Show	Presenting of the order
Cancelling	Cancelling of the order
Shipping	Shipping of the order
Completing the payment	Complete the order's payment

Show Action

You can request detailed order information by executing the following request:

Definition

```
GET /api/v1/orders/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested order

Example

```
$ curl http://demo.sylius.com/api/v1/orders/21 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The value *21* was taken from previous responses, where we managed the cart and proceed the checkout. Your value can be different. If you need more information about Cart API please, check *this article*.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id":21,
    "checkoutCompletedAt":"2017-02-15T13:31:33+0100",
    "number":"000000021",
    "items":[
        {
            "id":74,
            "quantity":1,
            "unitPrice":100000,
            "total":100000,
            "units":[
                {
                    "id":228,
                    "adjustments":[

                    ],
                    "adjustmentsTotal":0,
                    "_links":{
                        "order":{
                            "href":"\/api\/v1\/orders\/21"
                        }
                    }
                }
            ],
            "unitsTotal":100000,
            "adjustments":[

            ],
            "adjustmentsTotal":0,
            "variant":{
                "id":331,
                "code":"MEDIUM_MUG_CUP",
                "optionValues":[
                    {
                        "name":"Mug type",
                        "code":"mug_type_medium"
                    }
                ],
                "position":2,
                "translations":{
                    "en_US":{
                        "locale":"en_US",
                        "id":331,
                        "name":"Medium Mug"
                    }
                },
                "version": 1,
```

```json
                    "onHold":0,
                    "onHand":10,
                    "tracked":false,
                    "channelPricings":{
                        "US_WEB":{
                            "channelCode":"US_WEB",
                            "price":100000
                        }
                    },
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/products\/5\/variants\/331"
                        },
                        "product":{
                            "href":"\/api\/v1\/products\/5"
                        }
                    }
                },
                "_links":{
                    "product":{
                        "href":"\/api\/v1\/products\/5"
                    },
                    "variant":{
                        "href":"\/api\/v1\/products\/5\/variants\/331"
                    }
                }
            }
        ],
        "itemsTotal":100000,
        "adjustments":[
            {
                "id":252,
                "type":"shipping",
                "label":"DHL Express",
                "amount":3549
            }
        ],
        "adjustmentsTotal":3549,
        "total":103549,
        "state":"new",
        "customer":{
            "id":1,
            "email":"shop@example.com",
            "emailCanonical":"shop@example.com",
            "firstName":"John",
            "lastName":"Doe",
            "gender":"u",
            "user":{
                "id":1,
                "username":"shop@example.com",
                "usernameCanonical":"shop@example.com",
                "roles":[
                    "ROLE_USER"
                ],
                "enabled":true
            },
            "_links":{
```

```
            "self":{
                "href":"\/api\/v1\/customers\/1"
            }
        }
    },
    "channel":{
        "id":1,
        "code":"US_WEB",
        "name":"US Web Store",
        "hostname":"localhost",
        "color":"MediumPurple",
        "createdAt":"2017-02-14T11:10:02+0100",
        "updatedAt":"2017-02-14T11:10:02+0100",
        "enabled":true,
        "taxCalculationStrategy":"order_items_based",
        "_links":{
            "self":{
                "href":"\/api\/v1\/channels\/1"
            }
        }
    },
    "shippingAddress":{
        "id":71,
        "firstName":"Frederick D.",
        "lastName":"Gregory",
        "countryCode":"US",
        "street":"300 E St SW",
        "city":"\u2019Washington",
        "postcode":"DC 20546",
        "createdAt":"2017-02-14T11:55:40+0100",
        "updatedAt":"2017-02-14T17:00:17+0100"
    },
    "billingAddress":{
        "id":72,
        "firstName":"Frederick D.",
        "lastName":"Gregory",
        "countryCode":"US",
        "street":"300 E St SW",
        "city":"\u2019Washington",
        "postcode":"DC 20546",
        "createdAt":"2017-02-14T11:55:40+0100",
        "updatedAt":"2017-02-14T17:00:17+0100"
    },
    "payments":[
        {
            "id":21,
            "method":{
                "id":2,
                "code":"bank_transfer",
                "createdAt":"2017-02-14T11:10:02+0100",
                "updatedAt":"2017-02-14T11:10:02+0100",
                "channels":[
                    {
                        "id":1,
                        "code":"US_WEB",
                        "name":"US Web Store",
                        "hostname":"localhost",
```

```json
                    "color":"MediumPurple",
                    "createdAt":"2017-02-14T11:10:02+0100",
                    "updatedAt":"2017-02-14T11:10:02+0100",
                    "enabled":true,
                    "taxCalculationStrategy":"order_items_based",
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/channels\/1"
                        }
                    }
                }
            ],
            "_links":{
                "self":{
                    "href":"\/api\/v1\/payment-methods\/bank_transfer"
                }
            }
        },
        "amount":103549,
        "state":"new",
        "createdAt":"2017-02-14T11:53:41+0100",
        "updatedAt":"2017-02-15T13:31:33+0100",
        "_links":{
            "self":{
                "href":"\/api\/v1\/payments\/21"
            },
            "payment-method":{
                "href":"\/api\/v1\/payment-methods\/bank_transfer"
            },
            "order":{
                "href":"\/api\/v1\/orders\/21"
            }
        }
    }
],
"shipments":[
    {
        "id":21,
        "state":"ready",
        "method":{
            "id":2,
            "code":"dhl_express",
            "category_requirement":1,
            "calculator":"flat_rate",
            "configuration":{
                "US_WEB":{
                    "amount":3549
                }
            },
            "createdAt":"2017-02-14T11:10:02+0100",
            "updatedAt":"2017-02-14T11:10:02+0100",
            "enabled":true,
            "_links":{
                "self":{
                    "href":"\/api\/v1\/shipping-methods\/dhl_express"
                },
                "zone":{
```

```
                        "href":"\/api\/v1\/zones\/US"
                    }
                }
            },
            "createdAt":"2017-02-14T11:53:41+0100",
            "updatedAt":"2017-02-15T13:31:33+0100",
            "_links":{
                "self":{
                    "href":"\/api\/v1\/shipments\/21"
                },
                "method":{
                    "href":"\/api\/v1\/shipping-methods\/dhl_express"
                },
                "order":{
                    "href":"\/api\/v1\/orders\/21"
                }
            }
        }
    ],
    "currencyCode":"USD",
    "localeCode":"en_US",
    "checkoutState":"completed"
}
```

Cancel Action

You can cancel an already placed order by executing the following request:

Definition

```
PUT /api/v1/orders/{id}/cancel
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested order

Example

```
$ curl http://demo.sylius.com/api/v1/orders/21/cancel \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 204 NO CONTENT
```

Ship Action

You can ship an already placed order by executing the following request:

Definition

```
PUT /api/v1/orders/{orderId}/shipments/{id}/ship
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
orderId	url attribute	Id of the requested order
id	url attribute	Id of the shipped shipment
tracking	request	*(optional)* The tracking code of the shipment

Example

```
$ curl http://demo.sylius.com/api/v1/orders/21/shipments/21/ship \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 204 No Content
```

Note: It is important to emphasise that in this example the shipment id is the same value as for the order, but it is a coincidence rather than a rule.

Complete The Payment Action

You can complete the payment of an already placed order by executing the following request:

Definition

```
PUT /api/v1/orders/{orderId}/payments/{id}/complete
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
orderId	url attribute	Id of the requested order
id	url attribute	Id of payment to complete

Example

```
$ curl http://demo.sylius.com/api/v1/orders/21/payments/21/complete \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 204 No content
```

5.1.13 Payment Methods API

These endpoints will allow you to easily manage payment methods. Base URI is */api/v1/payment-methods*.

Payment Method API response structure

If you request a payment method via API, you will receive an object with the following fields:

Field	Description
id	Unique id of the payment method
code	Unique code of the payment method
name	The payment method's name
createdAt	Date of creation
updatedAt	Date of the last update

Note: Read more about *Payment Methods in the component docs*.

Getting a Single Payment Method

To retrieve the details of a payment method you will need to call the `/api/v1/payment-methods/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/payment-methods/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested payment method

Example

To see the details of the payment method with `code` = `cash_on_delivery` use the below method:

```
$ curl http://demo.sylius.com/api/v1/payment-methods/cash_on_delivery \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *cash_on_delivery* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id": 1,
    "code": "cash_on_delivery",
    "position": 0,
    "createdAt": "2017-02-24T16:14:03+0100",
    "updatedAt": "2017-02-24T16:14:03+0100",
    "enabled": true,
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 1,
            "name": "Cash on delivery",
            "description": "Rerum expedita sit aut praesentium soluta sint aperiam."
        }
    },
    "channels": [
        {
            "id": 1,
            "code": "US_WEB",
            "name": "US Web Store",
            "hostname": "localhost",
            "color": "SlateBlue",
            "createdAt": "2017-02-24T16:14:03+0100",
            "updatedAt": "2017-02-24T16:14:03+0100",
            "enabled": true,
            "taxCalculationStrategy": "order_items_based",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/channels\/US_WEB"
                }
            }
        }
    ],
    "_links": {
        "self": {
            "href": "\/api\/v1\/payment-methods\/cash_on_delivery"
        }
    }
}
```

5.1.14 Payments API

These endpoints will allow you to easily present payments. Base URI is */api/v1/payments*.

Payment API response structure

If you request a payment via API, you will receive an object with the following fields:

Field	Description
id	Unique id of the payment
method	*The payment method object serialized* for the cart
amount	The amount of payment
state	*State of the payment process*
_links[self]	Link to itself
_links[payment-method]	Link to the related payment method
_links[order]	Link to the related order

Note: Read more about *Payments in the component docs*.

Getting a Single Payment

To retrieve the details of a payment you will need to call the `/api/v1/payments/{id}` endpoint with the `GET` method.

Definition

```
GET /api/v1/payments/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested payment

Example

To see the details of the payment with `id = 20` use the below method:

```
$ curl http://demo.sylius.com/api/v1/payments/20 \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *20* id is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id":20,
    "method":{
        "id":2,
        "code":"bank_transfer",
        "channels":[
            {
                "id":1,
                "code":"US_WEB",
                "name":"US Web Store",
                "hostname":"localhost",
                "color":"DeepSkyBlue",
                "createdAt":"2017-02-26T11:31:19+0100",
                "updatedAt":"2017-02-26T11:31:19+0100",
                "enabled":true,
                "taxCalculationStrategy":"order_items_based",
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/channels\/US_WEB"
                    }
                }
            }
        ],
        "_links":{
            "self":{
                "href":"\/api\/v1\/payment-methods\/bank_transfer"
            }
        }
    },
    "amount":4507,
    "state":"new",
    "_links":{
        "self":{
            "href":"\/api\/v1\/payments\/20"
        },
        "payment-method":{
            "href":"\/api\/v1\/payment-methods\/bank_transfer"
        },
        "order":{
            "href":"\/api\/v1\/orders\/20"
        }
    }
}
```

Collection of Payments

To retrieve a paginated list of payments you will need to call the `/api/v1/payments/` endpoint with the `GET` method.

Definition

```
GET /api/v1/payments/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting[amount]	query	*(optional)* Sorting direction on the `amount` field (`DESC`/`ASC`)
sorting[createdAt]	query	*(optional)* Sorting direction on the `createdAt` field (`ASC` by default)
sorting[updatedAt]	query	*(optional)* Sorting direction on the `updatedAt` field (`DESC`/`ASC`)

Example

To see the first page of the paginated list of payments with two payments on each page use the below snippet:

```
$ curl http://demo.sylius.com/api/v1/payments/\?limit\=2 \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page":1,
    "limit":2,
    "pages":10,
    "total":20,
    "_links":{
        "self":{
            "href":"\/api\/v1\/payments\/?page=1&limit=2"
        },
        "first":{
            "href":"\/api\/v1\/payments\/?page=1&limit=2"
        },
        "last":{
            "href":"\/api\/v1\/payments\/?page=10&limit=2"
        },
        "next":{
            "href":"\/api\/v1\/payments\/?page=2&limit=2"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":1,
                "method":{
                    "id":2,
                    "code":"bank_transfer",
                    "_links":{
```

```json
                    "self":{
                        "href":"\/api\/v1\/payment-methods\/bank_transfer"
                    }
                }
            },
            "amount":3812,
            "state":"new",
            "_links":{
                "self":{
                    "href":"\/api\/v1\/payments\/1"
                },
                "payment-method":{
                    "href":"\/api\/v1\/payment-methods\/bank_transfer"
                },
                "order":{
                    "href":"\/api\/v1\/orders\/1"
                }
            }
        },
        {
            "id":2,
            "method":{
                "id":2,
                "code":"bank_transfer",
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/payment-methods\/bank_transfer"
                    }
                }
            },
            "amount":3915,
            "state":"new",
            "_links":{
                "self":{
                    "href":"\/api\/v1\/payments\/2"
                },
                "payment-method":{
                    "href":"\/api\/v1\/payment-methods\/bank_transfer"
                },
                "order":{
                    "href":"\/api\/v1\/orders\/2"
                }
            }
        }
    ]
  }
}
```

5.1.15 Product Attributes API

These endpoints will allow you to easily manage product attributes. Base URI is */api/v1/product-attributes*.

Product Attribute API response structure

If you request a product attribute via API, you will receive an object with the following fields:

Field	Description
id	Id of the product attribute
code	Unique product attribute identifier
position	The position of the product attribute among other product attributes
type	Type of the product attribute (for example text)
translations	Collection of translations (each contains name in given language)

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the product attribute
code	Unique product attribute identifier
position	The position of the product attribute among other product attributes
type	Type of the product attribute (for example text)
translations	Collection of translations (each contains name in given language)
updatedAt	Last update date of the product attribute
createdAt	Creation date of the product attribute

Note: Read more about *Product Attributes in the component docs*.

Creating a Product Attribute

To create a new product attribute you will need to call the `/api/v1/products-attributes/{type}` endpoint with the `POST` method.

Definition

```
POST /api/v1/product-attributes/{type}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
type	url attribute	Type of the product attribute (for example text)
code	request	**(unique)** Product attribute identifier

Example

To create a new text product attribute use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-attributes/text \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "mug_material"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```json
{
    "id": 1,
    "code": "mug_material",
    "type": "text",
    "configuration": [],
    "position": 0,
    "translations": {},
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-attributes\/mug_material"
        }
    }
}
```

> **Warning:** If you try to create a product attribute without code you will receive a `400 Bad Request` error, that will contain validation errors.

```
$ curl http://demo.sylius.com/api/v1/product-attributes/text \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
```

Exemplary Response

```
STATUS: 400 BAD REQUEST
```

```json
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "type": {},
            "position": {},
            "translations": {},
            "code": {
                "errors": [
                    "Please enter attribute code."
                ]
            },
            "configuration": {
                "children": {
                    "min": {},
                    "max": {}
                }
            }
        }
    }
}
```

You can also create a product attribute with additional (not required) fields:

Parameter	Parameter type	Description
position	request	Position within sorted product attribute list of the new product attribute
translations['localeCode']['name']	request	Name of the product attribute

Some of product attributes have also their own (optional) configuration:

Product attribute type	configuration construction	Description
text	configuration['min'] configuration['max']	Both field must be defined together. They described minimal and maximal length of the text attribute.
select	configuration['multiple'] configuration['min'] configuration['max'] configuration['choices']	The *multiple*, *min*, and *max* must be defined together. They allow to select several values, limited by minimal and maximal amount of entries. The *choices* is an array of available options in the product attribute.

Note: You can also see exemplary request about creating configured select product attribute here.

Example

```
$ curl http://demo.sylius.com/api/v1/product-attributes/text/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "mug_material",
            "translations": {
                "de_CH": {
                    "name": "Becher Material"
                },
                "en_US": {
                    "name": "Mug material"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 1,
    "code": "mug_material",
    "type": "text",
    "configuration": [],
    "position": 0,
    "createdAt": "2017-02-24T16:14:05+0100",
    "updatedAt": "2017-02-24T16:14:05+0100",
    "translations": {
        "de_CH": {
            "id": 1,
            "locale": "de_CH",
            "name": "Becher Material"
        },
        "en_US": {
            "id": 2,
            "locale": "en_US",
            "name": "Mug material"
        }
    },
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-attributes\/mug_material"
        }
    }
}
```

Getting a Single Product Attribute

To retrieve the details of a product attribute you will need to call the `/api/v1/product-attributes/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/product-attributes/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested product attribute

Example

To see the details of the product attribute with `code = sticker_paper` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-attributes/sticker_paper \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *sticker_paper* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 2,
    "code": "sticker_paper",
    "type": "text",
    "configuration": [],
    "position": 1,
    "createdAt": "2017-03-29T10:05:00+0200",
    "updatedAt": "2017-03-31T09:48:37+0200",
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 2,
            "name": "Sticker paper"
        }
    },
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-attributes\/sticker_paper"
        }
    }
}
```

Collection of Product Attributes

To retrieve a paginated list of product attributes you will need to call the `/api/v1/product-attributes/` endpoint with the `GET` method.

Definition

```
GET /api/v1/product-attributes/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all product attributes use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/product-attributes/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 10,
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-attributes\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/product-attributes\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/product-attributes\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "mug_material",
                "type": "select",
                "position": 0,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "name": "Mug material"
                    }
                },
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/product-attributes\/mug_material"
                    }
                }
            },
            {
                "id": 2,
                "code": "sticker_paper",
                "type": "text",
                "position": 1,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 2,
                        "name": "Sticker paper"
                    }
                },
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/product-attributes\/sticker_paper"
                    }
                }
            }
```

```
        },
        {
            "id": 3,
            "code": "sticker_resolution",
            "type": "text",
            "position": 2,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 3,
                    "name": "Sticker resolution"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/sticker_resolution"
                }
            }
        },
        {
            "id": 4,
            "code": "book_author",
            "type": "text",
            "position": 3,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 4,
                    "name": "Book author"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/book_author"
                }
            }
        },
        {
            "id": 5,
            "code": "book_isbn",
            "type": "text",
            "position": 4,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 5,
                    "name": "Book ISBN"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/book_isbn"
                }
            }
        },
        {
            "id": 6,
```

```
            "code": "book_pages",
            "type": "integer",
            "position": 5,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 6,
                    "name": "Book pages"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/book_pages"
                }
            }
        },
        {
            "id": 7,
            "code": "book_genre",
            "type": "select",
            "position": 6,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 7,
                    "name": "Book genre"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/book_genre"
                }
            }
        },
        {
            "id": 8,
            "code": "t_shirt_brand",
            "type": "text",
            "position": 7,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 8,
                    "name": "T-Shirt brand"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/t_shirt_brand"
                }
            }
        },
        {
            "id": 9,
            "code": "t_shirt_collection",
            "type": "text",
            "position": 8,
```

```
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 9,
                    "name": "T-Shirt collection"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/t_shirt_collection"
                }
            }
        },
        {
            "id": 10,
            "code": "t_shirt_material",
            "type": "text",
            "position": 9,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 10,
                    "name": "T-Shirt material"
                }
            },
            "_links": {
                "self": {
                    "href": "\/api\/v1\/product-attributes\/t_shirt_material"
                }
            }
        }
    ]
  }
}
```

Updating a Product Attribute

To fully update a product attribute you will need to call the `/api/v1/product-attributes/code` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/product-attributes/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product attribute identifier

Example

To fully update the product attribute with `code = mug_material` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-attributes/mug_material \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Mug material"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

To update a product attribute partially you will need to call the `/api/v1/product-attributes/code` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/product-attributes/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product attribute identifier

Example

To partially update the product attribute with `code = mug_material` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-attributes/mug_material \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Mug material"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Product Attribute

To delete a product attribute you will need to call the `/api/v1/product-attributes/code` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/product-attributes/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product attribute identifier

Example

To delete the product attribute with `code = mug_material` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-attributes/mug_material \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.16 Product Options API

These endpoints will allow you to easily manage product options. Base URI is */api/v1/product-options*.

Product Option API response structure

If you request a product option via API, you will receive an object with the following fields:

Field	Description
id	Id of the product option
code	Unique product option identifier
position	The position of the product option among other product options

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the product option
code	Unique product option identifier
position	The position of the product option among other product options
translations	Collection of translations (each contains name in given language)
values	Names of options in which the product can occur

Note: Read more about *Product Options in the component docs*.

Creating a Product Option

To create a new product option you will need to call the `/api/v1/products-options/` endpoint with the `POST` method.

Definition

```
POST /api/v1/product-options/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Product option identifier
values	request	At least two option values

Example

To create a new product option use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-options/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "MUG_SIZE",
            "values": [
                {
                    "code": "MUG_SIZE_S",
                    "translations": {
                        "en_US": {
                            "value": "Small"
                        }
                    }
                },
                {
                    "code": "MUG_SIZE_L",
                    "translations": {
                        "en_US": {
                            "value": "Large"
```

```
                }
            }
        }
    ]
}
'
```

Exemplary Response

```
STATUS: 201 CREATED
```

```json
{
    "id": 1,
    "code": "MUG_SIZE",
    "position": 0,
    "translations": {},
    "values": [
        {
            "code": "MUG_SIZE_S",
            "translations": {
                "en_US": {
                    "id": 1,
                    "locale": "en_US",
                    "value": "Small"
                }
            }
        },
        {
            "code": "MUG_SIZE_L",
            "translations": {
                "en_US": {
                    "id": 2,
                    "locale": "en_US",
                    "value": "Large"
                }
            }
        }
    ],
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-options\/MUG_SIZE"
        }
    }
}
```

Warning: If you try to create a product option without all necessary fields you will receive a `400 Bad Request` error, that will contain validation errors.

```
$ curl http://demo.sylius.com/api/v1/product-options/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
```

Exemplary Response

```
STATUS: 400 BAD REQUEST
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "errors": [
            "Please add at least 2 option values."
        ],
        "children": {
            "position": {},
            "translations": {},
            "values": {},
            "code": {
                "errors": [
                    "Please enter option code."
                ]
            }
        }
    }
}
```

You can also create a product option with additional (not required) fields:

Parameter	Parameter type	Description
position	request	Position within sorted product option list of the new product option
translations['localeCode']['name']	request	Name of the product option
values	request	Collection of option values

Each product option value has the following fields:

Parameter	Parameter type	Description
code	request	(**unique**) Product option value identifier
translations['localeCode']['value']	request	Translation of the value

Example

```
$ curl http://demo.sylius.com/api/v1/product-options/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "MUG_SIZE",
            "translations": {
                "de_CH": {
                    "name": "Bechergröße"
```

(continues on next page)

```
            },
            "en_US": {
                "name": "Mug size"
            }
        },
        "values": [
            {
                "code": "MUG_SIZE_S",
                "translations": {
                    "de_CH": {
                        "value": "Klein"
                    },
                    "en_US": {
                        "value": "Small"
                    }
                }
            },
            {
                "code": "MUG_SIZE_L",
                "translations": {
                    "de_CH": {
                        "value": "Groß"
                    },
                    "en_US": {
                        "value": "Large"
                    }
                }
            }
        ]
    }
'
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 1,
    "code": "MUG_SIZE",
    "position": 0,
    "translations": {
        "en_US": {
            "id": 1,
            "locale": "en_US",
            "name": "Mug size"
        },
        "de_CH": {
            "id": 2,
            "locale": "de_CH",
            "name": "Bechergröße"
        }
    },
    "values": [
        {
```

```json
                "code": "MUG_SIZE_S",
                "translations": {
                    "en_US": {
                        "id": 1,
                        "locale": "en_US",
                        "value": "Small"
                    },
                    "de_CH": {
                        "id": 2,
                        "locale": "de_CH",
                        "value": "Klein"
                    }
                }
            },
            {
                "code": "MUG_SIZE_L",
                "translations": {
                    "de_CH": {
                        "id": 3,
                        "locale": "de_CH",
                        "value": "Groß"
                    },
                    "en_US": {
                        "id": 4,
                        "locale": "en_US",
                        "value": "Large"
                    }
                }
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/products\/MUG_SIZE"
            }
        }
    }
}
```

Getting a Single Product Option

To retrieve the details of a product option you will need to call the `/api/v1/product-options/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/product-options/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested the product option

Example

To see the details of the product option with `code = MUG_TYPE` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-options/MUG_TYPE \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *mug_type* is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id": 1,
    "code": "MUG_TYPE",
    "position": 0,
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 1,
            "value": "Mug type"
        }
    },
    "values": [
        {
            "code": "mug_type_medium",
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 1,
                    "value": "Medium mug"
                }
            }
        },
        {
            "code": "mug_type_double",
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 2,
                    "value": "Double mug"
                }
            }
        },
        {
            "code": "mug_type_monster",
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 3,
                    "value": "Monster mug"
```

(continues on next page)

```
                    }
                }
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/products\/MUG_TYPE"
            }
        }
    }
}
```

Collection of Product Options

To retrieve a paginated list of product options you will need to call the `/api/v1/product-options/` endpoint with the `GET` method.

Definition

```
GET /api/v1/product-options/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all product options use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/product-options/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 1,
    "total": 4,
    "_links": {
        "self": {
            "href": "\/api\/v1\/product-options\/?sorting%5Bcode%5D=desc&page=1&
→limit=4"
        },
```

```
        "first": {
            "href": "\/api\/v1\/product-options\/?sorting%5Bcode%5D=desc&page=1&
→limit=4"
        },
        "last": {
            "href": "\/api\/v1\/product-options\/?sorting%5Bcode%5D=desc&page=1&
→limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "mug_type",
                "position": 0,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "value": "Mug type"
                    }
                },
                "values": [
                    {
                        "code": "mug_type_medium",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 1,
                                "value": "Medium mug"
                            }
                        }
                    },
                    {
                        "code": "mug_type_double",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 2,
                                "value": "Double mug"
                            }
                        }
                    },
                    {
                        "code": "mug_type_monster",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 3,
                                "value": "Monster mug"
                            }
                        }
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/mug_type"
```

```
                }
            }
        },
        {
            "id": 2,
            "code": "sticker_size",
            "position": 1,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 2,
                    "value": "Sticker size"
                }
            },
            "values": [
                {
                    "code": "sticker_size-3",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 4,
                            "value": "3\""
                        }
                    }
                },
                {
                    "code": "sticker_size_5",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 5,
                            "value": "5\""
                        }
                    }
                },
                {
                    "code": "sticker_size_7",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 6,
                            "value": "7\""
                        }
                    }
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/sticker_size"
                }
            }
        },
        {
            "id": 3,
            "code": "t_shirt_color",
            "position": 2,
            "translations": {
```

```json
                "en_US": {
                    "locale": "en_US",
                    "id": 3,
                    "value": "T-Shirt color"
                }
            },
            "values": [
                {
                    "code": "t_shirt_color_red",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 7,
                            "value": "Red"
                        }
                    }
                },
                {
                    "code": "t_shirt_color_black",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 8,
                            "value": "Black"
                        }
                    }
                },
                {
                    "code": "t_shirt_color_white",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 9,
                            "value": "White"
                        }
                    }
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/t_shirt_color"
                }
            }
        },
        {
            "id": 4,
            "code": "t_shirt_size",
            "position": 3,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 4,
                    "value": "T-Shirt size"
                }
            },
            "values": [
                {
```

```
                "code": "t_shirt_size_s",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 10,
                        "value": "S"
                    }
                }
            },
            {
                "code": "t_shirt_size_m",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 11,
                        "value": "M"
                    }
                }
            },
            {
                "code": "t_shirt_size_l",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 12,
                        "value": "L"
                    }
                }
            },
            {
                "code": "t_shirt_size_xl",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 13,
                        "value": "XL"
                    }
                }
            },
            {
                "code": "t_shirt_size_xxl",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 14,
                        "value": "XXL"
                    }
                }
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/products\/t_shirt_size"
            }
        }
    }
]
```

```
      }
}
```

Updating a Product Option

To fully update a product option you will need to call the `/api/v1/product-options/code` endpoint with the
`PUT` method.

Definition

```
PUT /api/v1/product-options/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product option identifier

Example

To fully update the product option with `code = MUG_SIZE` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-options/MUG_SIZE \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Mug size"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

To update a product option partially you will need to call the `/api/v1/product-options/code` endpoint with
the `PATCH` method.

Definition

```
PATCH /api/v1/product-options/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product option identifier

Example

To partially update the product option with `code = MUG_SIZE` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-options/MUG_SIZE \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Mug size"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Product Option

To delete a product option you will need to call the `/api/v1/product-options/code` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/product-options/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product option identifier

Example

To delete the product option with `code = MUG_SIZE` use the below method:

```
$ curl http://demo.sylius.com/api/v1/product-options/MUG_SIZE \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.17 Product Reviews API

These endpoints will allow you to easily manage product reviews. Base URI is
/api/v1/products/{productCode}/reviews/.

Product Reviews API response structure

When you get a collection of resources, you will receive objects with the following fields:

Field	Description
id	Id of product review
title	Title of product review
comment	Comment of product review
author	Customer author for product review (This is customer that added the product review; this will contain customer resource information)
status	Status of product review (New, Accepted, Rejected)
review-Subject	This is the review subject for the product review. For this case of the product review, this will contain a product resource

Note: Read more about *ProductReviews docs*.

Creating a Product Review

To create a new product review you will need to call the `/api/v1/products/{productCode}/reviews/`
endpoint with the `POST` method.

Definition

```
POST /api/v1/products/{productCode}/reviews/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
productCode	url attribute	Code of product for which the reviews should be created
title	request	Product review title
comment	request	Product review comment
rating	request	Product review rating (1..5)
author	request	Product review author

Example

To create a new product review for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "title": "A product review",
            "rating": "3",
            "comment": "This is a comment review",
            "author": {
                "email": "test@example.com"
            }
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id": 4,
    "title": "A product review",
    "rating": 3,
    "comment": "This is a comment review",
    "author": {
        "id": 2,
        "email": "test@example.com",
        "emailCanonical": "test@example.com",
        "gender": "u",
        "_links": {
            "self": {
                "href": "/api/v1/customers/2"
            }
        }
    },
    "status": "new",
    "reviewSubject": {
        "id": 1,
        "name": "MUG-TH",
        "code": "MUG-TH",
        "attributes": [],
        "options": [],
        "associations": [],
        "translations": []
    }
}
```

> **Warning:** If you try to create a resource without title, rating, comment or author, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```json
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "rating": {
                "errors": [
                    "You must check review rating."
                ],
                "children": [
                    {},
                    {},
                    {},
                    {},
                    {}
                ]
            },
            "title": {
                "errors": [
                    "Review title should not be blank."
                ]
            },
            "comment": {
                "errors": [
                    "Review comment should not be blank."
                ]
            },
            "author": {
                "children": {
                    "email": {
                        "errors": [
                            "Please enter your email."
                        ]
                    }
                }
            }
        }
    }
}
```

Getting a Single Product Review

To retrieve the details of a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}` endpoint with the `GET` method.

Definition

```
GET /api/v1/products/{productCode}/reviews/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Identifier of the product review
productCode	url attribute	Code of product for which the reviews should be displayed

Example

To see the details of the product review with `id = 1`, which is defined for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "title": "A product review",
    "rating": 3,
    "comment": "This is a comment review",
    "author": {
        "id": 2,
        "email": "test@example.com",
        "emailCanonical": "test@example.com",
        "gender": "u",
        "_links": {
            "self": {
                "href": "/api/v1/customers/2"
            }
        }
    },
    "status": "new",
    "reviewSubject": {
        "id": 1,
        "name": "MUG-TH",
        "code": "MUG-TH",
        "attributes": [],
        "options": [],
```

```
        "associations": [],
        "translations": []
    }
}
```

Collection of Product Reviews

To retrieve a paginated list of reviews for a selected product you will need to call the `/api/v1/products/{productCode}/reviews/` endpoint with the `GET` method.

Definition

```
GET /api/v1/products/{productCode}/reviews/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
productCode	url attribute	Code of product for which the reviews should be displayed
limit	query	*(optional)* Number of items to display per page, by default = 10
sort-ing['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

Example

To see the first page of all product reviews for the product with `code = MUG-TH` use the method below.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 3,
    "_links": {
        "self": {
            "href": "/api/v1/products/MUG-TH/reviews/?page=1&limit=10"
        },
        "first": {
            "href": "/api/v1/products/MUG-TH/reviews/?page=1&limit=10"
        },
        "last": {
```

```
                "href": "/api/v1/products/MUG-TH/reviews/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 4,
                "title": "A product review",
                "rating": 3,
                "comment": "This is a comment review",
                "author": {
                    "id": 2,
                    "email": "test@example.com",
                    "_links": {
                        "self": {
                            "href": "/api/v1/customers/2"
                        }
                    }
                },
                "status": "new",
                "reviewSubject": {
                    "id": 1,
                    "name": "MUG-TH",
                    "code": "MUG-TH",
                    "options": [],
                    "averageRating": 0,
                    "images": [],
                    "_links": {
                        "self": {
                            "href": "/api/v1/products/MUG-TH"
                        }
                    }
                },
                "createdAt": "2017-10-04T20:19:06+03:00",
                "updatedAt": "2017-10-04T20:19:06+03:00"
            },
            {
                "id": 3,
                "title": "A product review 2",
                "rating": 5,
                "comment": "This is a comment review 2",
                "author": {
                    "id": 1,
                    "email": "onetest@example.com",
                    "_links": {
                        "self": {
                            "href": "/api/v1/customers/1"
                        }
                    }
                },
                "status": "new",
                "reviewSubject": {
                    "id": 1,
                    "name": "MUG-TH",
                    "code": "MUG-TH",
                    "options": [],
                    "averageRating": 0,
```

```
                "images": [],
                "_links": {
                    "self": {
                        "href": "/api/v1/products/MUG-TH"
                    }
                }
            },
            "createdAt": "2017-10-04T18:23:56+03:00",
            "updatedAt": "2017-10-04T18:44:08+03:00"
        },
        {
            "id": 1,
            "title": "Test review 3",
            "rating": 4,
            "comment": "This is a comment review 3",
            "author": {
                "id": 1,
                "email": "onetest@example.com",
                "_links": {
                    "self": {
                        "href": "/api/v1/customers/1"
                    }
                }
            },
            "status": "accepted",
            "reviewSubject": {
                "id": 1,
                "name": "MUG-TH",
                "code": "MUG-TH",
                "options": [],
                "averageRating": 0,
                "images": [],
                "_links": {
                    "self": {
                        "href": "/api/v1/products/MUG-TH"
                    }
                }
            },
            "createdAt": "2017-10-03T23:53:24+03:00",
            "updatedAt": "2017-10-04T19:18:00+03:00"
        }
    ]
  }
}
```

Updating Product Review

To fully update a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/products/{productCode}/reviews/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Product review id
productCode	url attribute	Code of product for which the reviews should be updated
title	request	Product review title
comment	request	Product review comment
rating	request	Product review rating (1..5)

Example

To fully update the product review with `id = 1` for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
          "title": "A product review",
          "rating": "4",
          "comment": "This is a comment for review"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

To partially update a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/products/{productCode}/reviews/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Identifier of the product review
productCode	url attribute	Code of product for which the reviews should be updated
title	request	Product review title

Example

To partially update the product review with `id = 1` for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1 \
    -H "Authorization: Bearer SampleToken" \
```

```
-H "Content-Type: application/json" \
-X PATCH \
--data '
    {
        "title": "This is an another title for the review"
    }
'
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Product Review

To delete a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}`
endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/products/{productCode}/reviews/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Identifier of the product review
productCode	url attribute	Code of product for which the reviews should be deleted

Example

To delete the product review with `id = 1` from the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

Accept a Product Review

To accept a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}/`
`accept` endpoint with the `POST`, `PUT` or `PATCH` method.

Definition

```
POST /api/v1/products/{productCode}/reviews/{id}/accept
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Identifier of the product review
productCode	url attribute	Code of product for which the reviews should be accepted

Example

To accept the product review with `id = 1` from the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1/accept \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 204 No Content
```

Reject a Product Review

To reject a product review you will need to call the `/api/v1/products/{productCode}/reviews/{id}/reject` endpoint with the `POST`, `PUT` or `PATCH` method.

Definition

```
POST /api/v1/products/{productCode}/reviews/{id}/reject
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Identifier of the product review
productCode	url attribute	Code of product for which the reviews should be rejected

Example

To reject the product review with `id = 1` from the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.org/api/v1/products/MUG-TH/reviews/1/reject \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.18 Product Variants API

These endpoints will allow you to easily manage product variants. Base URI is */api/v1/products/{productCode}/variants/.*

Product Variant API response structure

When you get a collection of resources, "Default" serialization group will be used and the following fields will be exposed:

Field	Description
id	Id of product variant
code	Unique product variant's identifier
position	Position of variant in product (each product can have many variants and they can be ordered by position)
optionValues	Collection of options in which product is available (for example: small, medium and large mug)
translations	Collection of translations (each contains name in given language)
tracked	The information if the variant is tracked by inventory
channelPricings	Collection of prices defined for all enabled channels
taxCategory	Tax category to which variant is assigned
shippingCate-gory	Shipping category to which variant is assigned
version	Version of the product variant

If you request more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of product variant
code	Unique product variant's identifier
position	Position of variant in product (each product can have many variant and they can be ordered by position)
tracked	The information if the variant is tracked by inventory
channelPricings	Collection of prices defined for all enabled channels
taxCategory	Tax category to which variant is assigned
shippingCate-gory	Shipping category to which variant is assigned
version	Version of the product variant
optionValues	Collection of options in which product is available (for example: small, medium and large mug)
translations	Collection of translations (each contains name in given language)
onHold	Information about how many product are currently reserved by customer
onHand	Information about the number of product in given variant currently available in shop
width	The physical width of variant
height	The physical height of variant
depth	The physical depth of variant
weight	The physical weight of variant

Note: Read more about *ProductVariant model in the component docs*.

Creating a Product Variant

To create a new product variant you will need to call the `/api/v1/products/productCode/variants/` endpoint with the `POST` method.

Definition

```
POST /api/v1/products/{productCode}/variants/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
productCode	url attribute	Id of product for which the variants should be displayed
code	request	**(unique)** Product variant identifier

Example

To create new product variant for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "medium-theme-mug"
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id": 331,
    "code": "medium-theme-mug",
    "optionValues": [],
    "position": 0,
    "translations": [],
    "version": 1,
    "onHold": 0,
    "onHand": 0,
    "tracked": false,
    "channelPricings": [],
    "_links": {
        "self": {
```

(continues on next page)

```
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/medium-theme-mug"
        },
        "product": {
            "href": "\/api\/v1\/products\/MUG_TH"
        }
    }
}
```

> **Warning:** If you try to create a resource without code, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code":400,
    "message":"Validation Failed",
    "errors": {
        "children": {
            "enabled":{},
            "translations":{},
            "attributes":{},
            "associations":{},
            "channels":{},
            "mainTaxon":{},
            "productTaxons":{},
            "images":{},
            "code":{
                "errors":["Please enter product code."]
            },
            "options":{}
        }
    }
}
```

You can also create a product variant with additional (not required) fields:

Parameter	Parameter type	Description
transla-tions['localeCode']['name']	request	Name of the product variant
position	request	Position of variant in product
tracked	request	The information if the variant is tracked by inventory (true or false)
channelPricings	request	Collection of objects which contains prices for all enabled channels
taxCategory	request	Code of object which provides information about tax category to which variant is assigned
shippingCategory	request	Code of object which provides information about shipping category to which variant is assigned
optionValues	request	Object with information about ProductOption (by code) and ProductOptionValue (by code)
onHand	request	Information about the number of product in given variant currently available in shop
width	request	The width of variant
height	request	The height of variant
depth	request	The depth of variant
weight	request	The weight of variant

Warning: Channels must be created and enabled before the prices will be defined for they.

Example

Here is an example of creating a product variant with additional data for the product with `code` = `MUG-TH`.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "double-theme-mug",
            "translations": {
                    "en_US": {
                        "name": "Double Theme Mug"
                }
            },
            "channelPricings": {
                "US_WEB": {
                    "price": "1243"
                }
            },
            "tracked": true,
            "onHand": 5,
            "taxCategory": "other",
            "shippingCategory": "default",
            "optionValues": {
                "mug_type": "mug_type_double"
            },
            "width": 5,
```

```
            "height": 10,
            "depth": 15,
            "weight": 20
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```json
{
    "id": 332,
    "code": "double-theme-mug",
    "optionValues": [
        {
            "name": "Mug type",
            "code": "mug_type_double"
        }
    ],
    "position": 1,
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 332,
            "name": "Double Theme Mug"
        }
    },
    "version": 1,
    "onHold": 0,
    "onHand": 5,
    "tracked": true,
    "weight": 20,
    "width": 5,
    "height": 10,
    "depth": 15,
    "taxCategory": {
        "id": 3,
        "code": "other",
        "name": "Other",
        "description": "Error est aut libero et. Recusandae rerum rem enim qui␣
↪sapiente ea sed. Provident et aspernatur molestias et et.",
        "createdAt": "2017-02-27T09:12:17+0100",
        "updatedAt": "2017-02-27T09:12:17+0100",
        "_links": {
            "self": {
                "href": "\/api\/v1\/tax-categories\/other"
            }
        }
    },
    "shippingCategory": {
        "id": 1,
        "code": "default",
        "name": "Default shipping category",
        "createdAt": "2017-02-27T10:48:14+0100",
```

```
            "updatedAt": "2017-02-27T10:48:15+0100",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/shipping-categories\/default"
                }
            }
        }
    "channelPricings": {
        "US_WEB": {
            "channelCode": "US_WEB",
            "price": 124300
        }
    },
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/double-theme-mug"
        },
        "product": {
            "href": "\/api\/v1\/products\/MUG_TH"
        }
    }
}
```

Getting a Single Product Variant

To retrieve the details of a product variant you will need to call the `/api/v1/products/productCode/variants/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/products/{productCode}/variants/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Identifier of the product variant
productCode	url attribute	Id of product for which the variants should be displayed

Example

To see the details of the product variant with `code = medium-theme-mug`, which is defined for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/medium-theme-mug \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 331,
    "code": "medium-mug-theme",
    "optionValues": [],
    "position": 0,
    "translations": [],
    "version": 1,
    "onHold": 0,
    "onHand": 0,
    "tracked": false,
    "channelPricings": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/medium-mug-theme"
        },
        "product": {
            "href": "\/api\/v1\/products\/MUG_TH"
        }
    }
}
```

Collection of Product Variants

To retrieve a paginated list of variants for a selected product you will need to call the `/api/v1/products/productCode/variants/` endpoint with the `GET` method.

Definition

```
GET /api/v1/products/{productCode}/variants/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
productCode	url attribute	Code of product for which the variants should be displayed
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

Example

To see the first page of all product variants for the product with `code = MUG-TH` use the method below.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 2,
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 331,
                "code": "medium-mug-theme",
                "optionValues": [],
                "position": 0,
                "translations": [],
                "version": 1,
                "tracked": false,
                "channelPricings": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/MUG_TH\/variants\/medium-mug-
↪theme"
                    }
                }
            },
            {
                "id": 332,
                "code": "double-theme-mug",
                "optionValues": [
                    {
                        "name": "Mug type",
                        "code": "mug_type_double"
                    }
                ],
                "position": 1,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 332,
                        "name": "Double Theme Mug"
                    }
                },
                "version": 1,
                "tracked": true,
```

```json
                "taxCategory": {
                    "id": 3,
                    "code": "other",
                    "name": "Other",
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/tax-categories\/other"
                        }
                    }
                },
                "shippingCategory": {
                    "id": 1,
                    "code": "default",
                    "name": "Default shipping category",
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/shipping-categories\/default"
                        }
                    }
                },
                "tracked": false,
                "channelPricings": {
                    "US_WEB": {
                        "channelCode": "US_WEB",
                        "price": 1200
                    }
                },
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/MUG_TH\/variants\/double-theme-
→mug"
                    }
                }
            }
        ]
    }
}
```

Updating Product Variant

To fully update a product variant you will need to call the `/api/v1/products/productCode/variants/`
`code` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/products/{productCode}/variants/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Identifier of the product variant
productCode	url attribute	Id of product for which the variants should be displayed
translations['localeCode']['name']	request	*(optional)* Name of the product variant
position	request	*(optional)* Position of the variant in product
tracked	request	*(optional)* The information if the variant is tracked by inventory (true or false)
channelPricings	request	*(optional)* Collection of prices for all the enabled channels
taxCategory	request	*(optional)* Code of object which provides information about tax category to which the variant is assigned
shippingCategory	request	*(optional)* Code of object which provides information about shipping category to which the variant is assigned
optionValues	request	*(optional)* Object with information about ProductOption (by code) and ProductOptionValue (by code)
onHand	request	*(optional)* Information about the number of product in the given variant currently available in shop
width	request	*(optional)* The width of the variant
height	request	*(optional)* The height of the variant
depth	request	*(optional)* The depth of the variant
weight	request	*(optional)* The weight of the variant

Example

To fully update the product variant with `code = double-theme-mug` for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/double-theme-mug \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "translations":{
                "en_US": {
                    "name": "Monster mug"
                }
            },
            "version": 1,
            "channelPricings": {
                "US_WEB": {
                    "price": 54
                }
            },
            "tracked": true,
            "onHand": 3,
            "taxCategory": "other",
            "shippingCategory": "default",
            "width": 5,
```

```
            "height": 10,
            "depth": 15,
            "weight": 20,
            "optionValues": {
                "mug_type" :"mug_type_monster"
            }
        }
    '
```

> **Warning:** Do not forget to pass version of the variant. Without this you will receive a `409 Conflict` error.

Exemplary Response

```
STATUS: 204 No Content
```

To partially update a product variant you will need to call the `/api/v1/products/productCode/variants/code` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/products/{productCode}/variants/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Identifier of the product variant
productCode	url attribute	Id of product for which the variants should be displayed
translations['localeCode']['name']	request	Name of product variant

Example

To partially update the product variant with `code = double-theme-mug` for the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/double-theme-mug \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "translations": {
                "pl": {
                    "name": "Gigantyczny kubek"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Product Variant

To delete a product variant you will need to call the `/api/v1/products/productCode/variants/code` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/products/{productCode}/variants/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Identifier of the product variant
productCode	url attribute	Id of product for which the variants should be displayed

Example

To delete the product variant with `code = double-theme-mug` from the product with `code = MUG-TH` use the below method.

```
$ curl http://demo.sylius.com/api/v1/products/MUG-TH/variants/double-theme-mug \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.19 Products API

These endpoints will allow you to easily manage products. Base URI is *api/v1/products*.

Product API response structure

If you request a product via API, you will receive an object with the following fields:

Field	Description
id	Id of the product
code	Unique product identifier (for example SKU)
averageRating	Average from accepted ratings given by customer
channels	Collection of channels to which the product was assigned
translations	Collection of translations (each contains slug and name in given language)
options	Options assigned to the product
images	Images assigned to the product

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the product
code	Unique product identifier
averageRating	Average from ratings given by customer
channels	Collection of channels to which the product was assigned
translations	Collection of translations (each contains slug and name in given language)
attributes	Collection of attributes connected with the product (for example material)
associations	Collection of products associated with the created product (for example accessories to this product)
variants	Collection of variants connected with the product
reviews	Collection of reviews passed by customers
productTaxons	Collection of relations between product and taxons
mainTaxon	The main taxon to whose the product is assigned

Note: Read more about *Product model in the component docs*.

Creating a Product

To create a new product you will need to call the `/api/v1/products/` endpoint with the `POST` method.

Definition

```
POST /api/v1/products/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Product identifier

Example

To create a new product use the below method:

```
$ curl http://demo.sylius.com/api/v1/products/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "TS3"
        }
'
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 61,
    "code": "TS3",
    "attributes": [],
    "options": [],
    "associations": [],
    "productTaxons": [],
    "channels": [],
    "reviews": [],
    "averageRating": 0,
    "images": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/TS3"
        },
        "variants": {
            "href": "\/api\/v1\/products\/TS3\/variants\/"
        }
    }
}
```

> **Warning:** If you try to create a product without name, code or slug, you will receive a `400 Bad Request`
> error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/products/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
   -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "enabled": {},
            "translations": {},
            "attributes": {},
            "associations": {},
            "channels": {},
            "mainTaxon": {},
            "productTaxons": {},
            "images": {},
            "code": {
                "errors": [
                    "Please enter product code."
                ]
            },
            "options": {}
        }
    }
}
```

You can also create a product with additional (not required) fields:

Parameter	Parameter type	Description
channels	request	Collection of channels codes, which we want to associate with created product
translations['localeCode']['name']	request	Name of the product
translations['localeCode']['slug']	request	**(unique)** Slug for the product
options	request	Collection of options codes, which we want to associate with created product
images	request	Collection of images types, which we want to associate with created product
attributes	request	Array of attributes (each object has information about selected attribute's code, its value and locale in which it was defined)
associations	request	Object with code of productAssociationType and string in which the codes of associated products was written down.
productTaxons	request	String in which the codes of taxons was written down (separated by comma)
mainTaxon	request	The main taxon's code to whose product is assigned

Example

```
$ curl http://demo.sylius.com/api/v1/products/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "MUG_TH",
            "mainTaxon": "mugs",
```

```
            "productTaxons": "mugs",
            "channels": [
                "US_WEB"
            ],
            "attributes": [
                {
                    "attribute": "mug_color",
                    "localeCode": "en_US",
                    "value": "yellow"
                }
            ],
            "options": [
                "mug_type"
            ],
            "associations": {
                "similar_products": "SMM,BMM"
            },
            "translations": {
                "en_US": {
                    "name": "Theme Mug",
                    "slug": "theme-mug"
                },
                "pl": {
                    "name": "Kubek z motywem",
                    "slug": "kubek-z-motywem"
                }
            },
            "images": [
                {
                    "type": "ford"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "name": "Theme Mug",
    "id": 69,
    "code": "MUG_TH",
    "attributes": [
        {
            "code": "mug_material",
            "name": "Mug material",
            "value": "concrete",
            "type": "text",
            "id": 155
        }
    ],
    "options": [
        {
```

```
            "id": 1,
            "code": "mug_type",
            "position": 0,
            "values": [
                {
                    "name": "Mug type",
                    "code": "mug_type_medium"
                },
                {
                    "name": "Mug type",
                    "code": "mug_type_double"
                },
                {
                    "name": "Mug type",
                    "code": "mug_type_monster"
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/mug_type"
                }
            }
        }
    ],
    "associations": [
        {
            "id": 13,
            "type": {
                "name": "Similar products",
                "id": 1,
                "code": "similar_products",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "name": "Similar products"
                    }
                }
            },
            "associatedProducts": [
                {
                    "name": "Batman mug",
                    "id": 63,
                    "code": "BMM",
                    "attributes": [],
                    "options": [],
                    "associations": [],
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 63,
                            "name": "Batman mug",
                            "slug": "batman-mug"
                        }
                    },
                    "productTaxons": [],
                    "channels": [],
```

```
                        "reviews": [],
                        "averageRating": 0,
                        "images": [],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/products\/BMM"
                            },
                            "variants": {
                                "href": "\/api\/v1\/products\/BMM\/variants\/"
                            }
                        }
                    },
                    {
                        "name": "Spider-Man Mug",
                        "id": 68,
                        "code": "SMM",
                        "attributes": [],
                        "options": [],
                        "associations": [],
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 70,
                                "name": "Spider-Man Mug",
                                "slug": "spider-man-mug"
                            }
                        },
                        "productTaxons": [],
                        "channels": [],
                        "reviews": [],
                        "averageRating": 0,
                        "images": [],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/products\/SMM"
                            },
                            "variants": {
                                "href": "\/api\/v1\/products\/SMM\/variants\/"
                            }
                        }
                    }
                ]
            }
    ],
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 71,
            "name": "Theme Mug",
            "slug": "theme-mug"
        },
        "pl": {
            "locale": "pl",
            "id": 72,
            "name": "Kubek z motywem",
            "slug": "kubek-z-motywem"
        }
```

```
    },
    "productTaxons": [
        {
            "id": 78,
            "taxon": {
                "name": "Mugs",
                "id": 2,
                "code": "mugs",
                "root": {
                    "name": "Category",
                    "id": 1,
                    "code": "category",
                    "children": {
                        "1": {
                            "name": "T-Shirts",
                            "id": 5,
                            "code": "t_shirts",
                            "children": [],
                            "left": 4,
                            "right": 5,
                            "level": 1,
                            "position": 1,
                            "translations": [],
                            "images": [],
                            "_links": {
                                "self": {
                                    "href": "\/api\/v1\/taxons\/t_shirts"
                                }
                            }
                        }
                    },
                    "left": 1,
                    "right": 6,
                    "level": 0,
                    "position": 0,
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 1,
                            "name": "Category",
                            "slug": "category",
                            "description": "Cupiditate ut esse perspiciatis.␣
→Aspernatur nihil ducimus maxime doloremque. Ut aut ad unde necessitatibus␣
→voluptatibus id in."
                        }
                    },
                    "images": [],
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/taxons\/category"
                        }
                    }
                },
                "parent": {
                    "name": "Category",
                    "id": 1,
                    "code": "category",
```

```
                "children": {
                    "1": {
                        "name": "T-Shirts",
                        "id": 5,
                        "code": "t_shirts",
                        "children": [],
                        "left": 4,
                        "right": 5,
                        "level": 1,
                        "position": 1,
                        "translations": [],
                        "images": [],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/taxons\/t_shirts"
                            }
                        }
                    }
                },
                "left": 1,
                "right": 6,
                "level": 0,
                "position": 0,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "name": "Category",
                        "slug": "category",
                        "description": "Cupiditate ut esse perspiciatis.
→Aspernatur nihil ducimus maxime doloremque. Ut aut ad unde necessitatibus
→voluptatibus id in."
                    }
                },
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/category"
                    }
                }
            },
            "children": [],
            "left": 2,
            "right": 3,
            "level": 1,
            "position": 0,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 2,
                    "name": "Mugs",
                    "slug": "mugs",
                    "description": "Non omnis vel impedit eaque necessitatibus et
→eveniet. Fugiat distinctio quos aut commodi ea minima. Et natus ratione sit aperiam
→a molestiae. Eligendi sed cumque deleniti unde magnam."
                }
            },
```

```
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/mugs"
                }
            }
        },
        "position": 0
    }
],
"channels": [
    {
        "id": 1,
        "code": "US_WEB",
        "name": "US Web Store",
        "hostname": "localhost",
        "color": "DarkSeaGreen",
        "createdAt": "2017-02-27T09:12:16+0100",
        "updatedAt": "2017-02-27T09:12:16+0100",
        "enabled": true,
        "taxCalculationStrategy": "order_items_based",
        "_links": {
            "self": {
                "href": "\/api\/v1\/channels\/US_WEB"
            }
        }
    }
],
"mainTaxon": {
    "name": "Mugs",
    "id": 2,
    "code": "mugs",
    "root": {
        "name": "Category",
        "id": 1,
        "code": "category",
        "children": {
            "1": {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "children": [],
                "left": 4,
                "right": 5,
                "level": 1,
                "position": 1,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/t_shirts"
                    }
                }
            }
        },
        "left": 1,
        "right": 6,
```

```
                "level": 0,
                "position": 0,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "name": "Category",
                        "slug": "category",
                        "description": "Cupiditate ut esse perspiciatis. Aspernatur nihil␣
→ducimus maxime doloremque. Ut aut ad unde necessitatibus voluptatibus id in."
                    }
                },
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/category"
                    }
                }
            },
            "parent": {
                "name": "Category",
                "id": 1,
                "code": "category",
                "children": {
                    "1": {
                        "name": "T-Shirts",
                        "id": 5,
                        "code": "t_shirts",
                        "children": [],
                        "left": 4,
                        "right": 5,
                        "level": 1,
                        "position": 1,
                        "translations": [],
                        "images": [],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/taxons\/t_shirts"
                            }
                        }
                    }
                },
                "left": 1,
                "right": 6,
                "level": 0,
                "position": 0,
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 1,
                        "name": "Category",
                        "slug": "category",
                        "description": "Cupiditate ut esse perspiciatis. Aspernatur nihil␣
→ducimus maxime doloremque. Ut aut ad unde necessitatibus voluptatibus id in."
                    }
                },
                "images": [],
```

```
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/category"
                }
            }
        },
        "children": [],
        "left": 2,
        "right": 3,
        "level": 1,
        "position": 0,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 2,
                "name": "Mugs",
                "slug": "mugs",
                "description": "Non omnis vel impedit eaque necessitatibus et eveniet.
→ Fugiat distinctio quos aut commodi ea minima. Et natus ratione sit aperiam a␣
→molestiae. Eligendi sed cumque deleniti unde magnam."
            }
        },
        "images": [],
        "_links": {
            "self": {
                "href": "\/api\/v1\/taxons\/mugs"
            }
        }
    },
    "reviews": [],
    "averageRating": 0,
    "images": [
        {
            "id": 121,
            "type": "ford",
            "path": "65\/f6\/1e3b25f3721768b535e5c37ac005.jpeg"
        }
    ],
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/MUG_TH"
        },
        "variants": {
            "href": "\/api\/v1\/products\/MUG_TH\/variants\/"
        }
    }
}
```

Note: The images (files) should be passed in an array as an attribute of request. See how it is done in Sylius here.

Getting a Single Product

To retrieve the details of a product you will need to call the `/api/v1/product/code` endpoint with the `GET` method.

Definition

```
GET /api/v1/products/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product identifier

Example

To see the details for the product with `code` = `BMM` use the below method:

```
$ curl http://demo.sylius.com/api/v1/products/BMM \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *BMM* code is an exemplary value. Your value can be different. Check in the list of all products if you are not sure which code should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "name": "Batman mug",
    "id": 63,
    "code": "BMM",
    "attributes": [],
    "options": [],
    "associations": [],
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 63,
            "name": "Batman mug",
            "slug": "batman-mug"
        }
    },
    "productTaxons": [],
    "channels": [],
    "reviews": [],
    "averageRating": 0,
    "images": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/BMM"
        },
        "variants": {
            "href": "\/api\/v1\/products\/BMM\/variants\/"
        }
```

(continues on next page)

```
    }
}
```

Collection of Products

To retrieve a paginated list of products you will need to call the `/api/v1/products/` endpoint with the `GET`
method.

Definition

```
GET /api/v1/products/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

To see the first page of all products use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/products/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 16,
    "total": 63,
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
        "first": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
        "last": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=16&limit=4"
        },
        "next": {
```

```
                "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=2&limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "name": "Spiderman Mug",
                "id": 61,
                "code": "SMM",
                "options": [],
                "averageRating": 0,
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/SMM"
                    }
                }
            },
            {
                "name": "Theme Mug",
                "id": 63,
                "code": "MUG_TH",
                "options": [
                    {
                        "id": 1,
                        "code": "mug_type",
                        "position": 0,
                        "values": [
                            {
                                "code": "mug_type_medium",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 1,
                                        "value": "Medium mug"
                                    }
                                }
                            },
                            {
                                "code": "mug_type_double",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 2,
                                        "value": "Double mug"
                                    }
                                }
                            },
                            {
                                "code": "mug_type_monster",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 3,
                                        "value": "Monster mug"
                                    }
                                }
                            }
```

```
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/mug_type"
                }
            }
        }
    ],
    "averageRating": 0,
    "images": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/MUG_TH"
        }
    }
},
{
    "name": "Sticker \"quis\"",
    "id": 16,
    "code": "fe06f44e-2169-328f-8cd2-cd5495b4b6ad",
    "options": [
        {
            "id": 2,
            "code": "sticker_size",
            "position": 1,
            "values": [
                {
                    "code": "sticker_size-3",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 4,
                            "value": "3\""
                        }
                    }
                },
                {
                    "code": "sticker_size_5",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 5,
                            "value": "5\""
                        }
                    }
                },
                {
                    "code": "sticker_size_7",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 6,
                            "value": "7\""
                        }
                    }
                }
```

```
                    ],
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/products\/sticker_size"
                        }
                    }
                }
            ],
            "averageRating": 0,
            "images": [
                {
                    "id": 31,
                    "type": "main"
                },
                {
                    "id": 32,
                    "type": "thumbnail"
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/fe06f44e-2169-328f-8cd2-
→cd5495b4b6ad"
                }
            }
        },
        {
            "name": "T-Shirt \"vel\"",
            "id": 51,
            "code": "f6858e9c-2f48-3d59-9f54-e7ac9898c0bd",
            "options": [
                {
                    "id": 3,
                    "code": "t_shirt_color",
                    "position": 2,
                    "values": [
                        {
                            "code": "t_shirt_color_red",
                            "translations": {
                                "en_US": {
                                    "locale": "en_US",
                                    "id": 7,
                                    "value": "Red"
                                }
                            }
                        },
                        {
                            "code": "t_shirt_color_black",
                            "translations": {
                                "en_US": {
                                    "locale": "en_US",
                                    "id": 8,
                                    "value": "Black"
                                }
                            }
                        },
                        {
```

```
                    "code": "t_shirt_color_white",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 9,
                            "value": "White"
                        }
                    }
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/t_shirt_color"
                }
            }
        }
    },
    {
        "id": 4,
        "code": "t_shirt_size",
        "position": 3,
        "values": [
            {
                "code": "t_shirt_size_s",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 10,
                        "value": "S"
                    }
                }
            },
            {
                "code": "t_shirt_size_m",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 11,
                        "value": "M"
                    }
                }
            },
            {
                "code": "t_shirt_size_l",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 12,
                        "value": "L"
                    }
                }
            },
            {
                "code": "t_shirt_size_xl",
                "translations": {
                    "en_US": {
                        "locale": "en_US",
                        "id": 13,
```

```json
                                "value": "XL"
                            }
                        }
                    },
                    {
                        "code": "t_shirt_size_xxl",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 14,
                                "value": "XXL"
                            }
                        }
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/t_shirt_size"
                    }
                }
            }
        ],
        "averageRating": 0,
        "images": [
            {
                "id": 101,
                "type": "main"
            },
            {
                "id": 102,
                "type": "thumbnail"
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/products\/f6858e9c-2f48-3d59-9f54-
→e7ac9898c0bd"
            }
        }
    }
]
    }
}
```

Updating a Product

To fully update a product you will need to call the `/api/v1/products/code` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/products/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product identifier
translations['localeCode']['name']	request	Name of the product
translations['localeCode']['slug']	request	**(unique)** Slug

Example

To fully update the product with `code = BMM` use the below method:

```
$ curl http://demo.sylius.com/api/v1/products/BMM \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Batman mug",
                    "slug": "batman-mug"
                }
            }
        }
'
```

Exemplary Response

```
STATUS: 204 No Content
```

To update a product partially you will need to call the `/api/v1/products/code` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/products/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product identifier

Example

To partially update the product with `code = BMM` use the below method:

```
$ curl http://demo.sylius.com/api/v1/products/BMM \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
```

```
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Batman mug"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Product

To delete a product you will need to call the `/api/v1/products/code` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/products/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique product identifier

Example

To delete the product with `code = MUG_TH` use the below method:

```
$ curl http://demo.sylius.com/api/v1/products/MUG_TH \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.20 Promotion Coupons API

These endpoints will allow you to easily manage promotion coupons. Base URI is
/api/v1/promotions/{promotionCode}/coupons.

Promotion Coupon API response structure

If you request a promotion coupon via API, you will receive an object with the following fields:

Field	Description
id	Id of the coupon
code	Unique coupon identifier
used	Number of times this coupon has been used
expiresAt	The date when the coupon will be no longer valid
usageLimit	Number of times this coupon has been used

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the coupon
code	Unique coupon identifier
used	Number of times this coupon has been used
expiresAt	The date when the coupon will be no longer valid
usageLimit	Number of times this coupon has been used
createdAt	Date of creation
updatedAt	Date of last update
perCustomerUsageLimit	Limit of the coupon usage by single customer

---

Note: Read more about *Promotion Coupons in the component docs*.

Creating a Promotion Coupon

To create a new promotion coupon you will need to call the `/api/v1/promotions/{promotionCode}/coupons/` endpoint with the `POST` method.

Definition

```
POST /api/v1/promotions/{promotionCode}/coupons/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
promotionCode	url attribute	Code of the promotion for which the coupon should be created
code	request	**(unique)** Promotion coupon identifier

Example

To create a new promotion coupon for the promotion with `code = HOLIDAY-SALE` use the below method.

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
```

```
    -X POST \
    --data '
        {
            "code": "A3BCB"
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id": 5,
    "code": "A3BCB",
    "used": 0,
    "createdAt": "2017-03-06T13:14:19+0100",
    "updatedAt": "2017-03-06T13:14:19+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/A3BCB"
        },
        "promotion": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE"
        }
    }
}
```

> **Warning:** If you try to create a resource without code, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "usageLimit": {},
            "expiresAt": {},
```

```
            "perCustomerUsageLimit": {},
            "code": {
                "errors": [
                    "Please enter coupon code."
                ]
            }
        }
    }
}
```

You can also create a promotion coupon with additional (not required) fields:

Parameter	Parameter type	Description
usageLimit	request	The information on how many times the coupon can be used
perCustomerUsage-Limit	request	The information on how many times the coupon can be used by one customer
expiresAt	request	The information on when the coupon expires

Example

Here is an example of creating a promotion coupon with additional data for the promotion with `code = HOLIDAY-SALE`.

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "A8BAB",
            "expiresAt": "2020-01-01",
            "usageLimit": 10,
            "perCustomerUsageLimit": 1
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id": 6,
    "code": "A8BAB",
    "usageLimit": 10,
    "used": 0,
    "expiresAt": "2020-01-01T00:00:00+0100",
    "createdAt": "2017-03-06T13:15:27+0100",
    "updatedAt": "2017-03-06T13:15:27+0100",
    "perCustomerUsageLimit": 1,
    "_links": {
```

```
        "self": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/A8BAB"
        },
        "promotion": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE"
        }
    }
}
```

Getting a Single Promotion Coupon

To retrieve the details of a promotion coupon you will need to call the `/api/v1/promotions/{promotionCode}/coupons/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/promotions/{promotionCode}/coupons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested coupon
promotionCode	url attribute	Code of promotion to which the coupon is assigned

Example

To see the details of the promotion coupon with `code = A3BCB` which belongs to the promotion with `code = HOLIDAY-SALE` use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/A3BCB \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *A3BCB* and *HOLIDAY-SALE* codes are just examples. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 5,
    "code": "A3BCB",
    "used": 0,
    "createdAt": "2017-03-06T13:14:19+0100",
    "updatedAt": "2017-03-06T13:14:19+0100",
    "_links": {
        "self": {
```

```
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/A3BCB"
        },
        "promotion": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE"
        }
    }
}
```

Collection of Promotion Coupons

To retrieve a paginated list of promotion coupons you will need to call the /api/v1/promotions/
{promotionCode}/coupons endpoint with the GET method.

Definition

```
GET /api/v1/promotions/{promotionCode}/coupons
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
promotionCode	url attribute	Code of promotion to which the coupons are assigned
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all promotion coupons assigned to the promotion with code = HOLIDAY-SALE use the
below method:

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 1,
    "total": 2,
    "_links": {
        "self": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/?sorting%5Bcode
↪%5D=desc&page=1&limit=4"
        },
        "first": {
```

```
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/?sorting%5Bcode
↪%5D=desc&page=1&limit=4"
        },
        "last": {
            "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/?sorting%5Bcode
↪%5D=desc&page=1&limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 5,
                "code": "A3BCB",
                "used": 0,
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/A3BCB"
                    }
                }
            },
            {
                "id": 6,
                "code": "A8BAB",
                "usageLimit": 10,
                "used": 0,
                "expiresAt": "2020-01-01T00:00:00+0100",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/promotions\/HOLIDAY-SALE\/coupons\/A8BAB"
                    }
                }
            }
        ]
    }
}
```

Updating Promotion Coupon

To fully update a promotion coupon you will need to call the `/api/v1/promotions/{promotionCode}/coupons/{code}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/promotions/{promotionCode}/coupons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Promotion coupon identifier
promotionCode	url attribute	Code of the promotion to which the coupon is assigned
usageLimit	request	The information on how many times the coupon can be used
perCustomerUsage-Limit	request	The information on how many times the coupon can be used by one customer
expiresAt	request	The information on when the coupon expires

Example

To fully update the promotion coupon with `code = A3BCB` for the promotion with `code = HOLIDAY-SALE` use the below method.

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/A3BCB \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "expiresAt": "2020-01-01",
            "usageLimit": 30,
            "perCustomerUsageLimit": 2
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

To partially update a promotion coupon you will need to call the `/api/v1/promotions/{promotionCode}/coupons/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/promotions/{promotionCode}/coupons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Promotion coupon identifier
promotionCode	url attribute	Code of promotion to which the coupon is assigned
usageLimit	request	The information how many times the coupon can be used

Example

To partially update the promotion coupon with `code = A3BCB` for the promotion with `code = HOLIDAY-SALE` use the below method.

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/A3BCB \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "usageLimit": 30
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Promotion coupon

To delete a promotion coupon you will need to call the `/api/v1/promotions/{promotionCode}/coupons/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/promotions/{promotionCode}/coupons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Promotion coupon identifier
promotionCode	url attribute	Code of promotion to which the coupon is assigned

Example

To delete the promotion coupon with `code = A3BCB` from the promotion with `code = HOLIDAY-SALE` use the below method.

```
$ curl http://demo.sylius.com/api/v1/promotions/HOLIDAY-SALE/coupons/A3BCB \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.21 Promotions API

These endpoints will allow you to easily manage promotions. Base URI is */api/v1/promotions*.

Promotion structure

Promotion API response structure

If you request a promotion via API, you will receive an object with the following fields:

Field	Description
id	Id of the promotion
code	Unique promotion identifier
name	The name of the promotion

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the promotion
code	Unique promotion identifier
name	The name of the promotion
startsAt	Start date
endsAt	End date
usageLimit	Promotion's usage limit
used	Number of times this promotion has been used
priority	When exclusive, promotion with top priority will be applied
couponBased	Whether this promotion is triggered by a coupon
exclusive	When true the promotion cannot be applied together with other promotions
rules	Associated rules
actions	Associated actions
createdAt	Date of creation
updatedAt	Date of last update
channels	Collection of channels in which the promotion is available

Note: Read more about *Promotions in the component docs*.

Creating a Promotion

To create a new promotion you will need to call the `/api/v1/promotions/` endpoint with the `POST` method.

Definition

```
POST /api/v1/promotions/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Promotion identifier
name	request	Name of the promotion

Example

To create a new promotion use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "sd-promo",
            "name": "Sunday promotion"
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 6,
    "code": "sd-promo",
    "name": "Sunday promotion",
    "priority": 4,
    "exclusive": false,
    "used": 0,
    "couponBased": false,
    "rules": [],
    "actions": [],
    "createdAt": "2017-02-28T12:05:12+0100",
    "updatedAt": "2017-02-28T12:05:13+0100",
    "channels": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/promotions\/sd-promo"
        }
    }
}
```

Warning: If you try to create a promotion without name or code, you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
   -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```json
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "name": {
                "errors": [
                    "Please enter promotion name."
                ]
            },
            "description": {},
            "exclusive": {},
            "usageLimit": {},
            "startsAt": {
                "children": {
                    "date": {},
                    "time": {}
                }
            },
            "endsAt": {
                "children": {
                    "date": {},
                    "time": {}
                }
            },
            "priority": {},
            "couponBased": {},
            "rules": {},
            "actions": {},
            "channels": {
                "children": [
                    {},
                    {}
                ]
            },
            "code": {
                "errors": [
                    "Please enter promotion code."
                ]
            }
        }
    }
}
```

You can also create a promotion with additional (not required) fields:

Parameter	Parameter type	Description
startsAt	request	Object with date and time fields
endsAt	request	Object with date and time fields
usageLimit	request	Promotion's usage limit
used	request	Number of times this promotion has been used
priority	request	When exclusive, promotion with top priority will be applied
couponBased	request	Whether this promotion is triggered by a coupon
exclusive	request	When true the promotion cannot be applied together with other promotions
rules	request	Collection of rules which determines when the promotion will be applied
actions	request	Collections of actions which will be done when the promotion will be
channels	request	Collection of channels in which the promotion is available

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "christmas-promotion",
            "name": "Christmas Promotion",
            "exclusive": true,
            "priority": 0,
            "couponBased": true,
            "channels": [
                "US_WEB"
            ],
            "startsAt": {
            "date": "2017-12-05",
            "time": "11:00"
            },
            "endsAt": {
                "date": "2017-12-31",
                "time": "11:00"
            },
            "rules": [
                {
                    "type": "nth_order",
                    "configuration": {
                        "nth": 3
                    }
                }
            ],
            "actions": [
                {
                    "type": "order_fixed_discount",
                    "configuration": {
                        "US_WEB": {
                            "amount": 12
                        }
                    }
                }
            ]
```

```
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```json
{
    "id": 7,
    "code": "christmas-promotion",
    "name": "Christmas Promotion",
    "priority": 0,
    "exclusive": true,
    "used": 0,
    "startsAt": "2017-12-05T11:00:00+0100",
    "endsAt": "2017-12-31T11:00:00+0100",
    "couponBased": true,
    "rules": [
        {
            "id": 3,
            "type": "nth_order",
            "configuration": {
                "nth": 3
            }
        }
    ],
    "actions": [
        {
            "id": 3,
            "type": "order_fixed_discount",
            "configuration": {
                "US_WEB": {
                    "amount": 1200
                }
            }
        }
    ],
    "createdAt": "2017-03-06T11:40:38+0100",
    "updatedAt": "2017-03-06T11:40:39+0100",
    "channels": [
        {
            "id": 1,
            "code": "US_WEB",
            "name": "US Web Store",
            "hostname": "localhost",
            "color": "LawnGreen",
            "createdAt": "2017-03-06T11:20:32+0100",
            "updatedAt": "2017-03-06T11:24:37+0100",
            "enabled": true,
            "taxCalculationStrategy": "order_items_based",
            "_links": {
                "self": {
                    "href": "\/api\/v1\/channels\/US_WEB"
                }
```

```
                }
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/promotions\/christmas-promotion"
            },
            "coupons": {
                "href": "\/api\/v1\/promotions\/christmas-promotion\/coupons\/"
            }
        }
    }
}
```

Getting a Single Promotion

To retrieve the details of a promotion you will need to call the `/api/v1/promotions/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/promotions/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested promotion

Example

To see the details of the promotion with `code = sd-promo` use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/sd-promo \
  -H "Authorization: Bearer SampleToken" \
  -H "Accept: application/json"
```

Note: The *sd-promo* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 6,
    "code": "sd-promo",
    "name": "Sunday promotion",
    "priority": 2,
    "exclusive": false,
```

```
    "used": 0,
    "couponBased": false,
    "rules": [],
    "actions": [],
    "createdAt": "2017-02-28T12:05:12+0100",
    "updatedAt": "2017-02-28T12:05:13+0100",
    "channels": [],
    "_links": {
        "self": {
            "href": "\/api\/v1\/promotions\/sd-promo"
        }
    }
}
```

Collection of Promotions

To retrieve a paginated list of promotions you will need to call the `/api/v1/promotions/` endpoint with the `GET` method.

Definition

```
GET /api/v1/promotions/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'priority'
criteria['nameOfCriterion']['searchOption'] criteria['nameOfCriterion']['searchingPhrase']	query	*(optional)* Criterion, option and phrase of filtering, the criteria can be for example 'couponBased' and 'search', option can be 'equal', 'contains'.

To see the first page of all promotions use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 2,
    "_links": {
        "self": {
            "href": "\/api\/v1\/promotions\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/promotions\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/promotions\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 6,
                "code": "sd-promo",
                "name": "Sunday promotion",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/promotions\/sd-promo"
                    }
                }
            },
            {
                "id": 7,
                "code": "christmas-promotion",
                "name": "Christmas Promotion",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/promotions\/christmas-promotion"
                    },
                    "coupons": {
                        "href": "\/api\/v1\/promotions\/christmas-promotion\/coupons\/
"
                    }
                }
            }
        ]
    }
}
```

Updating a Promotion

To fully update a promotion you will need to call the `/api/v1/promotions/{code}` endpoint with the `PUT`
method.

Definition

```
PUT /api/v1/promotions/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique promotion identifier
name	request	Name of the promotion

Example

To fully update the promotion with `code = christmas-promotion` use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/christmas-promotion \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "name": "Christmas special promotion"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform a full promotion update without all the required fields specified, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/promotions/christmas-promotion \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "name": {
```

```
            "errors": [
                "Please enter promotion name."
            ]
        },
        "description": {},
        "exclusive": {},
        "usageLimit": {},
        "startsAt": {
            "children": {
                "date": {},
                "time": {}
            }
        },
        "endsAt": {
            "children": {
                "date": {},
                "time": {}
            }
        },
        "priority": {},
        "couponBased": {},
        "rules": {},
        "actions": {},
        "channels": {
            "children": [
                {},
                {}
            ]
        },
        "code": {}
    }
  }
}
```

To update a promotion partially you will need to call the `/api/v1/promotions/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/promotions/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique promotion identifier

Example

To partially update the promotion with `code = christmas-promotion` use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/christmas-promotion \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
```

```
    -X PATCH \
    --data '
        {
            "exclusive": true,
            "priority": 1
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Promotion

To delete a promotion you will need to call the `/api/v1/promotions/{code}` endpoint with the `DELETE`
method.

Definition

```
DELETE /api/v1/promotions/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique promotion identifier

Example

To delete the promotion with `code = christmas-promotion` use the below method:

```
$ curl http://demo.sylius.com/api/v1/promotions/christmas-promotion \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.22 Provinces API

These endpoints will allow you to easily manage provinces. Base URI is */api/v1/provinces*.

Province API response structure

If you request a province via API, you will receive an object with the following fields:

Field	Description
id	Id of the province
code	Unique province identifier
name	Name of the province

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the province
code	Unique province identifier
name	Name of the province
abbreviation	Abbreviation of the province
createdAt	The province's creation date
updatedAt	The province's last updating date

Note: Read more about *Provinces in the component docs*.

Getting a Single Province

To retrieve the details of a specific province you will need to call the `/api/v1/countries/{countryCode}/provinces/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/countries/{countryCode}/provinces/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
countryCode	url attribute	Code of the country to which the province belongs
code	url attribute	Code of the requested province

Example

To see the details of the province with `code = PL-MZ` which belongs to the country with `code = PL` use the below method:

```
$ curl http://demo.sylius.com/api/v1/countries/PL/provinces/PL-MZ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *PL* ans *PL-MZ* codes are just examples. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "code": "PL-MZ",
    "name": "mazowieckie",
    "_links": {
        "self": {
            "href": "\/api\/v1\/countries\/PL\/provinces\/PL-MZ"
        },
        "country": {
            "href": "\/api\/v1\/countries\/PL"
        }
    }
}
```

Deleting a Province

To delete a province you will need to call the `/api/v1/countries/{countryCode}/provinces/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/countries/{countryCode}/provinces/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
countryCode	url attribute	Code of the country to which the province belongs
code	url attribute	Code of the requested province

Example

```
$ curl http://sylius.test/api/v1/countries/PL/provinces/PL-MZ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.23 Shipments API

These endpoints will allow you to easily present shipments. Base URI is */api/v1/shipments*.

Shipment API response structure

If you request a shipping via API, you will receive an object with the following fields:

Field	Description
id	Unique id of the shipment
state	*State of the shipping process*
method	*The shipping method object serialized* for cart
_links[self]	Link to itself
_links[shipping-method]	Link to related shipping method
_links[order]	Link to related order

Getting a Single Shipment

To retrieve the details of a shipment you will need to call the `/api/v1/shipments/{id}` endpoint with the `GET` method.

Definition

```
GET /api/v1/shipments/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
id	url attribute	Id of the requested shipment

Example

To see the details of the shipment method with `id = 20` use the below method:

```
$ curl http://demo.sylius.com/api/v1/shipments/20 \
   -H "Authorization: Bearer SampleToken" \
   -H "Accept: application/json"
```

Note: The *id = 20* is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id":20,
    "state":"ready",
    "method":{
        "id":1,
        "code":"ups",
        "enabled":true,
```

(continues on next page)

```
        "_links":{
            "self":{
                "href":"\/api\/v1\/shipping-methods\/ups"
            },
            "zone":{
                "href":"\/api\/v1\/zones\/US"
            }
        }
    },
    "_links":{
        "self":{
            "href":"\/api\/v1\/shipments\/20"
        },
        "shipping-method":{
            "href":"\/api\/v1\/shipping-methods\/ups"
        },
        "order":{
            "href":"\/api\/v1\/orders\/20"
        }
    }
}
```

Collection of Shipments

To retrieve a paginated list of shipments you will need to call the `/api/v1/shipments/` endpoint with the `GET` method.

Definition

```
GET /api/v1/shipments/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting[createdAt]	query	*(optional)* Order of sorting on created at field (asc by default)
sorting[updatedAt]	query	*(optional)* Order of sorting on updated at field (desc/asc)

Example

To see first page of paginated list of shipments with two shipments on each page use the below snippet:

```
$ curl http://demo.sylius.com/api/v1/shipments/\?limit\=2 \
  -H "Authorization: Bearer SampleToken" \
  -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page":1,
    "limit":2,
    "pages":10,
    "total":20,
    "_links":{
        "self":{
            "href":"\/api\/v1\/shipments\/?page=1&limit=2"
        },
        "first":{
            "href":"\/api\/v1\/shipments\/?page=1&limit=2"
        },
        "last":{
            "href":"\/api\/v1\/shipments\/?page=10&limit=2"
        },
        "next":{
            "href":"\/api\/v1\/shipments\/?page=2&limit=2"
        }
    },
    "_embedded":{
        "items":[
            {
                "id":1,
                "state":"ready",
                "method":{
                    "id":2,
                    "code":"dhl_express",
                    "enabled":true,
                    "_links":{
                        "self":{
                            "href":"\/api\/v1\/shipping-methods\/dhl_express"
                        },
                        "zone":{
                            "href":"\/api\/v1\/zones\/US"
                        }
                    }
                },
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/shipments\/1"
                    },
                    "shipping-method":{
                        "href":"\/api\/v1\/shipping-methods\/dhl_express"
                    },
                    "order":{
                        "href":"\/api\/v1\/orders\/1"
                    }
                }
            },
            {
                "id":2,
                "state":"ready",
                "method":{
```

```
                "id":2,
                "code":"dhl_express",
                "enabled":true,
                "_links":{
                    "self":{
                        "href":"\/api\/v1\/shipping-methods\/dhl_express"
                    },
                    "zone":{
                        "href":"\/api\/v1\/zones\/US"
                    }
                }
            },
            "_links":{
                "self":{
                    "href":"\/api\/v1\/shipments\/2"
                },
                "shipping-method":{
                    "href":"\/api\/v1\/shipping-methods\/dhl_express"
                },
                "order":{
                    "href":"\/api\/v1\/orders\/2"
                }
            }
        }
    ]
    }
}
```

5.1.24 Shipping Categories API

These endpoints will allow you to easily manage shipping categories. Base URI is */api/v1/shipping-categories*.

When you get a collection of resources, "Default" serialization group will be used and following fields will be exposed:

Field	Description
id	Id of shipping category
name	Name of shipping category
code	Unique shipping category identifier

If you request for a more detailed data, you will receive an object with following fields:

Field	Description
id	Id of shipping category
name	Name of shipping category
code	Unique shipping category identifier
description	Description of shipping category

Note: Read more about *Shipping Categories in the component docs*.

Creating Shipping Category

To create a new shipping category you will need to call the `/api/v1/shipping-categories/` endpoint with the `POST` method.

Definition

```
POST /api/v1/shipping-categories/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
name	request	Name of creating shipping category
code	request	**(unique)** Shipping category identifier
description	request	*(optional)* Description of creating shipping category

Example

To create a new shipping category use the below method.

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "name": "Light",
            "description": "Light weight items",
            "code": "SC3"
        }
    '
```

Exemplary Response

```
STATUS: 201 Created
```

```
{
    "id": 3,
    "code": "SC3",
    "name": "Light",
    "description": "Light weight items",
    "_links": {
        "self": {
            "href": "\/api\/shipping-categories\/SC3"
        }
    }
}
```

If you try to create a resource without name or code, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "name": {
                "errors": [
                    "Please enter shipping category name."
                ]
            },
            "code": {
                "errors": [
                    "Please enter shipping category code."
                ]
            },
            "description": []
        }
    }
}
```

Getting a Single Shipping Category

To retrieve the details of a shipping category you will need to call the `/api/v1/shipping-categories/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/shipping-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested resource

Example

To see the details of the shipping category with `code = SC3` use the below method:

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/SC3 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *SC3* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "code": "SC3",
    "name": "Light",
    "createdAt": "2017-03-06T12:41:33+0100",
    "updatedAt": "2017-03-06T12:44:01+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/shipping-categories\/SC3"
        }
    }
}
```

Collection of Shipping Categories

To retrieve a paginated list of shipping categories you will need to call the `/api/v1/shipping-categories/` endpoint with the `GET` method.

Definition

```
GET /api/v1/shipping-categories/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
limit	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all shipping categories assigned to the promotion with `code = HOLIDAY-SALE` use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 1,
    "total": 2,
    "_links": {
        "self": {
            "href": "\/api\/v1\/shipping-categories\/?sorting%5Bcode%5D=desc&page=1&
↪limit=4"
        },
        "first": {
            "href": "\/api\/v1\/shipping-categories\/?sorting%5Bcode%5D=desc&page=1&
↪limit=4"
        },
        "last": {
            "href": "\/api\/v1\/shipping-categories\/?sorting%5Bcode%5D=desc&page=1&
↪limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "SC3",
                "name": "Light",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/shipping-categories\/SC3"
                    }
                }
            },
            {
                "id": 2,
                "code": "SC1",
                "name": "Regular",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/shipping-categories\/SC1"
                    }
                }
            }
        ]
    }
}
```

Updating Shipping Category

To fully update a shipping category you will need to call the `/api/v1/shipping-categories/{code}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/shipping-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested resource
name	request	Name of creating shipping category
description	request	Description of creating shipping category

Example

To fully update the shipping category with `code = SC3` use the below method.

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/SC3 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "name": "Ultra light",
            "description": "Ultra light weight items"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform full shipping category update without all the required fields specified, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/SC3 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
```

```
        "children": {
            "name": {
                "errors": [
                    "Please enter shipping category name."
                ]
            },
            "description": []
        }
    }
}
```

To partially update a shipping category you will need to call the `/api/v1/shipping-categories/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/shipping-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested resource
name	request	*(optional)* Name of creating shipping category
description	request	*(optional)* Description of creating shipping category

Example

To partially update the shipping category with `code = SC3` use the below method.

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/SC3 \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "name": "Light"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting Shipping Category

To delete a shipping category you will need to call the `/api/v1/shipping-categories/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/shipping-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of requested resource

Example

To delete the shipping category with `code = SC3` use the below method.

```
$ curl http://demo.sylius.com/api/v1/shipping-categories/SC3 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.25 Shipping Methods API

These endpoints will allow you to easily manage shipping methods. Base URI is */api/v1/shipping-methods*.

Shipping Method API response structure

If you request a shipping method via API, you will receive an object with the following fields:

Field	Description
id	Id of the shipping method
code	Unique shipping method identifier
name	The name of the shipping method
enabled	Determine if the shipping method is enabled
categoryRequirement	Reference to constant from ShippingMethodInterface
calculator	Reference to constant from DefaultCalculators
configuration	Extra configuration for the calculator
createdAt	Date of creation
updatedAt	Date of last update

Note: Read more about *Shipping Methods in the component docs*.

Getting a Single Shipping Method

To retrieve the details of a shipping method you will need to call the `/api/v1/shipping-methods/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/shipping-methods/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested shipping method

Example

To see the details of the shipping method with `code = ups` use the below method:

```
$ curl http://demo.sylius.com/api/v1/shipping-methods/ups \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *ups* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "id": 1,
    "code": "ups",
    "enabled": true,
    "_links": {
        "self": {
            "href": "\/api\/v1\/shipping-methods\/ups"
        },
        "zone": {
            "href": "\/api\/v1\/zones\/US"
        }
    }
}
```

5.1.26 Tax Categories API

These endpoints will allow you to easily manage tax categories. Base URI is *api/v1/tax-categories*.

Tax Category structure

Tax Category API response structure

If you request a tax category via API, you will receive an object with the following fields:

Field	Description
id	Id of the tax category
code	Unique tax category identifier
name	Name of the tax category

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the tax category
code	Unique tax category identifier
name	Name of the tax category
description	Description of the tax category
createdAt	Date of creation
updatedAt	Date of last update

Note: Read more about *the Tax Category model in the component docs*.

Creating a Tax Category

To create a new tax category you will need to call the `/api/v1/tax-categories/` endpoint with the `POST` method.

Definition

```
POST /api/v1/tax-categories/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Tax category identifier
name	request	Name of the tax category

Example

To create a new tax category use the below method:

```
$ curl http://demo.sylius.com/api/v1/tax-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
```

```
        {
            "code": "food",
            "name": "Food"
        }
'
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 4,
    "code": "food",
    "name": "Food",
    "createdAt": "2017-02-21T12:49:48+0100",
    "updatedAt": "2017-02-21T12:49:50+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-categories\/food"
        }
    }
}
```

> **Warning:** If you try to create a tax category without name or code you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "name": {
                "errors": [
                    "Please enter tax category name."
                ]
            },
```

```
            "description": {},
            "code": {
                "errors": [
                    "Please enter tax category code."
                ]
            }
        }
    }
}
```

You can also create a tax category with additional (not required) fields:

Parameter	Parameter type	Description
description	request	Description of the tax category

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "food",
            "name": "Food",
            "description": "The food category."
        }
     '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 5,
    "code": "food",
    "name": "Food",
    "description": "The food category.",
    "createdAt": "2017-02-21T12:58:41+0100",
    "updatedAt": "2017-02-21T12:58:42+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-categories\/food"
        }
    }
}
```

Getting a Single Tax Category

To retrieve the details of a tax category you will need to call the `/api/v1/tax-categories/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/tax-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique tax category identifier

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/food \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *food* is an exemplary value. Your value can be different. Check in the list of all tax categories if you are not sure which code should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 5,
    "code": "food",
    "name": "Food",
    "description": "The food category.",
    "createdAt": "2017-02-21T12:58:41+0100",
    "updatedAt": "2017-02-21T12:58:42+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-categories\/food"
        }
    }
}
```

Collection of Tax Categories

To retrieve a paginated list of tax categories you will need to call the `/api/v1/tax-categories/` endpoint with the `GET` method.

Definition

```
GET /api/v1/tax-categories/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

To see the first page of all tax categories use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 4,
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-categories\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/tax-categories\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/tax-categories\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "clothing",
                "name": "Clothing",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-categories\/clothing"
                    }
                }
            },
            {
                "id": 2,
                "code": "books",
                "name": "Books",
                "_links": {
```

```
                    "self": {
                        "href": "\/api\/v1\/tax-categories\/books"
                    }
                }
            },
            {
                "id": 3,
                "code": "other",
                "name": "Other",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-categories\/other"
                    }
                }
            },
            {
                "id": 5,
                "code": "food",
                "name": "Food",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-categories\/food"
                    }
                }
            }
        ]
    }
}
```

Updating a Tax Category

To fully update a tax category you will need to call the `/api/v1/tax-categories/{code}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/tax-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique tax category identifier
name	request	Name of the tax category
description	request	Description of the tax category

Example

To fully update the tax category with `code = food` use the below method:

```
$ curl http://demo.sylius.com/api/v1/tax-categories/food \
    -H "Authorization: Bearer SampleToken" \
```

```
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "name": "Vegetables",
            "description": "The category of food: vegetables"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform a full tax category update without all the required fields specified, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/food \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "name": {
                "errors": [
                    "Please enter tax category name."
                ]
            },
            "description": {},
            "code": {}
        }
    }
}
```

To update a tax category partially you will need to call the `/api/v1/tax-categories/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/tax-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique tax category identifier

Example

To partially update the tax category with `code = food` use the below method:

```
$ curl http://demo.sylius.com/api/v1/tax-categories/food \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "description": "The category of food: vegetables"
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Tax Category

To delete a tax category you will need to call the `/api/v1/tax-categories/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/tax-categories/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique tax category identifier

Example

```
$ curl http://demo.sylius.com/api/v1/tax-categories/food \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.27 Tax Rates API

These endpoints will allow you to easily manage tax rates. Base URI is */api/v1/tax-rates*.

Tax Rate structure

Tax Rate API response structure

If you request a tax rate via API, you will receive an object with the following fields:

Field	Description
id	Id of the tax rate
code	Unique tax rate identifier
name	The name of the tax rate
amount	Amount as float (for example 0,23)
includedInPrice	Is the tax included in price?
calculator	Type of calculator
createdAt	Date of creation
updatedAt	Date of last update

Note: Read more about *Tax Rates in the component docs*.

Getting a Single Tax Rate

To retrieve the details of a tax rate you will need to call the `/api/v1/tax-rates/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/tax-rates/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the requested tax rate

Example

To see the details of the tax rate with `code = clothing_sales_tax_7` use the below method:

```
$ curl http://demo.sylius.com/api/v1/tax-rates/clothing_sales_tax_7 \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *clothing_sales_tax_7* code is just an example. Your value can be different.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 1,
    "code": "clothing_sales_tax_7",
    "name": "Clothing Sales Tax 7%",
    "amount": 0.07,
    "includedInPrice": false,
    "calculator": "default",
    "createdAt": "2017-02-17T15:01:15+0100",
    "updatedAt": "2017-02-17T15:01:15+0100",
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-rates\/clothing_sales_tax_7"
        },
        "category": {
            "href": "\/api\/v1\/tax-categories\/clothing"
        },
        "zone": {
            "href": "\/api\/v1\/zones\/US"
        }
    }
}
```

Collection of Tax Rates

To retrieve a paginated list of tax rates you will need to call the `/api/v1/tax-rates/` endpoint with the `GET` method.

Definition

```
GET /api/v1/tax-rates/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
page	query	*(optional)* Number of the page, by default = 1
paginate	query	*(optional)* Number of items to display per page, by default = 10

To see the first page of all tax rates use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/tax-rates/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 3,
    "_links": {
        "self": {
            "href": "\/api\/v1\/tax-rates\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/tax-rates\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/tax-rates\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "clothing_sales_tax_7",
                "name": "Clothing Sales Tax 7%",
                "amount": 0.07,
                "includedInPrice": false,
                "calculator": "default",
                "createdAt": "2017-02-17T15:01:15+0100",
                "updatedAt": "2017-02-17T15:01:15+0100",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-rates\/clothing_sales_tax_7"
                    },
                    "category": {
                        "href": "\/api\/v1\/tax-categories\/clothing"
                    },
                    "zone": {
                        "href": "\/api\/v1\/zones\/US"
                    }
                }
            },
            {
                "id": 2,
                "code": "books_sales_tax_2",
                "name": "Books Sales Tax 2%",
                "amount": 0.02,
                "includedInPrice": true,
```

(continues on next page)

```json
                "calculator": "default",
                "createdAt": "2017-02-17T15:01:15+0100",
                "updatedAt": "2017-02-17T15:01:15+0100",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-rates\/books_sales_tax_2"
                    },
                    "category": {
                        "href": "\/api\/v1\/tax-categories\/books"
                    },
                    "zone": {
                        "href": "\/api\/v1\/zones\/US"
                    }
                }
            },
            {
                "id": 3,
                "code": "sales_tax_20",
                "name": "Sales Tax 20%",
                "amount": 0.2,
                "includedInPrice": true,
                "calculator": "default",
                "createdAt": "2017-02-17T15:01:15+0100",
                "updatedAt": "2017-02-17T15:01:15+0100",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/tax-rates\/sales_tax_20"
                    },
                    "category": {
                        "href": "\/api\/v1\/tax-categories\/other"
                    },
                    "zone": {
                        "href": "\/api\/v1\/zones\/US"
                    }
                }
            }
        ]
    }
}
```

5.1.28 Taxons API

These endpoints will allow you to easily manage taxons. Base URI is */api/v1/taxons*.

Taxon API response structure

If you request a taxon via API, you will receive an object with the following fields:

Field	Description
id	Id of the taxon
code	Unique taxon identifier
root	The main ancestor of the taxon
parent	Parent of the taxon
translations	Collection of translations (each contains slug, name and description in the respective language)
position	The position of the taxon among other taxons
images	Images assigned to the taxon

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the taxon
code	Unique taxon identifier
root	The main ancestor of the taxon
parent	Parent of the taxon
translations	Collection of translations (each contains slug, name and description in the respective language)
position	Position of the taxon among other taxons
images	Images assigned to the taxon
left	Location within the whole taxonomy
right	Location within the whole taxonomy
level	How deep the taxon is in the tree
children	Descendants of the taxon

Note: Read more about *Taxons*.

Creating a Taxon

To create a new taxon you will need to call the `/api/v1/taxons/` endpoint with the `POST` method.

Definition

```
POST /api/v1/taxons/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	request	**(unique)** Taxon identifier
translations['localeCode']['name']	request	Taxon name
translations['localeCode']['slug']	request	**(unique)** Taxon slug

Example

To create new taxon use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "toys",
            "translations": {
                "en_US": {
                    "name": "Toys",
                    "slug": "category/toys"
                }
            }
        }
    '
```

Note: If you want to create your taxon as a child of another taxon, you should pass also the parent taxon's code.

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 11,
    "code": "toys",
    "children": [],
    "left": 1,
    "right": 2,
    "level": 0,
    "position": 1,
    "translations": [],
    "images": [],
    "_links": {
        "self": {
            "href": "/api/v1/taxons/11"
        }
    }
}
```

Warning: If you try to create a taxon without code you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/taxons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "children": {
            "translations": {},
            "images": {},
            "code": {
                "errors": [
                    "Please enter taxon code."
                ]
            },
            "parent": {}
        }
    }
}
```

You can also create a taxon with additional (not required) fields:

Parameter	Parameter type	Description
translations['localeCode']['description']	request	Description of the taxon
parent	request	The parent taxon's code
images	request	Images codes assigned to the taxon

Example

```
$ curl http://demo.sylius.com/api/v1/taxons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code":"toys",
            "translations":{
                "en_US": {
                    "name": "Toys",
                    "slug": "category/toys",
                    "description": "Toys for boys"
                }
            },
            "parent": "category",
            "images": [
                {
                    "type": "ford"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```json
{
    "name": "toys",
    "id": 9,
    "code": "toys",
    "root": {
        "name": "Category",
        "id": 1,
        "code": "category",
        "children": [
            {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "children": [],
                "left": 2,
                "right": 7,
                "level": 1,
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/5"
                    }
                }
            }
        ],
        "left": 1,
        "right": 10,
        "level": 0,
        "position": 0,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 1,
                "name": "Category",
                "slug": "category",
                "description": "Consequatur illo amet aliquam."
            }
        },
        "images": [],
        "_links": {
            "self": {
                "href": "\/api\/v1\/taxons\/1"
            }
        }
    },
    "parent": {
        "name": "Category",
        "id": 1,
        "code": "category",
        "children": [
            {
```

```
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "children": [],
                "left": 2,
                "right": 7,
                "level": 1,
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/5"
                    }
                }
            }
        ],
        "left": 1,
        "right": 10,
        "level": 0,
        "position": 0,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 1,
                "name": "Category",
                "slug": "category",
                "description": "Consequatur illo amet aliquam."
            }
        },
        "images": [],
        "_links": {
            "self": {
                "href": "\/api\/v1\/taxons\/1"
            }
        }
    },
    "children": [],
    "left": 8,
    "right": 9,
    "level": 1,
    "position": 1,
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 9,
            "name": "toys",
            "slug": "toys",
            "description": "Toys for boys"
        }
    },
    "images": [
        {
            "id": 1,
            "type": "ford",
            "path": "b9/65/01cec3d87aa2b819e195331843f6.jpeg"
        }
```

```
    ],
    "_links": {
        "self": {
            "href": "\/api\/v1\/taxons\/9"
        }
    }
}
```

Note: The images should be passed in array as an attribute (files) of request. See how it is done in Sylius here.

Getting a Single Taxon

To retrieve the details of a taxon you will need to call the `/api/v1/taxons/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/taxons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Identifier of the requested taxon

Example

To see the details of the taxon with `code = toys` use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/toys \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *toys* value was taken from the previous create response. Your value can be different. Check in the list of all taxons if you are not sure which id should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "name": "toys",
    "id": 9,
    "code": "toys",
    "root": {
        "name": "Category",
```

```
        "id": 1,
        "code": "category",
        "children": [
            {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "children": [],
                "left": 2,
                "right": 7,
                "level": 1,
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/5"
                    }
                }
            }
        ],
        "left": 1,
        "right": 10,
        "level": 0,
        "position": 0,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 1,
                "name": "Category",
                "slug": "category",
                "description": "Consequatur illo amet aliquam."
            }
        },
        "images": [],
        "_links": {
            "self": {
                "href": "\/api\/v1\/taxons\/1"
            }
        }
    },
    "parent": {
        "name": "Category",
        "id": 1,
        "code": "category",
        "children": [
            {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "children": [],
                "left": 2,
                "right": 7,
                "level": 1,
                "position": 0,
                "translations": [],
                "images": [],
```

```
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/taxons\/5"
                        }
                    }
                }
            ],
            "left": 1,
            "right": 10,
            "level": 0,
            "position": 0,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 1,
                    "name": "Category",
                    "slug": "category",
                    "description": "Consequatur illo amet aliquam."
                }
            },
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        }
    },
    "children": [],
    "left": 8,
    "right": 9,
    "level": 1,
    "position": 1,
    "translations": {
        "en_US": {
            "locale": "en_US",
            "id": 9,
            "name": "toys",
            "slug": "toys",
            "description": "Toys for boys"
        }
    },
    "images": [
        {
            "id": 1,
            "type": "ford",
            "path": "b9/65/01cec3d87aa2b819e195331843f6.jpeg"
        }
    ],
    "_links": {
        "self": {
            "href": "\/api\/v1\/taxons\/9"
        }
    }
}
}
```

Collection of Taxons

To retrieve a paginated list of taxons you will need to call the `/api/v1/taxons/` endpoint with the `GET` method.

Definition

```
GET /api/v1/taxons/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

To see the first page of all taxons use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/taxons/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 5,
    "_links": {
        "self": {
            "href": "\/api\/v1\/taxons\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/taxons\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/taxons\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "name": "Category",
                "id": 1,
                "code": "category",
                "position": 0,
```

```
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 1,
                    "name": "Category",
                    "slug": "category",
                    "description": "Consequatur illo amet aliquam."
                }
            },
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        }
    },
    {
        "name": "T-Shirts",
        "id": 5,
        "code": "t_shirts",
        "root": {
            "name": "Category",
            "id": 1,
            "code": "category",
            "position": 0,
            "translations": [],
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        },
        "parent": {
            "name": "Category",
            "id": 1,
            "code": "category",
            "position": 0,
            "translations": [],
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        },
        "position": 0,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 5,
                "name": "T-Shirts",
                "slug": "t-shirts",
                "description": "Modi aut laborum aut sint aut ea itaque porro.
↪"
            }
        },
```

```
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/5"
                }
            }
        },
        {
            "name": "Men",
            "id": 6,
            "code": "mens_t_shirts",
            "root": {
                "name": "Category",
                "id": 1,
                "code": "category",
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/1"
                    }
                }
            },
            "parent": {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/5"
                    }
                }
            },
            "position": 0,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 6,
                    "name": "Men",
                    "slug": "t-shirts\/men",
                    "description": "Reprehenderit vero atque eaque sunt␣
→perferendis est."
                }
            },
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/6"
                }
            }
        },
        {
            "name": "Women",
```

```
            "id": 7,
            "code": "womens_t_shirts",
            "root": {
                "name": "Category",
                "id": 1,
                "code": "category",
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/1"
                    }
                }
            },
            "parent": {
                "name": "T-Shirts",
                "id": 5,
                "code": "t_shirts",
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/taxons\/5"
                    }
                }
            },
            "position": 1,
            "translations": {
                "en_US": {
                    "locale": "en_US",
                    "id": 7,
                    "name": "Women",
                    "slug": "t-shirts\/women",
                    "description": "Illum quia beatae assumenda impedit."
                }
            },
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/7"
                }
            }
        },
        {
            "name": "toys",
            "id": 9,
            "code": "toys",
            "root": {
                "name": "Category",
                "id": 1,
                "code": "category",
                "position": 0,
                "translations": [],
                "images": [],
                "_links": {
```

```
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        },
        "parent": {
            "name": "Category",
            "id": 1,
            "code": "category",
            "position": 0,
            "translations": [],
            "images": [],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/taxons\/1"
                }
            }
        },
        "position": 1,
        "translations": {
            "en_US": {
                "locale": "en_US",
                "id": 9,
                "name": "toys",
                "slug": "toys",
                "description": "Toys for boys"
            }
        },
        "images": [],
        "_links": {
            "self": {
                "href": "\/api\/v1\/taxons\/9"
            }
        }
    }
  ]
 }
}
```

Updating Taxon

To fully update a taxon you will need to call the `/api/v1/taxons/{code}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/taxons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	**(unique)** Identifier of the requested taxon
translations['localeCode']['name']	request	*(optional)* Name of the taxon
translations['localeCode']['slug']	request	*(optional)* **(unique)** Slug
translations['localeCode']['description']	request	*(optional)* Description of the taxon
parent	request	*(optional)* The parent taxon's code
images	request	*(optional)* Images codes assigned to the taxon

Example

To fully update the taxon with `code = toys` use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/toys \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Dolls",
                    "slug": "dolls"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

To update a taxon partially you will need to call the `/api/v1/taxons/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/taxons/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	**(unique)** Identifier of the requested taxon

Example

To partially update the taxon with `code = toys` use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/toys \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "translations": {
                "en_US": {
                    "name": "Dolls"
                }
            }
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Taxon

To delete a taxon you will need to call the `/api/v1/taxons/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/taxons/{id}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	**(unique)** Identifier of the requested taxon

Example

To delete the taxon with `code` = `toys` use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/toys \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

Set position of product in a Taxon

The products in Sylius can by grouped by taxon, therefore for every product there is a relation between the product and the assigned taxon. What is more, every product can have a specific position in the taxon to which it belongs. To put products in a specific order you will need to call the `/api/v1/taxons/{code}/products` endpoint wih the `PUT` method.

Definition

```
PUT /api/v1/taxons/{code}/products
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Code of the taxon in which the order of product will be changed

Example

To change the order of products with codes `yellow_t_shirt` and `princess_t_shirt` in taxon with code `womens_t_shirts` use the below method:

```
$ curl http://demo.sylius.com/api/v1/taxons/womens_t_shirts/products \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "productsPositions": [
                {
                    "productCode": "yellow_t_shirt",
                    "position": 3
                },
                {
                    "productCode": "princess_t_shirt",
                    "position": 0
                }
            ]
        }
    '
```

Note: Remember the *yellow_t_shirt* and *princess_t_shirt* and *womens_t_shirts* are just exemplary codes and you can change them for the ones you need. Check in the list of all products if you are not sure which codes should be used.

Exemplary Response

```
STATUS: 204 NO CONTENT
```

5.1.29 Zones API

These endpoints will allow you to easily manage zones. Base URI is */api/v1/zones*.

Zone structure

Zone API response structure

If you request a zone via API, you will receive an object with the following fields:

Field	Description
id	Id of the zone
code	Unique zone identifier
name	Name of the zone
type	Type of the zone

If you request for more detailed data, you will receive an object with the following fields:

Field	Description
id	Id of the zone
code	Unique zone identifier
name	Name of the zone
type	Type of the zone
scope	Scope of the zone
members	Members of the zone
createdAt	Date of creation
updatedAt	Date of last update

Note: Read more about *the Zone model in the component docs*.

Creating a Zone

To create a new zone you will need to call the `/api/v1/zones/{type}` endpoint with the `POST` method.

Definition

```
POST /api/v1/zones/{type}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
type	url attribute	Type of a creating zone
code	request	**(unique)** Zone identifier
name	request	Name of the zone
scope	request	Scope of the zone
members	request	Members of the zone

Note: Read more about *Zone types in the component docs*.

Example

To create a new country zone use the below method:

```
$ curl http://demo.sylius.com/api/v1/zones/country \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X POST \
    --data '
        {
            "code": "EU",
            "name": "European Union",
            "scope": "all",
            "members": [
                {
                    "code": "PL"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 201 CREATED
```

```
{
    "id": 2,
    "code": "EU",
    "name": "European Union",
    "type": "country",
    "scope": "all",
    "_links": {
        "self": {
            "href": "\/api\/v1\/zones\/EU"
        }
    }
}
```

Warning: If you try to create a zone without name, code, scope or member, you will receive a `400 Bad Request` error, that will contain validation errors.

Example

```
$ curl http://demo.sylius.com/api/v1/zones/country \
    -H "Authorization: Bearer SampleToken" \
```

```
-H "Content-Type: application/json" \
-X POST
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "errors": [
            "Please add at least 1 zone member."
        ],
        "children": {
            "name": {
                "errors": [
                    "Please enter zone name."
                ]
            },
            "type": {},
            "scope": {
                "errors": [
                    "Please enter the scope."
                ]
            },
            "code": {
                "errors": [
                    "Please enter zone code."
                ]
            },
            "members": {}
        }
    }
}
```

Getting a Single Zone

To retrieve the details of a zone you will need to call the `/api/v1/zone/{code}` endpoint with the `GET` method.

Definition

```
GET /api/v1/zones/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique zone identifier

Example

To see the details of the zone with `code` = `EU` use the below method:

```
$ curl http://demo.sylius.com/api/v1/zones/EU \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Note: The *EU* code is an exemplary value. Your value can be different. Check in the list of all zones if you are not sure which code should be used.

Exemplary Response

```
STATUS: 200 OK
```

```
{
    "id": 2,
    "code": "EU",
    "name": "European Union",
    "type": "country",
    "scope": "all",
    "_links": {
        "self": {
            "href": "\/api\/v1\/zones\/EU"
        }
    }
}
```

Collection of Zones

To retrieve a paginated list of zones you will need to call the `/api/v1/zones/` endpoint with the `GET` method.

Definition

```
GET /api/v1/zones/
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
limit	query	*(optional)* Number of items to display per page, by default = 10
sorting['nameOfField']['direction']	query	*(optional)* Field and direction of sorting, by default 'desc' and 'createdAt'

To see the first page of all zones use the below method:

Example

```
$ curl http://demo.sylius.com/api/v1/zones/ \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json"
```

Exemplary Response

```
STATUS: 200 OK
```

```json
{
    "page": 1,
    "limit": 10,
    "pages": 1,
    "total": 2,
    "_links": {
        "self": {
            "href": "\/api\/v1\/zones\/?page=1&limit=10"
        },
        "first": {
            "href": "\/api\/v1\/zones\/?page=1&limit=10"
        },
        "last": {
            "href": "\/api\/v1\/zones\/?page=1&limit=10"
        }
    },
    "_embedded": {
        "items": [
            {
                "id": 1,
                "code": "US",
                "name": "United States of America",
                "type": "country",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/zones\/US"
                    }
                }
            },
            {
                "id": 2,
                "code": "EU",
                "name": "European Union",
                "type": "country",
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/zones\/EU"
                    }
                }
            }
        ]
    }
}
```

Updating a Zone

To fully update a zone you will need to call the `/api/v1/zones/{code}` endpoint with the `PUT` method.

Definition

```
PUT /api/v1/zones/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique zone identifier
name	request	Name of the zone
scope	request	Scope of the zone
members	request	Members of the zone

Example

To fully update the zone with `code = EU` use the below method:

```
$ curl http://demo.sylius.com/api/v1/zones/EU \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT \
    --data '
        {
            "name": "European Union Zone",
            "scope": "shipping",
            "members": [
                {
                    "code": "DE"
                }
            ]
        }
    '
```

Exemplary Response

```
STATUS: 204 No Content
```

If you try to perform a full zone update without all the required fields specified, you will receive a `400 Bad Request` error.

Example

```
$ curl http://demo.sylius.com/api/v1/zones/EU \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PUT
```

Exemplary Response

```
STATUS: 400 Bad Request
```

```json
{
    "code": 400,
    "message": "Validation Failed",
    "errors": {
        "errors": [
            "Please add at least 1 zone member."
        ],
        "children": {
            "name": {
                "errors": [
                    "Please enter zone name."
                ]
            },
            "type": {},
            "scope": {
                "errors": [
                    "Please enter the scope."
                ]
            },
            "code": {},
            "members": {}
        }
    }
}
```

To update a zone partially you will need to call the `/api/v1/zones/{code}` endpoint with the `PATCH` method.

Definition

```
PATCH /api/v1/zones/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique zone identifier
scope	request	Scope of the zone

Example

To partially update the zone with `code = EU` use the below method:

```
$ curl http://demo.sylius.com/api/v1/zones/EU \
    -H "Authorization: Bearer SampleToken" \
    -H "Content-Type: application/json" \
    -X PATCH \
    --data '
        {
            "scope": "tax"
```

```
    }
'
```

Exemplary Response

```
STATUS: 204 No Content
```

Deleting a Zone

To delete a zone you will need to call the `/api/v1/zones/{code}` endpoint with the `DELETE` method.

Definition

```
DELETE /api/v1/zones/{code}
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
code	url attribute	Unique zone identifier

Example

To delete the zone with `code = EU` use the below method:

```
$ curl http://demo.sylius.com/api/v1/zones/EU \
    -H "Authorization: Bearer SampleToken" \
    -H "Accept: application/json" \
    -X DELETE
```

Exemplary Response

```
STATUS: 204 No Content
```

5.1.30 Sorting and filtration

In the Sylius API, a list of resources can be sorted and filtered by passed url query parameters. Here you can find examples how to do it with sample resources.

Note: To find out by which fields the api resources can be sorted and how they can be filtered you should check the grid configuration of these here

How to sort resources?

Let's assume that you want to sort products by code in descending order. In this case you should call the `/api/v1/products/` endpoint with the `GET` method and provide sorting query parameters.

Definition

```
GET /api/v1/products/?sorting\[{nameOfField}\]={direction}'
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
nameOfField	query	*(required)* Name of field by which the resource will be sorted
direction	query	*(required)* Define a direction of ordering
limit	query	*(optional)* Number of items to display per page, by default = 10

Example

```
$  curl 'http://demo.sylius.com/api/v1/products/?sorting\[code\]=desc&limit=4' \
        -H "Authorization: Bearer SampleToken"
```

Exemplary response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 15,
    "total": 60,
    "_links": {
        "self": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
        "first": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=1&limit=4"
        },
        "last": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=15&limit=4"
        },
        "next": {
            "href": "\/api\/v1\/products\/?sorting%5Bcode%5D=desc&page=2&limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "name": "Book \"facilis\" by Deborah Schmitt",
                "id": 32,
                "code": "fe1a18b9-f67a-35fb-bc64-78a60c724493",
                "options": [],
```

(continues on next page)

```
                "averageRating": 3,
                "images": [
                    {
                        "id": 63,
                        "type": "main"
                    },
                    {
                        "id": 64,
                        "type": "thumbnail"
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/fe1a18b9-f67a-35fb-bc64-
→78a60c724493"
                    }
                }
            },
            {
                "name": "Book \"voluptate\" by Jazlyn Casper",
                "id": 39,
                "code": "f9d5ae66-6c1d-361b-a22d-28ed4bc8a10e",
                "options": [],
                "averageRating": 0,
                "images": [
                    {
                        "id": 77,
                        "type": "main"
                    },
                    {
                        "id": 78,
                        "type": "thumbnail"
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/f9d5ae66-6c1d-361b-a22d-
→28ed4bc8a10e"
                    }
                }
            },
            {
                "name": "Mug \"veniam\"",
                "id": 5,
                "code": "f64f7c29-1128-3d12-93d1-19932345b83d",
                "options": [
                    {
                        "id": 1,
                        "code": "mug_type",
                        "position": 0,
                        "values": [
                            {
                                "code": "mug_type_medium",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 1,
```

```json
                                "value": "Medium mug"
                            }
                        }
                    },
                    {
                        "code": "mug_type_double",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 2,
                                "value": "Double mug"
                            }
                        }
                    },
                    {
                        "code": "mug_type_monster",
                        "translations": {
                            "en_US": {
                                "locale": "en_US",
                                "id": 3,
                                "value": "Monster mug"
                            }
                        }
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/mug_type"
                    }
                }
            }
        ],
        "averageRating": 0,
        "images": [
            {
                "id": 9,
                "type": "main"
            },
            {
                "id": 10,
                "type": "thumbnail"
            }
        ],
        "_links": {
            "self": {
                "href": "\/api\/v1\/products\/f64f7c29-1128-3d12-93d1-
↪19932345b83d"
            }
        }
    },
    {
        "name": "Sticker \"animi\"",
        "id": 22,
        "code": "e77f129f-5921-3ad2-88bd-f27b59aad037",
        "options": [
            {
                "id": 2,
```

```json
                    "code": "sticker_size",
                    "position": 1,
                    "values": [
                        {
                            "code": "sticker_size-3",
                            "translations": {
                                "en_US": {
                                    "locale": "en_US",
                                    "id": 4,
                                    "value": "3\""
                                }
                            }
                        },
                        {
                            "code": "sticker_size_5",
                            "translations": {
                                "en_US": {
                                    "locale": "en_US",
                                    "id": 5,
                                    "value": "5\""
                                }
                            }
                        },
                        {
                            "code": "sticker_size_7",
                            "translations": {
                                "en_US": {
                                    "locale": "en_US",
                                    "id": 6,
                                    "value": "7\""
                                }
                            }
                        }
                    ],
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/products\/sticker_size"
                        }
                    }
                }
            ],
            "averageRating": 0,
            "images": [
                {
                    "id": 43,
                    "type": "main"
                },
                {
                    "id": 44,
                    "type": "thumbnail"
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/e77f129f-5921-3ad2-88bd-
↪f27b59aad037"
                }
```

```
            }
        }
    ]
  }
}
```

How to filter resources?

Let's assume that you want to find all products which contain the word `sticker` in the name. In this case you should call the `/api/v1/products/` endpoint with the `GET` method and provide filter query parameters.

Definition

```
GET /api/v1/products/?criteria\[{nameOfCriterion}\]\[type\]={searchOption}&criteria\[
→{nameOfCriterion}\]\[value\]={searchPhrase}'
```

Parameter	Parameter type	Description
Authorization	header	Token received during authentication
nameOfCriterion	query	*(required)* The name of criterion (for example "search", "couponBased")
searchPhrase	query	*(required)* The searching phrase
searchOption	query	*(required)* Option of searching (for example "contains", "equal")
limit	query	*(optional)* Number of items to display per page, by default = 10

Note: The *nameOfCriterion* is a key from the grid configuration of a sample resource.

Tip: You can find a list of all search options in *GridBundle docs*.

Example

```
$  curl 'http://demo.sylius.com/api/v1/products/?criteria\[search\]\[type\]=contains&
→criteria\[search\]\[value\]=sticker&limit=4' \
      -H "Authorization: Bearer SampleToken"
```

Exemplary response

```
STATUS: 200 OK
```

```
{
    "page": 1,
    "limit": 4,
    "pages": 15,
    "total": 60,
    "_links": {
```

```
        "self": {
            "href": "\/api\/v1\/products\/?criteria%5C%5Bsearch%5C%5D=sticker&page=1&
↪limit=4"
        },
        "first": {
            "href": "\/api\/v1\/products\/?criteria%5C%5Bsearch%5C%5D=sticker&page=1&
↪limit=4"
        },
        "last": {
            "href": "\/api\/v1\/products\/?criteria%5C%5Bsearch%5C%5D=sticker&page=15&
↪limit=4"
        },
        "next": {
            "href": "\/api\/v1\/products\/?criteria%5C%5Bsearch%5C%5D=sticker&page=2&
↪limit=4"
        }
    },
    "_embedded": {
        "items": [
            {
                "name": "Book \"voluptates\" by Eveline Waters",
                "id": 35,
                "code": "00ebc508-48f5-326e-8f71-81e4feb0da73",
                "options": [],
                "averageRating": 0,
                "images": [
                    {
                        "id": 69,
                        "type": "main"
                    },
                    {
                        "id": 70,
                        "type": "thumbnail"
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/00ebc508-48f5-326e-8f71-
↪81e4feb0da73"
                    }
                }
            },
            {
                "name": "Mug \"voluptatibus\"",
                "id": 7,
                "code": "0bd9c774-d659-37b7-a22e-44615c155633",
                "options": [
                    {
                        "id": 1,
                        "code": "mug_type",
                        "position": 0,
                        "values": [
                            {
                                "code": "mug_type_medium",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
```

```
                                        "id": 1,
                                        "value": "Medium mug"
                                    }
                                }
                            },
                            {
                                "code": "mug_type_double",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 2,
                                        "value": "Double mug"
                                    }
                                }
                            },
                            {
                                "code": "mug_type_monster",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 3,
                                        "value": "Monster mug"
                                    }
                                }
                            }
                        ],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/products\/mug_type"
                            }
                        }
                    }
                ],
                "averageRating": 0,
                "images": [
                    {
                        "id": 13,
                        "type": "main"
                    },
                    {
                        "id": 14,
                        "type": "thumbnail"
                    }
                ],
                "_links": {
                    "self": {
                        "href": "\/api\/v1\/products\/0bd9c774-d659-37b7-a22e-
→44615c155633"
                    }
                }
            },
            {
                "name": "Mug \"neque\"",
                "id": 10,
                "code": "13ad9ca9-8948-371b-b5b6-d2d988748b07",
                "options": [
                    {
```

```
                            "id": 1,
                            "code": "mug_type",
                            "position": 0,
                            "values": [
                                {
                                    "code": "mug_type_medium",
                                    "translations": {
                                        "en_US": {
                                            "locale": "en_US",
                                            "id": 1,
                                            "value": "Medium mug"
                                        }
                                    }
                                },
                                {
                                    "code": "mug_type_double",
                                    "translations": {
                                        "en_US": {
                                            "locale": "en_US",
                                            "id": 2,
                                            "value": "Double mug"
                                        }
                                    }
                                },
                                {
                                    "code": "mug_type_monster",
                                    "translations": {
                                        "en_US": {
                                            "locale": "en_US",
                                            "id": 3,
                                            "value": "Monster mug"
                                        }
                                    }
                                }
                            ],
                            "_links": {
                                "self": {
                                    "href": "\/api\/v1\/products\/mug_type"
                                }
                            }
                        }
                    ],
                    "averageRating": 0,
                    "images": [
                        {
                            "id": 19,
                            "type": "main"
                        },
                        {
                            "id": 20,
                            "type": "thumbnail"
                        }
                    ],
                    "_links": {
                        "self": {
                            "href": "\/api\/v1\/products\/13ad9ca9-8948-371b-b5b6-
→d2d988748b07"
```

```
                    }
                }
            },
            {
                "name": "T-Shirt \"a\"",
                "id": 56,
                "code": "1823af3c-184a-359d-9c05-6417c7e6abe0",
                "options": [
                    {
                        "id": 3,
                        "code": "t_shirt_color",
                        "position": 2,
                        "values": [
                            {
                                "code": "t_shirt_color_red",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 7,
                                        "value": "Red"
                                    }
                                }
                            },
                            {
                                "code": "t_shirt_color_black",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 8,
                                        "value": "Black"
                                    }
                                }
                            },
                            {
                                "code": "t_shirt_color_white",
                                "translations": {
                                    "en_US": {
                                        "locale": "en_US",
                                        "id": 9,
                                        "value": "White"
                                    }
                                }
                            }
                        ],
                        "_links": {
                            "self": {
                                "href": "\/api\/v1\/products\/t_shirt_color"
                            }
                        }
                    },
                    {
                        "id": 4,
                        "code": "t_shirt_size",
                        "position": 3,
                        "values": [
                            {
                                "code": "t_shirt_size_s",
```

```
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 10,
                            "value": "S"
                        }
                    }
                },
                {
                    "code": "t_shirt_size_m",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 11,
                            "value": "M"
                        }
                    }
                },
                {
                    "code": "t_shirt_size_l",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 12,
                            "value": "L"
                        }
                    }
                },
                {
                    "code": "t_shirt_size_xl",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 13,
                            "value": "XL"
                        }
                    }
                },
                {
                    "code": "t_shirt_size_xxl",
                    "translations": {
                        "en_US": {
                            "locale": "en_US",
                            "id": 14,
                            "value": "XXL"
                        }
                    }
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/t_shirt_size"
                }
            }
        }
    ],
    "averageRating": 3,
```

```json
            "images": [
                {
                    "id": 111,
                    "type": "main"
                },
                {
                    "id": 112,
                    "type": "thumbnail"
                }
            ],
            "_links": {
                "self": {
                    "href": "\/api\/v1\/products\/1823af3c-184a-359d-9c05-
↪6417c7e6abe0"
                }
            }
        }
    ]
  }
}
```

- *Introduction to Sylius REST API*

- *Authorization*

- *Admin Users API*

- *Carts API*

- *Channels API*

- *Checkout API*

- *Countries API*

- *Currencies API*

- *Customers API*

- *Exchange Rates API*

- *Locales API*

- *Orders API*

- *Payment Methods API*

- *Payments API*

- *Product Attributes API*

- *Product Options API*

- *Product Reviews API*

- *Product Variants API*

- *Products API*

- *Promotion Coupons API*

- *Promotions API*

- *Provinces API*

- *Zones API*

- *Sorting and filtration*

The BDD Guide

In *the BDD Guide* you will learn how to write clean and reusable features, contexts and pages using Behat.

6.1 The BDD Guide

Behaviour driven development is an approach to software development process that provides software development and management teams with shared tools and a shared process to collaborate on software development. The awesome part of BDD is its ubiquitous language, which is used to describe the software in English-like sentences of domain specific language.

The application's behaviour is described by scenarios, and those scenarios are turned into automated test suites with tools such as Behat.

Sylius behaviours are fully covered with Behat scenarios. There are more than 1200 scenarios in the Sylius suite, and if you want to understand some aspects of Sylius better, or are wondering how to configure something, we strongly recommend reading them. They can be found in the `features/` directory of the Sylius/Sylius repository.

6.1.1 Basic Usage

The best way of understanding how things work in detail is showing and analyzing examples, that is why this section gathers all the knowledge from the previous chapters. Let's assume that we are going to implement the functionality of managing countries in our system. Now let us show you the flow.

Describing features

Let's start with writing our feature file, which will contain answers to the most important questions: Why (benefit, business value), who (actor using the feature) and what (the feature itself). It should also include scenarios, which serve as examples of how things supposed to work. Let's have a look at the `features/addressing/managing_countries/adding_country.feature` file.

Scenario:

```
@managing_countries
Feature: Adding a new country
    In order to sell my goods to different countries
    As an Administrator
    I want to add a new country to the store

    Background:
        Given I am logged in as an administrator

    @ui
    Scenario: Adding country
        When I want to add a new country
        And I choose "United States"
        And I add it
        Then I should be notified that it has been successfully created
        And the country "United States" should appear in the store
```

Pay attention to the form of these sentences. From the developer point of view they are hiding the details of the feature's implementation. Instead of describing "When I click on the select box And I choose United States from the dropdown Then I should see the United States country in the table" - we are using sentences that are less connected with the implementation, but more focused on the effects of our actions. A side effect of such approach is that it results in steps being really generic, therefore if we want to add another way of testing this feature for instance in the domain or api context, it will be extremely easy to apply. We just need to add a different tag (in this case "@domain") and of course implement the proper steps in the domain context of our system. To be more descriptive let's imagine that we want to check if a country is added properly in two ways. First we are checking if the adding works via frontend, so we are implementing steps that are clicking, opening pages, filling fields on forms and similar, but also we want to check this action regardless of the frontend, for that we need the domain, which allows us to perform actions only on objects.

Choosing a correct suite

After we are done with a feature file, we have to create a new suite for it. At the beginning we have decided that it will be a frontend/user interface feature, that is why we are placing it in "etc/behat/suites/ui/addressing/managing_countries.yml".

```
default:
    suites:
        ui_managing_countries:
            contexts_services:
                - sylius.behat.context.hook.doctrine_orm
                # This service is responsible for clearing database before each
↪scenario,
                # so that only data from the current and its background is available.

                - sylius.behat.context.transform.country
                - sylius.behat.context.transform.shared_storage
                # The transformer contexts services are responsible for all the
↪transformations of data in steps:
                # For instance "And the country "France" should appear in the store"
↪transforms "(the country "France")" to a proper Country object, which is from now
↪on available in the scope of the step.

                - sylius.behat.context.setup.geographical
                - sylius.behat.context.setup.security
                # The setup contexts here are preparing the background, adding
↪available countries and users or administrators.
```

```
                # These contexts have steps like "I am logged in as an administrator"␣
↪already implemented.

                # Lights, Camera, Action!
                - sylius.behat.context.ui.admin.managing_countries
                - sylius.behat.context.ui.admin.notification
                # Those contexts are essential here we are placing all action steps␣
↪like "When I choose "France" and I add it Then I should ne notified that...".
            filters:
                tags: "@managing_countries && @ui"
```

A very important thing that is done here is the configuration of tags, from now on Behat will be searching for all your features tagged with `@managing_countries` and your scenarios tagged with `@ui`. Second thing is `contexts_services:` in this section we will be placing all our services with step implementation.

We have mentioned with the generic steps we can easily switch our testing context to @domain. Have a look how it looks:

```
default:
    suites:
        domain_managing_countries:
            contexts_services:
                - sylius.behat.context.hook.doctrine_orm

                - sylius.behat.context.transform.country
                - sylius.behat.context.transform.shared_storage

                - sylius.behat.context.setup.geographical
                - sylius.behat.context.setup.security

                - sylius.behat.context.domain.admin.managing_countries # Domain step␣
↪implementation.
            filters:
                tags: "@managing_countries && @domain"
```

We are almost finished with the suite configuration. Now we need to register our first Behat context as a service, but beforehand we need

Registering Pages

The page object approach allows us to hide all the detailed interaction with ui (html, javascript, css) inside.

We have three kinds of pages:

- Page - First layer of our pages it knows how to interact with DOM objects. It has a method `->getUrl(array $urlParameters)` where you can define a raw url to open it.

- SymfonyPage - This page extends the Page. It has a router injected so that the `->getUrl()` method generates a url from the route name which it gets from the `->getRouteName()` method.

- Base Crud Pages (IndexPage, CreatePage, UpdatePage) - These pages extend SymfonyPage and they are specific to the Sylius resources. They have a resource name injected and therefore they know about the route name.

There are two ways to manipulate UI - by using `->getDocument()` or `->getElement('your_element')`. First method will return a `DocumentElement` which represents an html structure of the currently opened page,

second one is a bit more tricky because it uses the `->getDefinedElements()` method and it will return a `NodeElement` which represents only the restricted html structure.

Usage example of `->getElement('your_element')` and `->getDefinedElements` methods.

```php
<?php

class CreatePage extends SymfonyPage implements CreatePageInterface
{
    // This method returns a simple associative array, where the key is the name of␣
    ↪your element and the value is its locator.
    protected function getDefinedElements(): array
    {
        return array_merge(parent::getDefinedElements(), [
            'provinces' => '#sylius_country_provinces',
        ]);
    }

    // By default it will assume that your locator is css.
    // Example with xpath.
    protected function getDefinedElements(): array
    {
        return array_merge(parent::getDefinedElements(), [
            'provinces' => ['xpath' => '//*[contains(@class, "provinces")]'] // Now␣
    ↪your value is an array where key is your locator type.
        ]);
    }

    // Like that you can easily manipulate your page elements.
    public function addProvince(ProvinceInterface $province)
    {
        $provinceSelectBox = $this->getElement('provinces');

        $provinceSelectBox->selectOption($province->getName());
    }
}
```

Let's get back to our main example and analyze our scenario. We have steps like "When I choose "France" And I add it Then I should be notified that it has been successfully created And the country "France" should appear in the store".

```php
<?php

namespace Sylius\Behat\Page\Admin\Country;

use Sylius\Behat\Page\Admin\Crud\CreatePage as BaseCreatePage;

class CreatePage extends BaseCreatePage implements CreatePageInterface
{
    /**
     * @param string $name
     */
    public function chooseName($name)
    {
        $this->getDocument()->selectFieldOption('Name', $name);
    }

    public function create()
    {
```

(continues on next page)

```php
        $this->getDocument()->pressButton('Create');
    }
}
```

```php
<? php

namespace Sylius\Behat\Page\Admin\Country;

use Sylius\Behat\Page\Admin\Crud\IndexPage as BaseIndexPage;

class IndexPage extends BaseIndexPage implements IndexPageInterface
{
    /**
     * @return bool
     */
    public function isSingleResourceOnPage(array $parameters)
    {
        try {
            $rows = $this->tableAccessor->getRowsWithFields($this->getElement('table
→'), $parameters);
            // Table accessor is a helper service which is responsible for all html
→table operations.

            return 1 === count($rows);
        } catch (ElementNotFoundException $exception) { // Table accessor throws this
→exception when cannot find table element on page.
            return false;
        }
    }
}
```

> **Warning:** There is one small gap in this concept - PageObjects is not a concrete instance of the currently opened page, they only mimic its behaviour (dummy pages). This gap will be more understandable on the below code example.

```php
<?php

// Of course this is only to illustrate this gap.

class HomePage
{
    // In this context on home page sidebar you have for example weather information
→in selected countries.
    public function readWeather()
    {
        return $this->getElement('sidebar')->getText();
    }

    protected function getDefinedElements(): array
    {
        return ['sidebar' => ['css' => '.sidebar']]
    }
```

```php
    protected function getUrl()
    {
        return 'http://your_domain.com';
    }
}

class LeagueIndexPage
{
    // In this context you have for example football match results.
    public function readMatchResults()
    {
        return $this->getElement('sidebar')->getText();
    }

    protected function getDefinedElements(): array
    {
        return ['sidebar' => ['css' => '.sidebar']]
    }

    protected function getUrl()
    {
        return 'http://your_domain.com/leagues/'
    }
}

final class GapContext implements Context
{
    private $homePage;
    private $leagueIndexPage;

    /**
     * @Given I want to be on Homepage
     */
    public function iWantToBeOnHomePage() // After this method call we will be on
→"http://your_domain.com".
    {
        $this->homePage->open(); //When we add @javascript tag we can actually see␣
→this thanks to selenium.
    }

    /**
     * @Then I want to see the sidebar and get information about the weather in France
     */
    public function iWantToReadSideBarOnHomePage($someInformation) // Still "http://
→your_domain.com".
    {
        $someInformation === $this->leagueIndexPage->readMatchResults() // This␣
→returns true, but wait a second we are on home page (dummy pages).

        $someInformation === $this->homePage->readWeather() // This also returns true.
    }
}
```

Registering contexts

As it was shown in the previous section we have registered a lot of contexts, so we will show you only some of the steps implementation.

Scenario:

```
Given I want to add a new country
And I choose "United States"
And I add it
Then I should be notified that it has been successfully created
And the country "United States" should appear in the store
```

Let's start with essential one ManagingCountriesContext

Ui contexts

```php
<?php

namespace Sylius\Behat\Context\Ui\Admin

final class ManagingCountriesContext implements Context
{
    /**
     * @var IndexPageInterface
     */
    private $indexPage;

    /**
     * @var CreatePageInterface
     */
    private $createPage;

    /**
     * @var UpdatePageInterface
     */
    private $updatePage;

    /**
     * @param IndexPageInterface $indexPage
     * @param CreatePageInterface $createPage
     * @param UpdatePageInterface $updatePage
     */
    public function __construct(
        IndexPageInterface $indexPage,
        CreatePageInterface $createPage,
        UpdatePageInterface $updatePage
    ) {
        $this->indexPage = $indexPage;
        $this->createPage = $createPage;
        $this->updatePage = $updatePage;
    }

    /**
     * @Given I want to add a new country
     */
```

```php
    public function iWantToAddNewCountry()
    {
        $this->createPage->open(); // This method will send request.
    }


    /**
     * @When I choose :countryName
     */
    public function iChoose($countryName)
    {
        $this->createPage->chooseName($countryName);
        // Great benefit of using page objects is that we hide html manipulation
→behind a interfaces so we can inject different CreatePage which implements
→CreatePageInterface
        // And have different html elements which allows for example chooseName(
→$countryName).
    }


    /**
     * @When I add it
     */
    public function iAddIt()
    {
        $this->createPage->create();
    }


    /**
     * @Then /^the (country "([^"]+)") should appear in the store$/
     */
    public function countryShouldAppearInTheStore(CountryInterface $country) // This
→step use Country transformer to get Country object.
    {
        $this->indexPage->open();

        //Webmozart assert library.
        Assert::true(
            $this->indexPage->isSingleResourceOnPage(['code' => $country->getCode()]),
            sprintf('Country %s should exist but it does not', $country->getCode())
        );
    }
}
```

```php
<?php

namespace Sylius\Behat\Context\Ui\Admin

final class NotificationContext implements Context
{
    /**
     * @var NotificationCheckerInterface
     */
    private $notificationChecker;
    // This is a helper service which give access to proper notification elements.

    /**
     * @param NotificationCheckerInterface $notificationChecker
```

```
    */
    public function __construct(NotificationCheckerInterface $notificationChecker)
    {
        $this->notificationChecker = $notificationChecker;
    }

    /**
     * @Then I should be notified that it has been successfully created
     */
    public function iShouldBeNotifiedItHasBeenSuccessfullyCreated()
    {
        $this->notificationChecker->checkNotification('has been successfully created.
↪', NotificationType::success());
    }
}
```

Transformer contexts

```php
<?php

namespace Sylius\Behat\Context\Transform;

final class CountryContext implements Context
{
    /**
     * @var CountryNameConverterInterface
     */
    private $countryNameConverter;

    /**
     * @var RepositoryInterface
     */
    private $countryRepository;

    /**
     * @param CountryNameConverterInterface $countryNameConverter
     * @param RepositoryInterface $countryRepository
     */
    public function __construct(
        CountryNameConverterInterface $countryNameConverter,
        RepositoryInterface $countryRepository
    ) {
        $this->countryNameConverter = $countryNameConverter;
        $this->countryRepository = $countryRepository;
    }

    /**
     * @Transform /^country "([^"]+)"$/
     * @Transform /^"([^"]+)" country$/
     */
    public function getCountryByName($countryName) // Thanks to this method we got in
↪our ManagingCountries an Country object.
    {
        $countryCode = $this->countryNameConverter->convertToCode($countryName);
```

```php
        $country = $this->countryRepository->findOneBy(['code' => $countryCode]);

        Assert::notNull(
            $country,
            'Country with name %s does not exist'
        );

        return $country;
    }
}
```

```php
<?php

namespace Sylius\Behat\Context\Ui\Admin;

use Sylius\Behat\Page\Admin\Country\UpdatePageInterface;

final class ManagingCountriesContext implements Context
{
    /**
     * @var UpdatePageInterface
     */
    private $updatePage;

    /**
     * @param UpdatePageInterface $updatePage
     */
    public function __construct(UpdatePageInterface $updatePage)
    {
        $this->updatePage = $updatePage;
    }

    /**
     * @Given /^I want to create a new province in (country "[^"]+")$/
     */
    public function iWantToCreateANewProvinceInCountry(CountryInterface $country)
    {
        $this->updatePage->open(['id' => $country->getId()]);

        $this->updatePage->clickAddProvinceButton();
    }
}
```

```php
<?php

namespace Sylius\Behat\Context\Transform;

final class ShippingMethodContext implements Context
{
    /**
     * @var ShippingMethodRepositoryInterface
     */
    private $shippingMethodRepository;

    /**
     * @param ShippingMethodRepositoryInterface $shippingMethodRepository
```

```php
     */
    public function __construct(ShippingMethodRepositoryInterface
↪$shippingMethodRepository)
    {
        $this->shippingMethodRepository = $shippingMethodRepository;
    }

    /**
     * @Transform :shippingMethod
     */
    public function getShippingMethodByName($shippingMethodName)
    {
        $shippingMethod = $this->shippingMethodRepository->findOneByName(
↪$shippingMethodName);
        if (null === $shippingMethod) {
            throw new \Exception('Shipping method with name "'.$shippingMethodName.'"␣
↪does not exist');
        }

        return $shippingMethod;
    }
}
```

```php
<?php

namespace Sylius\Behat\Context\Ui\Admin;

use Sylius\Behat\Page\Admin\ShippingMethod\UpdatePageInterface;

final class ShippingMethodContext implements Context
{
    /**
     * @var UpdatePageInterface
     */
    private $updatePage;

    /**
     * @param UpdatePageInterface $updatePage
     */
    public function __construct(UpdatePageInterface $updatePage)
    {
        $this->updatePage = $updatePage;
    }

    /**
     * @Given I want to modify a shipping method :shippingMethod
     */
    public function iWantToModifyAShippingMethod(ShippingMethodInterface
↪$shippingMethod)
    {
        $this->updatePage->open(['id' => $shippingMethod->getId()]);
    }
}
```

> **Warning:** Contexts should have single responsibility and this segregation (Setup, Transformer, Ui, etc. . .) is not accidental. We shouldn't create objects in transformer contexts.

Setup contexts

For setup context we need different scenario with more background steps and all preparing scene steps. Editing scenario will be great for this example:

Scenario:

```
Given the store has disabled country "France"
And I want to edit this country
When I enable it
And I save my changes
Then I should be notified that it has been successfully edited
And this country should be enabled
```

```php
<?php

namespace Sylius\Behat\Context\Setup;

final class GeographicalContext implements Context
{
    /**
     * @var SharedStorageInterface
     */
    private $sharedStorage;

    /**
     * @var FactoryInterface
     */
    private $countryFactory;

    /**
     * @var RepositoryInterface
     */
    private $countryRepository;

    /**
     * @var CountryNameConverterInterface
     */
    private $countryNameConverter;

    /**
     * @param SharedStorageInterface $sharedStorage
     * @param FactoryInterface $countryFactory
     * @param RepositoryInterface $countryRepository
     * @param CountryNameConverterInterface $countryNameConverter
     */
    public function __construct(
        SharedStorageInterface $sharedStorage,
        FactoryInterface $countryFactory,
        RepositoryInterface $countryRepository,
        CountryNameConverterInterface $countryNameConverter
    ) {
```

```php
        $this->sharedStorage = $sharedStorage;
        $this->countryFactory = $countryFactory;
        $this->countryRepository = $countryRepository;
        $this->countryNameConverter = $countryNameConverter;
    }

    /**
     * @Given /^the store has disabled country "([^"]*)"$/
     */
    public function theStoreHasDisabledCountry($countryName) // This method save
↪country in data base.
    {
        $country = $this->createCountryNamed(trim($countryName));
        $country->disable();

        $this->sharedStorage->set('country', $country);
        // Shared storage is an helper service for transferring objects between steps.
        // There is also SharedStorageContext which use this helper service to
↪transform sentences like "(this country), (it), (its), (theirs)" into Country
↪Object.

        $this->countryRepository->add($country);
    }

    /**
     * @param string $name
     *
     * @return CountryInterface
     */
    private function createCountryNamed($name)
    {
        /** @var CountryInterface $country */
        $country = $this->countryFactory->createNew();
        $country->setCode($this->countryNameConverter->convertToCode($name));

        return $country;
    }
}
```

6.1.2 Introduction to containers

In order to provide support for defining contexts and pages in Behat container with dependencies from Symfony application container, our service definitions may contain some extra features.

There are 3 available containers:

- `behat` (the default one) - the container which holds all Behat services, its extensions services, our contexts, pages and helper services used by them

- `symfony` - the container which holds all the services defined in the application, the services retrieved from this container are isolated between scenarios

- `symfony_shared` - the container which holds all the services defined in the application, created only once, the services retrieved from this container are not isolated between scenarios

Right now, you can only inject services from foreign containers into the default containers. To do so, prepend **service id** with `__CONTAINERNAME__.`:

```
<service id="service.id" class="Class">
    <argument type="service" id="behat.service.id" />
    <argument type="service" id="__behat__.another.behat.service.id" />
    <argument type="service" id="__symfony__.symfony.service.id" />
    <argument type="service" id="__symfony_shared__.shared.symfony.service.id" />
</service>
```

6.1.3 How to add a new context?

To add a new context to Behat container it is needed to add a service in to one of a following files `cli.xml/domain.xml/hook.xml/setup.xml/transform.xml/ui.xml` in `src/Sylius/Behat/Resources/config/services/contexts/` folder:

```
<service id="sylius.behat.context.CONTEXT_CATEGORY.CONTEXT_NAME" class="%sylius.behat.context.CONTEXT_CATEGORY.CONTEXT_NAME.class%">
    <tag name="fob.context_service" />
</service>
```

Then you can use it in your suite configuration:

```
default:
    suites:
        SUITE_NAME:
            contexts_services:
                - "sylius.behat.context.CONTEXT_CATEGORY.CONTEXT_NAME"

            filters:
                tags: "@SUITE_TAG"
```

Note: The context categories are usually one of `hook`, `setup`, `ui` and `domain` and, as you can guess, they are corresponded to files name mentioned above.

6.1.4 How to add a new page object?

Sylius uses a solution inspired by `SensioLabs/PageObjectExtension`, which provides an infrastructure to create pages that encapsulates all the user interface manipulation in page objects.

To create a new page object it is needed to add a service in Behat container in `etc/behat/services/pages.xml` file:

```
<service id="sylius.behat.page.PAGE_NAME" class="%sylius.behat.page.PAGE_NAME.class%" parent="sylius.behat.symfony_page" public="false" />
```

Note: There are some boilerplates for common pages, which you may use. The available parents are `sylius.behat.page` (`FriendsOfBehat\PageObjectExtension\Page\Page`) and `sylius.behat.symfony_page` (`FriendsOfBehat\PageObjectExtension\Page\SymfonyPage`). It is not required for a page to extend any class as pages are POPOs (Plain Old PHP Objects).

Then you will need to add that service as a regular argument in context service.

The simplest Symfony-based page looks like:

```
use FriendsOfBehat\PageObjectExtension\Page\SymfonyPage;

class LoginPage extends SymfonyPage
{
    public function getRouteName(): string
    {
        return 'sylius_user_security_login';
    }
}
```

6.1.5 How to define a new suite?

To define a new suite it is needed to create a suite configuration file in a one of `cli`/`domain`/`ui` directory inside `src/Sylius/Behat/Resources/config/suites`. Then register that file in `src/Sylius/Behat/Resources/config/suites.yml`.

6.1.6 How to use transformers?

Behat provides many awesome features, and one of them is definitely **transformers**. They can be used to transform (usually widely used) parts of steps and return some values from them, to prevent unnecessary duplication in many steps' definitions.

Basic transformer

Example is always the best way to clarify, so let's look at this:

```
/**
 * @Transform /^"([^"]+)" shipping method$/
 * @Transform /^shipping method "([^"]+)"$/
 * @Transform :shippingMethod
 */
public function getShippingMethodByName($shippingMethodName)
{
    $shippingMethod = $this->shippingMethodRepository->findOneByName(
↪$shippingMethodName);

    Assert::notNull(
        $shippingMethod,
        sprintf('Shipping method with name "%s" does not exist', $shippingMethodName)
    );

    return $shippingMethod;
}
```

This transformer is used to return `ShippingMethod` object from proper repository using it's name. It also throws exception if such a method does not exist. It can be used in plenty of steps, that have shipping method name in it.

Note: In the example above a Webmozart assertion library was used, to assert a value and throw an exception if needed.

But how to use it? It is as simple as that:

```
/**
 * @Given /^(shipping method "[^"]+") belongs to ("[^"]+" tax category)$/
 */
public function shippingMethodBelongsToTaxCategory(
    ShippingMethodInterface $shippingMethod,
    TaxCategoryInterface $taxCategory
) {
    // some logic here
}
```

If part of step matches transformer definition, it should be surrounded by parenthesis to be handled as whole expression. That's it! As it is shown in the example, many transformers can be used in the same step definition. Is it all? No! The following example will also work like charm:

```
/**
 * @When I delete shipping method :shippingMethod
 * @When I try to delete shipping method :shippingMethod
 */
public function iDeleteShippingMethod(ShippingMethodInterface $shippingMethod)
{
    // some logic here
}
```

It is worth to mention, that in such a case, transformer would be matched depending on a name after ':' sign. So many transformes could be used when using this signature also. This style gives an opportunity to write simple steps with transformers, without any regex, which would boost context readability.

Note: Transformer definition does not have to be implemented in the same context, where it is used. It allows to share them between many different contexts.

Transformers implemented in Sylius

Specified

There are plenty of transformers already implemented in *Sylius*. Most of them return specific resources from their repository, for example:

- `tax category "Fruits"` -> find tax category in their repository with name "Fruits"
- `"Chinese banana" variant of product "Banana"` -> find variant of specific product

etc. You're free to use them in your own behat scenarios.

Note: All transformers definitions are currently kept in `Sylius\Behat\Context\Transform` namespace.

> **Warning:** Remember to include contexts with transformers in custom suite to be able to use them!

Generic

Moreover, there are also some more generic transformers, that could be useful in many different cases. They are now placed in two contexts: `LexicalContext` and `SharedStorageContext`. Why are they so awesome? Let's describe them one by one:

LexicalContext

- `@Transform /^"(?:€|£|\$)((?:\d+\.)?\d+)"$/` -> tricky transformer used to parse price string with currency into integer (used to represent price in *Sylius*). It is used in steps like `this promotion gives "€30.00" fixed discount to every order`

- `@Transform /^"((?:\d+\.)?\d+)%"$/` -> similar one, transforming percentage string into float (example: `this promotion gives "10%" percentage discount to every order`)

SharedStorageContext

Note: `SharedStorage` is kind of container used to keep objects, which can be shared between steps. It can be used, for example, to keep newly created promotion, to use its name in checking existence step.

- `@Transform /^(it|its|theirs)$/` -> amazingly useful transformer, that returns last resource saved in `SharedStorage`. It allows to simplify many steps used after creation/update (and so on) actions. Example: instead of writing `When I create "Wade Wilson" customer`/`Then customer "Wade Wilson" should be registered` just write `When I create "Wade Wilson" customer`/`Then it should be registered`

- `@Transform /^(?:this|that|the) ([^"]+)$/` -> similar to previous one, but returns resource saved with specific key, for example `this promotion` will return resource saved with `promotion` key in `SharedStorage`

The Contribution Guide

The Contribution Guide to Sylius.

7.1 The Contribution Guide

Note: This section is based on the great Symfony documentation.

7.1.1 Install to Contribute

Before you can start contributing to Sylius code or documentation, you should install Sylius locally.

To install Sylius main application from our main repository and contribute, run the following command:

```
$ composer create-project sylius/sylius
```

This will create a new sylius project in the `sylius` directory. When all the dependencies are installed, you'll be asked to fill the `parameters.yml` file via an interactive script. Please follow the steps. If you hit enter, the default values will be loaded.

```
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_driver (pdo_mysql): # - provide a database driver that you are willing to use
database_host (127.0.0.1):
database_port (null):
database_name (sylius): # - you should rename the database to for instance `my_custom_
↪application_name`
database_user (root): # - provide the database user and password
database_password (null): 1234
mailer_transport (smtp): # - if you will be testing e-mails please provide here your␣
↪test account data, use `gmail` as transport for example.
```

```
mailer_host (127.0.0.1):
mailer_user (null): # - your test email
mailer_password (null): # - and password
secret (EDITME):
```

After everything is in place, run the following commands:

```
$ cd sylius # Move to the newly created directory
$ php bin/console sylius:install
```

The `sylius:install` command actually runs several other commands, which will ask you some questions and check if everything is setup to run Sylius properly.

This package contains our main Sylius development repository, with all the components and bundles in the `src/` folder.

In order to see a fully functional frontend you will need to install its assets.

Sylius already has a `gulpfile.babel.js`, therefore you just need to get Gulp using Node.js.

Having Node.js installed go to your project directory and run:

```
$ yarn install
```

And now you can use gulp for installing views, by just running a simple command:

```
$ yarn build
```

For the contributing process questions, please refer to the Contributing Guide that comes up in the following chapters:

Contributing Code

Backward Compatibility Promise

Sylius follows a versioning strategy called Semantic Versioning. It means that only major releases include BC breaks, whereas minor releases include new features without breaking backwards compatibility.

Since Sylius is based on Symfony, our BC promise extends Symfony's Backward Compatibility Promise with a few new rules and exceptions stated in this document.

Minor and patch releases

Patch releases (such as 1.0.x, 1.1.x, etc.) do not require any additional work apart from cleaning the Symfony cache.

Minor releases (such as 1.1.0, 1.2.0, etc.) require to run database migrations.

Code covered

This BC promise applies to all of Sylius' PHP code except for:

- code tagged with `@internal` tags
- event listeners
- model and repository interfaces

- PHPUnit tests (located at `tests/`, `src/**/Tests/`)

- PHPSpec tests (located at `src/**/spec/`)

- Behat tests (located at `src/Sylius/Behat/`)

Additional rules

Models & model interfaces

In order to fulfill the constant Sylius' need to evolve, model interfaces are excluded from this BC promise. Methods may be added to the interface, but backwards compatibility is promised as long as your custom model extends the one from Sylius, which is true for most cases.

Repositories & repository interfaces

Following the reasoning same as above and due to technological constraints, repository interfaces are also excluded from this BC promise.

Event listeners

They are excluded from this BC promise, but they should be as simple as possible and always call another service. Behaviour they're providing (the end result) is still included in BC promise.

Routing

The currently present routes cannot have their name changed, but optional parameters might be added to them. All the new routes will start with `sylius_` prefix in order to avoid conflicts.

Services

Services names cannot change, but new services might be added with `sylius.` prefix.

Templates

Neither template events, block or templates themselves cannot be deleted or renamed.

Reporting a Bug

Whenever you find a bug in Sylius, we kindly ask you to report it. It helps us make a better e-commerce solution for PHP.

> **Caution:** If you think you've found a security issue, please use the special *procedure* instead.

Before submitting a bug:

- Double-check the official documentation to see if you're not misusing the framework;

- Ask for assistance on stackoverflow.com or on the #sylius IRC channel if you're not sure your issue is really a bug.

If your problem definitely looks like a bug, report it using the official bug tracker and follow some basic rules:

- Use the title field to clearly describe the issue;

- Describe the steps needed to reproduce the bug with short code examples (providing a Behat scenario that illustrates the bug is best);

- Give as much detail as possible about your environment (OS, PHP version, Symfony version, Sylius version, enabled extensions, . . .);

- *(optional)* Attach a *patch*.

Submitting a Patch

Patches are the best way to provide a bug fix or to propose enhancements to Sylius.

Step 1: Setup your Environment

Install the Software Stack

Before working on Sylius, set a Symfony friendly environment up with the following software:

- Git

- PHP version 7.1 or above

- MySQL

Configure Git

Set your user information up with your real name and a working email address:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@example.com"
```

Tip: If you are new to Git, you are highly recommended to read the excellent and free ProGit book.

Tip: If your IDE creates configuration files inside the directory of the project, you can use global `.gitignore` file (for all projects) or `.git/info/exclude` file (per project) to ignore them. See Github's documentation.

Tip: Windows users: when installing Git, the installer will ask what to do with line endings, and will suggest replacing all LF with CRLF. This is the wrong setting if you wish to contribute to Sylius. Selecting the as-is method is your best choice, as Git will convert your line feeds to the ones in the repository. If you have already installed Git, you can check the value of this setting by typing:

```
$ git config core.autocrlf
```

This will return either "false", "input" or "true"; "true" and "false" being the wrong values. Change it to "input" by typing:

```
$ git config --global core.autocrlf input
```

Replace –global by –local if you want to set it only for the active repository

---

Get the Sylius Source Code

Get the Sylius source code:

- Create a GitHub account and sign in;

- Fork the Sylius repository (click on the "Fork" button);

- After the "forking action" has completed, clone your fork locally (this will create a `Sylius` directory):

```
$ git clone git@github.com:USERNAME/Sylius.git
```

- Add the upstream repository as a remote:

```
$ cd sylius
$ git remote add upstream git://github.com/Sylius/Sylius.git
```

Step 2: Work on your Patch

The License

Before you start, you must know that all patches you are going to submit must be released under the *MIT license*, unless explicitly specified in your commits.

Choose the right Base Branch

Before starting to work on a patch, you must determine on which branch you need to work. It will be:

- `1.0` or `1.1`, if you are fixing a bug for an existing feature or want to make a change that falls into the list of acceptable changes in patch versions

- `master`, if you are adding a new feature.

Note: All bug fixes merged into the `1.0` and `1.1` maintenance branches are also merged into `master` on a regular basis.

Create a Topic Branch

Each time you want to work on a patch for a bug or on an enhancement, create a topic branch, starting from the previously chosen base branch:

```
$ git checkout -b BRANCH_NAME master
```

Tip: Use a descriptive name for your branch (`issue_XXX` where `XXX` is the GitHub issue number is a good convention for bug fixes).

The above checkout command automatically switches the code to the newly created branch (check the branch you are working on with `git branch`).

Work on your Patch

Work on the code as much as you want and commit as much as you want; but keep in mind the following:

- Practice *BDD*, which is the development methodology we use at Sylius;

- Read about the Sylius *conventions* and follow the coding *standards* (use `git diff --check` to check for trailing spaces – also read the tip below);

- Do atomic and logically separate commits (use the power of `git rebase` to have a clean and logical history);

- Squash irrelevant commits that are just about fixing coding standards or fixing typos in your own code;

- Never fix coding standards in some existing code as it makes the code review more difficult (submit CS fixes as a separate patch);

- In addition to this "code" pull request, you must also update the documentation when appropriate. See more in *contributing documentation* section.

- Write good commit messages (see the tip below).

Tip: A good commit message is composed of a summary (the first line), optionally followed by a blank line and a more detailed description. The summary should start with the Component you are working on in square brackets (`[Cart]`, `[Taxation]`,...). Use a verb (`fixed ...`, `added ...`,...) to start the summary and **don't add a period at the end**.

Prepare your Patch for Submission

When your patch is not about a bug fix (when you add a new feature or change an existing one for instance), it must also include the following:

- An explanation of the changes in the relevant `CHANGELOG` file(s) (the `[BC BREAK]` or the `[DEPRECATION]` prefix must be used when relevant);

- An explanation on how to upgrade an existing application in the relevant `UPGRADE` file(s) if the changes break backward compatibility or if you deprecate something that will ultimately break backward compatibility.

Step 3: Submit your Patch

Whenever you feel that your patch is ready for submission, follow the following steps.

Rebase your Patch

Before submitting your patch, update your branch (needed if it takes you a while to finish your changes):

If you are basing on the `master` branch:

```
$ git checkout master
$ git fetch upstream
$ git merge upstream/master
$ git checkout BRANCH_NAME
$ git rebase master
```

If you are basing on the `1.0` branch:

```
$ git checkout 1.0
$ git fetch upstream
$ git merge upstream/1.0
$ git checkout BRANCH_NAME
$ git rebase 1.0
```

When doing the `rebase` command, you might have to fix merge conflicts. `git status` will show you the *unmerged* files. Resolve all the conflicts, then continue the rebase:

```
$ git add ... # add resolved files
$ git rebase --continue
```

Push your branch remotely:

```
$ git push --force-with-lease origin BRANCH_NAME
```

Make a Pull Request

Warning: Please remember that bug fixes must be submitted against the `1.0` branch, but features and deprecations against the `master` branch. Just accordingly to which branch you chose as the base branch before.

You can now make a pull request on the `Sylius/Sylius` GitHub repository.

To ease the core team work, always include the modified components in your pull request message, like in:

```
[Cart] Fixed something
[Taxation] [Addressing] Added something
```

The pull request description must include the following checklist at the top to ensure that contributions may be reviewed without needless feedback loops and that your contributions can be included into Sylius as quickly as possible:

```
| Q               | A
| --------------- | -----
| Branch?         | 1.0, 1.1 or master
| Bug fix?        | no/yes
| New feature?    | no/yes
| BC breaks?      | no/yes
| Deprecations?   | no/yes
| Related tickets | fixes #X, partially #Y, mentioned in #Z
| License         | MIT
```

An example submission could now look as follows:

```
| Q               | A
| --------------- | -----
```

```
| Branch?          | 1.0
| Bug fix?         | yes
| New feature?     | no
| BC breaks?       | no
| Deprecations?    | no
| Related tickets  | fixes #12
| License          | MIT
```

The whole table must be included (do **not** remove lines that you think are not relevant).

Some answers to the questions trigger some more requirements:

- If you answer yes to "Bug fix?", check if the bug is already listed in the Sylius issues and reference it/them in "Related tickets";

- If you answer yes to "New feature?", you should submit a pull request to the documentation;

- If you answer yes to "BC breaks?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;

- If you answer yes to "Deprecations?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;

If some of the previous requirements are not met, create a todo-list and add relevant items:

```
- [ ] Fix the specs as they have not been updated yet
- [ ] Submit changes to the documentation
- [ ] Document the BC breaks
```

If the code is not finished yet because you don't have time to finish it or because you want early feedback on your work, add an item to todo-list:

```
- [ ] Finish the feature
- [ ] Gather feedback for my changes
```

As long as you have items in the todo-list, please prefix the pull request title with "[WIP]".

In the pull request description, give as much details as possible about your changes (don't hesitate to give code examples to illustrate your points). If your pull request is about adding a new feature or modifying an existing one, explain the rationale for the changes. The pull request description helps the code review.

Rework your Patch

Based on the feedback on the pull request, you might need to rework your patch. Before re-submitting the patch, rebase with your base branch (`master` or `1.0`), don't merge; and force the push to the origin:

```
$ git rebase -f upstream/master
$ git push --force-with-lease origin BRANCH_NAME
```

or

```
$ git rebase -f upstream/1.0
$ git push --force-with-lease origin BRANCH_NAME
```

Note: When doing a `push --force-with-lease`, always specify the branch name explicitly to avoid messing other branches in the repo (`--force-with-lease` tells Git that you really want to mess with things so do it

carefully).

Often, Sylius team members will ask you to "squash" your commits. This means you will convert many commits to one commit. To do this, use the rebase command:

```
$ git rebase -i upstream/master
$ git push --force-with-lease origin BRANCH_NAME
```

or

```
$ git rebase -i upstream/1.0
$ git push --force-with-lease origin BRANCH_NAME
```

After you type this command, an editor will popup showing a list of commits:

```
pick 1a31be6 first commit
pick 7fc64b4 second commit
pick 7d33018 third commit
```

To squash all commits into the first one, remove the word `pick` before the second and the last commits, and replace it by the word `squash` or just `s`. When you save, Git will start rebasing, and if successful, will ask you to edit the commit message, which by default is a listing of the commit messages of all the commits. When you are finished, execute the push command.

Security Issues

This document explains how Sylius issues are handled by the Sylius core team.

Reporting a Security Issue

If you think that you have found a security issue in Sylius, don't use the bug tracker and do not post it publicly. Instead, all security issues must be sent to **security [at] sylius.com**. Emails sent to this address are forwarded to the Sylius core team members.

Resolving Process

For each report, we first try to confirm the vulnerability. When it is confirmed, the team works on a solution following these steps:

1. Send an acknowledgement to the reporter;

2. Work on a patch;

3. Write a security announcement for the official Sylius blog about the vulnerability. This post should contain the following information:

   - a title that always include the "Security release" string;

   - a description of the vulnerability;

   - the affected versions;

   - the possible exploits;

   - how to patch/upgrade/workaround affected applications;

- credits.

4. Send the patch and the announcement to the reporter for review;

5. Apply the patch to all maintained versions of Sylius;

6. Publish the post on the official Sylius blog;

7. Update the security advisory list (see below).

Note: Releases that include security issues should not be done on Saturday or Sunday, except if the vulnerability has been publicly posted.

Note: While we are working on a patch, please do not reveal the issue publicly.

BDD Methodology

Note: This part of documentation is inspired by the official *PHPSpec* docs.

Sylius adopted the full-stack BDD methodology for its development processes.

According to Wikipedia:

> "BDD is a software development process based on test-driven development (TDD). Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software developers and business analysts with shared tools and a shared process to collaborate on software development, with the aim of delivering software that matters."

Setting up Behat & PHPSpec

To run the entire suite of features and specs, including the ones that depend on external dependencies, Sylius needs to be able to autoload them. By default, they are autoloaded from `vendor/` under the main root directory (see `autoload.php.dist`).

To install them all, use Composer:

Step 1: Get Composer

```
$ curl -s http://getcomposer.org/installer | php
```

Make sure you download `composer.phar` in the same folder where the `composer.json` file is located.

Step 2: Install vendors

```
$ php composer.phar install
```

Note: Note that the script takes some time (several minutes) to finish.

Note: If you don't have `curl` installed, you can also just download the `installer` file manually at http://getcomposer.org/installer. Place this file into your project and then run:

```
$ php installer
$ php composer.phar install
```

Install Selenium2

Download Selenium server 3.4 here.

Create a VirtualHost

Add this VirtualHost configuration:

```
<VirtualHost *:80>
    ServerName sylius-test.local

    RewriteEngine On

    DocumentRoot /var/www/sylius/web
    <Directory /var/www/sylius/web>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule ^(.*) %{DOCUMENT_ROOT}/app_test.php [QSA,L]

    ErrorLog ${APACHE_LOG_DIR}/sylius-test-error.log

    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/sylius-test-access.log combined

</VirtualHost>
```

Update your `/etc/hosts` file to include the VirtualHost hostname:

```
127.0.0.1   sylius-test.local
```

Additionally, copy `behat.yml.dist` to `behat.yml`, edit base_url parameter to match your host:

```
default:
    ...
    extensions:
        Behat\MinkExtension:
            ...
            base_url: http://sylius-test.local/app_test.php/
```

Behat

We use *Behat* for StoryBDD and you should always write new scenarios when adding a feature, or update existing stories to adapt Sylius to business requirements changes.

Sylius is an open source project, so the **client** is not clearly visible at first look. But they are here - the Sylius users. We have our needs and Behat helps us understand and satisfy these needs.

Note: To be written.

You can launch Selenium by issuing the following command:

```
$ java -jar selenium-server-standalone-2.38.0.jar
```

Configure behat for Selenium:

```
default:
    ...
    extensions:
        Behat\MinkExtension:
            default_session: selenium2
            browser_name: firefox
            base_url: http://sylius-test.local/app_test.php
            selenium2:
                capabilities: { "browser": "firefox", "version": "28"}
```

Run your scenario using the `behat` console:

```
$ bin/behat
```

PHPSpec

PHPSpec is a PHP toolset to drive emergent design by specification. It is not really a testing tool, but a design instrument, which helps structuring the objects and how they work together.

Sylius approach is to always describe the behavior of the next object you are about to implement.

As an example, we'll write a service, which updates product prices based on an external API. To initialize a new spec, use the `desc` command.

We just need to tell **PHPSpec** we will be working on the *PriceUpdater* class.

```
$ bin/phpspec desc "Sylius\Bundle\CoreBundle\Pricing\PriceUpdater"
Specification for PriceUpdater created in spec.
```

What have we just done? **PHPSpec** has created the spec for us. You can navigate to the spec folder and see the spec there:

```php
<?php

namespace spec\Sylius\Bundle\CoreBundle\Pricing;

use PhpSpec\ObjectBehavior;
use Prophecy\Argument;
```

(continues on next page)

```php
class PriceUpdaterSpec extends ObjectBehavior
{
    function it_is_initializable()
    {
        $this->shouldHaveType('Sylius\Bundle\CoreBundle\Pricing\PriceUpdater');
    }
}
```

The object behavior is made of examples. Examples are encased in public methods, started with it_. or its_.

PHPSpec searches for such methods in your specification to run. Why underscores for example names? just_because_its_much_easier_to_read than someLongCamelCasingLikeThat.

Now, let's write first example which will update the products price:

```php
<?php

namespace spec\Sylius\Bundle\CoreBundle\Pricing;

use Acme\ApiClient;
use PhpSpec\ObjectBehavior;
use Prophecy\Argument;
use Sylius\Bundle\CoreBundle\Model\ProductInterface;

class PriceUpdaterSpec extends ObjectBehavior
{
    function let(ApiClient $api)
    {
        $this->beConstructedWith($api);
    }

    function it_updates_product_price_through_api($api, ProductInterface $product)
    {
        $product->getSku()->shouldBeCalled()->willReturn('TES-12-A-1090');
        $api->getCurrentProductPrice('TES-12-A-1090')->shouldBeCalled()->
→willReturn(1545);
        $product->setPrice(1545)->shouldBeCalled();

        $this->updatePrice($product);
    }
}
```

The example looks clear and simple, the `PriceUpdater` service should obtain the SKU of the product, call the external API and update products price accordingly.

Try running the example by using the following command:

```
$ bin/phpspec run

> spec\Sylius\Bundle\CoreBundle\Pricing\PriceUpdater

  it updates product price through api
    Class PriceUpdater does not exists.

        Do you want me to create it for you? [Y/n]
```

Once the class is created and you run the command again, PHPSpec will ask if it should create the method as well. Start implementing the very initial version of the price updater.

```php
<?php

namespace Sylius\Bundle\CoreBundle\Pricing;

use Sylius\Bundle\CoreBundle\Model\ProductInterface;
use Acme\ApiClient;

class PriceUpdater
{
    private $api;

    public function __construct(ApiClient $api)
    {
        $this->api = $api;
    }

    public function updatePrice(ProductInterface $product)
    {
        $price = $this->api->getCurrentProductPrice($product->getSku());
        $product->setPrice($price);
    }
}
```

Done! If you run PHPSpec again, you should see the following output:

```
$ bin/phpspec run

> spec\Sylius\Bundle\CoreBundle\Pricing\PriceUpdater

  ✓ it updates product price through api

1 examples (1 passed)
223ms
```

This example is greatly simplified, in order to illustrate how we work. There should be few more examples, which cover errors, API exceptions and other edge-cases.

Few tips & rules to follow when working with PHPSpec & Sylius:

- RED is good, add or fix the code to make it green;

- RED-GREEN-REFACTOR is our rule;

- All specs must pass;

- When writing examples, **describe** the behavior of the object in present tense;

- Omit the `public` keyword;

- Use underscores (_) in the examples;

- Use type hinting to mock and stub classes;

- If your specification is getting too complex, the design is wrong, try decoupling a bit more;

- If you cannot describe something easily, probably you should not be doing it that way;

- shouldBeCalled or willReturn, never together, except for builders;

- Use constants in assumptions but strings in expected results;

Happy coding!

Coding Standards

When contributing code to Sylius, you must follow its coding standards.

Sylius follows the standards defined in the PSR-0, PSR-1 and PSR-2 documents.

Here is a short example containing most features described below:

```php
<?php

/*
 * This file is part of the Sylius package.
 *
 * (c) Paweł Jędrzejewski
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Acme;

/**
 * Coding standards demonstration.
 */
class FooBar
{
    const SOME_CONST = 42;

    private $fooBar;

    /**
     * @param string $dummy Some argument description
     */
    public function __construct($dummy)
    {
        $this->fooBar = $this->transformText($dummy);
    }

    /**
     * @param string $dummy Some argument description
     * @param array  $options
     *
     * @return string|null Transformed input
     *
     * @throws \RuntimeException
     */
    private function transformText($dummy, array $options = array())
    {
        $mergedOptions = array_merge(
            array(
                'some_default'    => 'values',
                'another_default' => 'more values',
            ),
            $options
        );

        if (true === $dummy) {
            return;
```

(continues on next page)

```php
        }

        if ('string' === $dummy) {
            if ('values' === $mergedOptions['some_default']) {
                return substr($dummy, 0, 5);
            }

            return ucwords($dummy);
        }

        throw new \RuntimeException(sprintf('Unrecognized dummy option "%s"',
→$dummy));
    }
}
```

Structure

- Add a single space after each comma delimiter;

- Add a single space around operators (===, &&, ...);

- Add a comma after each array item in a multi-line array, even after the last one;

- Add a blank line before `return` statements, unless the return is alone inside a statement-group (like an `if` statement);

- Use braces to indicate control structure body regardless of the number of statements it contains;

- Define one class per file - this does not apply to private helper classes that are not intended to be instantiated from the outside and thus are not concerned by the PSR-0 standard;

- Declare class properties before methods;

- Declare public methods first, then protected ones and finally private ones;

- Use parentheses when instantiating classes regardless of the number of arguments the constructor has;

- Exception message strings should be concatenated using `sprintf`.

Naming Conventions

- Use camelCase, not underscores, for variable, function and method names, arguments;

- Use underscores for option names and parameter names;

- Use namespaces for all classes;

- Prefix abstract classes with `Abstract`.

- Suffix interfaces with `Interface`;

- Suffix traits with `Trait`;

- Suffix exceptions with `Exception`;

- Use alphanumeric characters and underscores for file names;

- Don't forget to look at the more verbose *Conventions* document for more subjective naming considerations.

Service Naming Conventions

- A service name contains groups, separated by dots;
- All Sylius services use `sylius` as first group;
- Use lowercase letters for service and parameter names;
- A group name uses the underscore notation;
- Each service has a corresponding parameter containing the class name, following the `service_name.class` convention.

Automated Coding Standard Enforcement

You can check your code for Sylius coding standard by running the following command:

```
$ bin/ecs check src tests
```

Some of the violations can be automatically fixed by running the same command with `--fix` suffix like:

```
$ bin/ecs check src tests --fix
```

Note: Most of Sylius coding standard checks are extracted to SyliusLabs/CodingStandard package so that reusing them in your own projects or Sylius plugins is effortless. Too learn about details, take a look at its readme.

Documentation

- Add PHPDoc blocks for all classes, methods, and functions;
- Omit the `@return` tag if the method does not return anything;
- The `@package` and `@subpackage` annotations are not used.

License

- Sylius is released under the MIT license, and the license block has to be present at the top of every PHP file, before the namespace.

Conventions

This document describes coding standards and conventions used in the Sylius codebase to make it more consistent and predictable.

Method Names

When an object has a "main" many relation with related "things" (objects, parameters, . . .), the method names are normalized:

- `get()`

- `set()`

- `has()`

- `all()`

- `replace()`

- `remove()`

- `clear()`

- `isEmpty()`

- `add()`

- `register()`

- `count()`

- `keys()`

The usage of these methods are only allowed when it is clear that there is a main relation:

- a `CookieJar` has many `Cookie` objects;

- a Service `Container` has many services and many parameters (as services is the main relation, the naming convention is used for this relation);

- a Console `Input` has many arguments and many options. There is no "main" relation, and so the naming convention does not apply.

For many relations where the convention does not apply, the following methods must be used instead (where `XXX` is the name of the related thing):

Main Relation	Other Relations
`get()`	`getXXX()`
`set()`	`setXXX()`
n/a	`replaceXXX()`
`has()`	`hasXXX()`
`all()`	`getXXXs()`
`replace()`	`setXXXs()`
`remove()`	`removeXXX()`
`clear()`	`clearXXX()`
`isEmpty()`	`isEmptyXXX()`
`add()`	`addXXX()`
`register()`	`registerXXX()`
`count()`	`countXXX()`
`keys()`	n/a

Note: While "setXXX" and "replaceXXX" are very similar, there is one notable difference: "setXXX" may replace, or add new elements to the relation. "replaceXXX", on the other hand, cannot add new elements. If an unrecognized key is passed to "replaceXXX" it must throw an exception.

Deprecations

> **Warning:** Sylius is the pre-alpha development stage. We release minor version before every larger change, but be prepared for BC breaks to happen until 1.0.0 release.

From time to time, some classes and/or methods are deprecated in the framework; that happens when a feature implementation cannot be changed because of backward compatibility issues, but we still want to propose a "better" alternative. In that case, the old implementation can simply be **deprecated**.

A feature is marked as deprecated by adding a `@deprecated` phpdoc to relevant classes, methods, properties, . . . :

```
/**
 * @deprecated Deprecated since version 1.X, to be removed in 1.Y. Use XXX instead.
 */
```

The deprecation message should indicate the version when the class/method was deprecated, the version when it will be removed, and whenever possible, how the feature was replaced.

A PHP `E_USER_DEPRECATED` error must also be triggered to help people with the migration starting one or two minor versions before the version where the feature will be removed (depending on the criticality of the removal):

```
trigger_error(
    'XXX() is deprecated since version 2.X and will be removed in 2.Y. Use XXX
↪instead.',
    E_USER_DEPRECATED
);
```

Git

This document explains some conventions and specificities in the way we manage the Sylius code with Git.

Pull Requests

Whenever a pull request is merged, all the information contained in the pull request is saved in the repository.

You can easily spot pull request merges as the commit message always follows this pattern:

```
merged branch USER_NAME/BRANCH_NAME (PR #1111)
```

The PR reference allows you to have a look at the original pull request on GitHub: https://github.com/Sylius/Sylius/pull/1111. Often, this can help understand what the changes were about and the reasoning behind the changes.

Sylius License

Sylius is released under the MIT license.

According to Wikipedia:

> "It is a permissive license, meaning that it permits reuse within proprietary software on the condition that the license is distributed with that software. The license is also GPL-compatible, meaning that the GPL permits combination and redistribution with software that uses the MIT License."

The License

Copyright (c) 2011-2017 Paweł Jędrzejewski

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- *Backward Compatibility Promise*
- *Reporting a Bug*
- *Submitting a Patch*
- *Security Issues*
- *BDD Methodology*
- *Coding Standards*
- *Conventions*
- *Git*
- *Sylius License*

Contributing Documentation

Contributing to the Documentation

The documentation is as important as the code. It follows the exact same principles: DRY, tests, ease of maintenance, extensibility, optimization, and refactoring just to name a few. And of course, documentation has bugs, typos, hard to read tutorials, and many more.

Contributing

Before contributing, you need to become familiar with the *markup language* used by the documentation.

The Sylius documentation is hosted on GitHub, in the main repository:

```
https://github.com/Sylius/Sylius
```

If you want to submit a patch, fork the official repository on GitHub and then clone your fork to you local destination:

```
$ git clone git@github.com:YOUR_USERNAME/Sylius.git
```

Under the name `origin` you will have from now on the access to your fork. Add also the main repository as the `upstream` remote.

```
$ git remote add upstream git@github.com:Sylius/Sylius.git
```

Choose the right Base Branch

Before starting to work on a patch, you must determine on which branch you need to work. It will be:

- `1.0` or `1.1`, if you are fixing or adding docs for features that exist in one of those versions,

- `master`, if you are documenting a new feature, that was not in `1.0` nor in `1.1`

Note: All bug fixes merged into the `1.0` and `1.1` maintenance branches are also merged into `master` on a regular basis.

Create a dedicated branch for your changes (for organization):

```
$ git checkout -b docs/improving_foo_and_bar
```

You can now make your changes directly to this branch and commit them. Remember to name your commits descriptively, keep them possibly small, with just unitary changes (such that change something only in one part of the docs, not everywhere).

When you're done, push this branch to *your* GitHub fork and initiate a pull request.

Your pull request will be reviewed, you will be asked to apply fixes if necessary and then it will be merged into the main repository.

Testing Documentation

To test the documentation before a commit:

- Install pip, the Python package manager,

- Download the documentation requirements,

```
$ pip install -r docs/requirements.txt
# This makes sure that the version of Sphinx you'll get is >=1.4.2!
```

- Install Sphinx,

```
$ pip install Sphinx
```

- In the `docs` directory run `sphinx-build -b html . build` and view the generated HTML files in the `build` directory.

Creating a Pull Request

Following the example, the pull request will be from your `improving_foo_and_bar` branch to the `Sylius master` branch by default.

GitHub covers the topic of pull requests in detail.

Note: The Sylius documentation is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported *License*.

> **Warning:** You should always prefix the PR name with a `[Documentation]` tag!

You can prefix the title of your pull request in a few cases:

- `[WIP]` (Work in Progress) is used when you are not yet finished with your pull request, but you would like it to be reviewed. The pull request won't be merged until you say it is ready.

- `[ComponentName]` if you are contributing docs that regard on of *the Sylius Components*.

- `[BundleName]` when you add documentation of *the Sylius Bundles*.

- `[Behat]` if you modify something in the *the BDD guide*.

- `[API]` when you are contributing docs to *the API guide*.

For instance if your pull request is about documentation of some feature of the Resource bundle, but it is still a work in progress it should look like : `[WIP][Documentation][ResourceBundle] Arbitrary feature documentation.`

Documenting new Features or Behavior Changes

If you're documenting a brand new feature or a change that's been made in Sylius, you should precede your description of the change with a `.. versionadded:: 1.X` tag and a short description:

```
.. versionadded:: 1.1
    The ``getProductDiscount`` method was introduced in Sylius 1.1.
```

Standards

All documentation in the Sylius Documentation should follow *the documentation standards*.

Reporting an Issue

The easiest contributions you can make is reporting issues: a typo, a grammar mistake, a bug in a code example, a missing explanation, and so on.

Steps:

- Submit a new issue in the GitHub tracker;

- *(optional)* Submit a patch.

Documentation Format

The Sylius documentation uses the *reStructuredText* as its markup language and *Sphinx* for building the output (HTML, PDF, . . .).

---

reStructuredText

reStructuredText "is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system".

You can learn more about its syntax by reading existing Sylius documents or by reading the reStructuredText Primer on the Sphinx website.

If you are familiar with Markdown, be careful as things are sometimes very similar but different:

- Lists starts at the beginning of a line (no indentation is allowed);
- Inline code blocks use double-ticks (``like this``).

Sphinx

Sphinx is a build system that adds some nice tools to create documentation from the reStructuredText documents. As such, it adds new directives and interpreted text roles to standard reST markup.

Syntax Highlighting

All code examples uses PHP as the default highlighted language. You can change it with the `code-block` directive:

```
.. code-block:: yaml

    { foo: bar, bar: { foo: bar, bar: baz } }
```

If your PHP code begins with `<?php`, then you need to use `html+php` as the highlighted pseudo-language:

```
.. code-block:: html+php

    <?php echo $this->foobar(); ?>
```

Note: A list of supported languages is available on the Pygments website.

Configuration Blocks

Whenever you show a configuration, you must use the `configuration-block` directive to show the configuration in all supported configuration formats (`PHP`, `YAML`, and `XML`)

```
.. configuration-block::

    .. code-block:: yaml

        # Configuration in YAML

    .. code-block:: xml

        <!-- Configuration in XML //-->

    .. code-block:: php

        // Configuration in PHP
```

The previous reST snippet renders as follow:

- *YAML*

```
# Configuration in YAML
```

- *XML*

```
<!-- Configuration in XML //-->
```

- *PHP*

```
// Configuration in PHP
```

The current list of supported formats are the following:

Markup format	Displayed
html	HTML
xml	XML
php	PHP
yaml	YAML
json	JSON
jinja	Twig
html+jinja	Twig
html+php	PHP
ini	INI
php-annotations	Annotations

Adding Links

To add links to other pages in the documents use the following syntax:

```
:doc:`/path/to/page`
```

Using the path and filename of the page without the extension, for example:

```
:doc:`/book/architecture`

:doc:`/components_and_bundles/bundles/SyliusAddressingBundle/installation`
```

The link's text will be the main heading of the document linked to. You can also specify an alternative text for the link:

```
:doc:`Simple CRUD </components_and_bundles/bundles/SyliusResourceBundle/installation>`
```

You can also link to pages outside of the documentation, for instance to the Github.

```
`Github`_ //it is an intext link.
```

At the bottom of the document in which you are using your link add a reference:

```
.. _`Github`: http://www.github.com // with a url to your desired destination.
```

Documentation Standards

In order to help the reader as much as possible and to create code examples that look and feel familiar, you should follow these standards.

Sphinx

- The following characters are chosen for different heading levels: level 1 is =, level 2 –, level 3 ~, level 4 `.` and level 5 `"`;

- Each line should break approximately after the first word that crosses the 72nd character (so most lines end up being 72-78 characters);

- The `::` shorthand is *preferred* over `.. code-block:: php` to begin a PHP code block (read the Sphinx documentation to see when you should use the shorthand);

- Inline hyperlinks are **not** used. Separate the link and their target definition, which you add on the bottom of the page;

- Inline markup should be closed on the same line as the open-string;

Example

```
Example
=======

When you are working on the docs, you should follow the
`Sylius Documentation`_ standards.

Level 2
-------

A PHP example would be::

    echo 'Hello World';

Level 3
~~~~~~~

.. code-block:: php

    echo 'You cannot use the :: shortcut here';

.. _`Sylius Documentation`: http://docs.sylius.com/en/latest/contributing/
→documentation/standards.html
```

Code Examples

- The code follows the *Sylius Coding Standards* as well as the Twig Coding Standards;

- To avoid horizontal scrolling on code blocks, we prefer to break a line correctly if it crosses the 85th character, which in many IDEs is signalised by a vertical line;

- When you fold one or more lines of code, place `...` in a comment at the point of the fold. These comments are:

```
// ... (php),
# ... (yaml/bash),
{# ... #} (twig)
<!-- ... --> (xml/html),
; ... (ini),
... (text)
```

- When you fold a part of a line, e.g. a variable value, put `...` (without comment) at the place of the fold;

- Description of the folded code: (optional) If you fold several lines: the description of the fold can be placed after the `...` If you fold only part of a line: the description can be placed before the line;

- If useful to the reader, a PHP code example should start with the namespace declaration;

- When referencing classes, be sure to show the `use` statements at the top of your code block. You don't need to show *all* `use` statements in every example, just show what is actually being used in the code block;

- If useful, a `codeblock` should begin with a comment containing the filename of the file in the code block. Don't place a blank line after this comment, unless the next line is also a comment;

- You should put a `$` in front of every bash line.

Formats

Configuration examples should show recommended formats using *configuration blocks*. The recommended formats (and their orders) are:

- **Configuration** (including services and routing): YAML

- **Validation**: XML

- **Doctrine Mapping**: XML

Example

```php
// src/Foo/Bar.php
namespace Foo;

use Acme\Demo\Cat;
// ...

class Bar
{
    // ...

    public function foo($bar)
    {
        // set foo with a value of bar
        $foo = ...;

        $cat = new Cat($foo);

        // ... check if $bar has the correct value

        return $cat->baz($bar, ...);
    }
}
```

> **Caution:** In YAML you should put a space after { and before } (e.g. `{ _controller:   ...   }`), but this should not be done in Twig (e.g. `{'hello' :   'value'}`).

Language Standards

- For sections, use the following capitalization rules: Capitalization of the first word, and all other words, except for closed-class words:

    The Vitamins are in my Fresh California Raisins

- Please use appropriate, informative, rather formal language;

- Do not place any kind of advertisements in the documentation;

- The documentation should be neutral, without judgements, opinions. Make sure you do not favor anyone, our community is great as a whole, there is no need to point who is better than the rest of us;

- You should use a form of *you* instead of *we* (i.e. avoid the first person point of view: use the second instead);

- When referencing a hypothetical person, such as "a user with a session cookie", gender-neutral pronouns (they/their/them) should be used. For example, instead of:

    - he or she, use they

    - him or her, use them

    - his or her, use their

    - his or hers, use theirs

    - himself or herself, use themselves

Sylius Documentation License

The Sylius documentation is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.

You are free:

- to *Share* — to copy, distribute and transmit the work;

- to *Remix* — to adapt the work.

Under the following conditions:

- *Attribution* — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work);

- *Share Alike* — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- *Waiver* — Any of the above conditions can be waived if you get permission from the copyright holder;

- *Public Domain* — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license;

- *Other Rights* — In no way are any of the following rights affected by the license:

    - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;

    - The author's moral rights;

  – Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

- *Notice* — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This is a human-readable summary of the Legal Code (the full license).

- *Contributing to the Documentation*
- *Documentation Format*
- *Documentation Standards*
- *Sylius Documentation License*

Contributing Translations

Sylius application is fully translatable, and what is more it is translated by the community to a variety of languages!

Sylius translations are kept on Crowdin.

How to contribute translations in any language?

The process of submitting new translations in any existing language is really simple:

1. First of all you need an account on Crowdin. If do not have one, please sign up.

2. Find a piece of Sylius and translate it to chosen language here.

3. After approval from the Crowdin community it will be automatically synced into the main repository.

That's all! You can start translating.

7.1.2 Contributing Code

- *Backward Compatibility Promise*
- *Reporting a Bug*
- *Submitting a Patch*
- *Security Issues*
- *BDD Methodology*
- *Coding Standards*
- *Conventions*
- *Git*
- *Sylius License*

7.1.3 Contributing Documentation

- *Contributing to the Documentation*
- *Documentation Format*
- *Documentation Standards*

- *Sylius Documentation License*

7.1.4 Contributing Translations

7.1.5 Organization

Sylius is developed by a vibrant community of commercial companies and individual developers. The chapter describes the rules & processes we use to organize our work.

Organization

Vision & Strategy

Vision & strategy is defined by the Project Leader, Core Team and Community members.

If you would like to suggest new tool, process, feel free to submit a PR to this section of the documentation.

GitHub

We use GitHub as the main tool to organize the community work and everything that happens around Sylius. Releases, bug reports, feature requests, roadmap management, etc. happens on this platform.

If you are not sure about your issue, please use Slack to discuss it with the fellow community members before opening it on GitHub.

Milestones

Since stable Sylius release, we use milestones to mark the lowest branch an issue or a PR applies to. For example, if a bug report is marked with 1.0 milestone, the related bugfix PR should be opened against 1.0 branch. Then, after this PR is merged, it would be released in the next 1.0.x release.

Learn more about the *The Release Process*.

Pull Request Checklist

Before any PR is merged, the following things need to be confirmed:

1. Changes can be included in the upcoming release.
2. PR has been approved by at least 1 fellow Core Team member.
3. PR adheres to the PR template and contains the MIT license.
4. PR includes relevant documentation updates.
5. PR contains appropriate UPGRADE file updates if necessary.
6. PR is properly labeled and milestone is assigned if needed.
7. All required checks are passing. It is green!

Certain PRs can only be merged by the Project Lead:

- BC Breaks

- Introducing new components, bundles or high level architecture layers

- Renaming existing components

- If in doubt, ask your friendly neighborhood Project Lead

Sylius Core Team

The **Sylius Core Team** is the group of developers that determine the direction and evolution of the Sylius project. Their votes rule if the features and patches proposed by the community are approved or rejected.

All the Sylius Core Team members are long-time contributors with solid technical expertise and they have demonstrated a strong commitment to drive the project forward.

This document states the rules that govern the Sylius Core Team. These rules are effective upon publication of this document and all Sylius Core Team members must adhere to said rules and protocol.

Core Team Organization

Sylius Core Team members have the following roles:

1. **Project Leader**
- Defines business vision & strategy
- Elects Sylius Core Team Members
2. **Lead Developer**
- Defines technical vision & strategy
- Coordinates community
- Connects with other projects
3. **Documentation Lead**
- Coordinates the work related to the documentation
4. **Core Developer**
- Can review & merge all code PRs and issues
- Focuses on a specific area of the system

Active Core Team Members

- **Project Leader**:
    - **Paweł Jędrzejewski** (pjedrzejewski)
- **Lead Developer**:
    - **Kamil Kokot** (pamil)
- **Documentation Lead**:
    - **Magdalena Banasiak** (CoderMaggie)
- **Core Developers**:
    - **Łukasz Chruściel** (lchrusciel)

> – **Mateusz Zalewski** (Zales0123)
>
> – **Grzegorz Sadowski** (GSadee)

Core Membership Application

At present, new Sylius Core Team membership applications are not accepted, although we are in the process of inviting new members.

Core Membership Revocation

A Sylius Core Team membership can be revoked for any of the following reasons:

- Refusal to follow the rules and policies stated in this document;
- Lack of activity for the past six months;
- Willful negligence or intent to harm the Sylius project;
- Upon decision of the **Project Leader**.

Should new Sylius Core Team memberships be accepted in the future, revoked members must wait at least 6 months before re-applying.

Sylius Core Team Rules and Protocol Amendments

The rules described in this document may be amended at anytime by the **Project Leader**.

The Release Process

This document explains the **release process** of the Sylius project (i.e. the code & documentation hosted on the main `Sylius/Sylius` Git repository).

Sylius manages its releases through a *time-based model* and follows the Semantic Versioning strategy:

- A new Sylius minor version (e.g. 1.1, 1.2, etc.) comes out every *four months*
- A new Sylius patch version (e.g. 1.0.1, 1.0.2, etc.) comes out every *two weeks*

Development

The full development period for any minor version lasts four months and is divided into two phases:

- **Development**: *Three months* to add new features and to enhance existing ones;
- **Stabilization**: *One month* to fix bugs, prepare the release, and wait for the whole Sylius ecosystem (third-party libraries, plugins, and projects using Sylius) to catch up.

During the development period, any new feature can be reverted if it won't be finished in time or if it won't be stable enough to be included in the coming release.

Maintenance

Each Sylius version is maintained for a fixed period of time. This maintenance is divided into:

- *Bug fixes and security fixes*: During this period, being *eight months* long, all issues can be fixed. The end of this period is referenced as being the *end of maintenance* of a release.

- *Security fixes only*: During this period, being *sixteen months* long, only security related issues can be fixed. The end of this period is referenced as being the *end of life* of a release.

Backward Compatibility

All Sylius releases have to comply with our Backward Compatibility Promise.

Whenever keeping backward compatibility is not possible, the feature, the enhancement or the bug fix will be scheduled for the next major version.

- *Vision & Strategy*
- *Sylius Core Team*
- *The Release Process*
- *Vision & Strategy*
- *Sylius Core Team*
- *The Release Process*
- *Contributing Code*
- *Contributing Documentation*
- *Contributing Translations*
- *Organization*

CHAPTER 8

Support

The Support section for Sylius.

8.1 Support

Besides documentation we have a very friendly community which provides support for all Sylius users seeking help!

At Sylius we have 3 main channels for communication and support.

8.1.1 Slack

Most of us use Slack for their online communication with co-workers. We know that it is not convenient to have another chat window open, therefore we've chosen Slack for the live communication with the community. Slack is supposed to handle the most urgent questions in the fastest way possible.

The most important rooms for newcomers are:

#general - for discussions about Sylius development itself, #docs - for discussions related to the documentation, #support - for asking questions and helping others, #random - for the non-work banter and water cooler conversation.

But there are many more specific channels also. If you have any suggestions regarding its organization, please let us know on Slack!

Slack requires inviting new members, but this can be done automatically, just go to sylius.com/slack, enter your email and you will get an invitation. If possible, please use your GitHub username - it will help us to recognize each other easily!

8.1.2 Forum

In response to the rapid growth of our Ecosystem, we decided it is necessary to launch a brand new platform for support and exchanging experience. On the contrary to Slack, our Forum is much easier to follow, on Slack unless you

are a part of the discussion when it is happening, it might be difficult to catch up when you have been offline. Forum has a search engine so it is convenient to browse for what is interesting for you.

The Sylius Community Forum is meant for everyone who is interested in eCommerce technology and Sylius. We invite both existing and new community members to join the discussion and help shape the ecosystem, products & services we are building. Get to know other community members, ask for support, suggest an improvement or discuss your challenge.

You can register via email, Twitter & GitHub by going to https://forum.sylius.com/ and hitting the "Sign-Up" button.

8.1.3 StackOverflow

We also encourage asking Sylius related questions on the stackoverflow.com platform.

- Search for the question before asking, maybe someone has already solved your problem,

- Be specific about your question, this is what SO is about, concrete questions and solutions,

- Be sure to tag them with **sylius** tag - it will make it easier to find for people who can answer it.

- Try also tagging your questions with the **symfony** tag - the Symfony community is very big and might be really helpful with Sylius related questions, as we are basing on Symfony.

To view all Sylius related questions - visit this link. You can also search for phrase.

Components & Bundles

Documentation of all Sylius components and bundles useful when using them standalone.

9.1 Components & Bundles

9.1.1 Sylius Bundles Documentation

SyliusAddressingBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

This bundle integrates the *Addressing* into Symfony and Doctrine.

With minimal configuration you can introduce addresses, countries, provinces and zones management into your project. It's fully customizable, but the default setup should be optimal for most use cases.

It also contains zone matching mechanisms, which allow you to match customer addresses to appropriate tax/shipping (or any other) zones. There are several models inside the bundle, *Address*, *Country*, *Province*, *Zone* and *ZoneMember*.

There is also a **ZoneMatcher** service. You'll get familiar with it in later parts of this documentation.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use following command to add the bundle to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/addressing-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/addressing-bundle:*
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),

        new Sylius\Bundle\AddressingBundle\SyliusAddressingBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

ZoneMatcher

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

This bundle exposes the **ZoneMatcher** as `sylius.zone_matcher` service.

```php
<?php

$zoneMatcher = $this->get('sylius.zone_matcher');

$zone = $zoneMatcher->match($user->getBillingAddress());
```

Forms

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

The bundle ships with a set of useful form types for all models. You can use the defaults or *override them* with your own types.

Address form

The address form type is named `sylius_address` and you can create it whenever you need, using the form factory.

```php
<?php

// src/Acme/ShopBundle/Controller/AddressController.php

namespace Acme\ShopBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DemoController extends Controller
{
    public function fooAction(Request $request)
    {
        $form = $this->get('form.factory')->create('sylius_address');
    }
}
```

You can also embed it into another form.

```php
<?php

// src/Acme/ShopBundle/Form/Type/OrderType.php

namespace Acme\ShopBundle\Form\Type;
```

```php
use Sylius\Bundle\OrderBundle\Form\Type\OrderType as BaseOrderType;
use Symfony\Component\Form\FormBuilderInterface;

class OrderType extends BaseOrderType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        parent::buildForm($builder, $options);

        $builder
            ->add('billingAddress', 'sylius_address')
            ->add('shippingAddress', 'sylius_address')
        ;
    }
}
```

Summary

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration Reference

```yaml
sylius_addressing:
    # The driver used for persistence layer.
    driver: ~
    resources:
        address:
            classes:
                model: Sylius\Component\Addressing\Model\Address
                interface: Sylius\Component\Addressing\Model\AddressInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\AddressType
        country:
            classes:
                model: Sylius\Component\Addressing\Model\Country
                interface: Sylius\Component\Addressing\Model\CountryInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\CountryType
        province:
            classes:
                model: Sylius\Component\Addressing\Model\Province
                interface: Sylius\Component\Addressing\Model\ProvinceInterface
                controller:
→Sylius\Bundle\AddressingBundle\Controller\ProvinceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
```

```
                form: Sylius\Bundle\AddressingBundle\Form\Type\ProvinceType
        zone:
            classes:
                model: Sylius\Component\Addressing\Model\Zone
                interface: Sylius\Component\Addressing\Model\ZoneInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\ZoneType
        zone_member:
            classes:
                model: Sylius\Component\Addressing\Model\ZoneMember
                interface: Sylius\Component\Addressing\Model\ZoneMemberInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\AddressingBundle\Form\Type\ZoneMemberType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Addresses in the Sylius platform* - concept documentation

SyliusAttributeBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

This bundle provides easy integration of the *Sylius Attribute component* with any Symfony full-stack application.

Sylius uses this bundle internally for its product catalog to manage the different attributes that are specific to each product.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/attribute-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/attribute-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Sylius\Bundle\AttributeBundle\SyliusAttributeBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
sylius_attribute:
    driver: doctrine/orm # Configure the doctrine orm driver used in the
→documentation.
```

And configure doctrine extensions which are used by the bundle.

```yaml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

Warning: This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Congratulations! The bundle is now installed and ready to use.

Configuration reference

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

```yaml
sylius_attribute:
    driver: ~ # The driver used for persistence layer. Currently only `doctrine/
↪orm` is supported.
    resources:
        # `subject_name` can be any name, for example `product`, `ad`, or `blog_
↪post`
        subject_name:
            subject: ~ # Required: The subject class implementing
↪`AttributeSubjectInterface`.
            attribute:
                classes:
                    model:      Sylius\Component\Attribute\Model\Attribute
                    interface:  Sylius\Component\Attribute\Model\AttributeInterface
                    repository:
↪Sylius\Bundle\TranslationBundle\Doctrine\ORM\TranslatableResourceRepository
                    controller:
↪Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    factory:    Sylius\Component\Resource\Factory\Factory
                    form: Sylius\Bundle\AttributeBundle\Form\Type\AttributeType
                translation:
                    classes:
                        model:
↪Sylius\Component\Attribute\Model\AttributeTranslation
                        interface:
↪Sylius\Component\Attribute\Model\AttributeTranslationInterface
                        controller:
↪Sylius\Bundle\ResourceBundle\Controller\ResourceController
                        repository: ~ # Required: The repository class for the
↪attribute translation.
                        factory:    Sylius\Component\Resource\Factory\Factory
                        form:
↪Sylius\Bundle\AttributeBundle\Form\Type\AttributeTranslationType
            attribute_value:
                classes:
                    model:      ~ # Required: The model of the attribute value
                    interface:  ~ # Required: The interface of the attribute value
                    controller:
↪Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    repository: ~ # Required: The repository class for the
↪attribute value.
```

(continues on next page)

```
                        factory:    Sylius\Component\Resource\Factory\Factory
                        form: Sylius\Bundle\AttributeBundle\Form\Type\AttributeValueType
```

Learn more

- *Attributes in the Sylius platform* - concept documentation

SyliusCustomerBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

A solution for customer management system inside of a Symfony application.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/customer-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/customer-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new FOS\RestBundle\FOSRestBundle(),
```

```
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Bazinga\Bundle\HateoasBundle\BazingaHateoasBundle(),
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new Sylius\Bundle\CustomerBundle\SyliusCustomerBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Configure Doctrine extensions

Configure doctrine extensions which are used by the bundle.

```
# app/config/config.yml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Congratulations! The bundle is now installed and ready to use.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration reference

```
sylius_customer:
    driver: doctrine/orm
    resources:
        customer:
            classes:
```

```
            model: Sylius\Component\Core\Model\Customer
            repository: Sylius\Bundle\CoreBundle\Doctrine\ORM\CustomerRepository
            form:
                default: Sylius\Bundle\CoreBundle\Form\Type\Customer\CustomerType
                profile:␣
→Sylius\Bundle\CustomerBundle\Form\Type\CustomerProfileType
                choice: Sylius\Bundle\ResourceBundle\Form\Type\ResourceChoiceType
            interface: Sylius\Component\Customer\Model\CustomerInterface
            controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
            factory: Sylius\Component\Resource\Factory\Factory
        customer_group:
            classes:
                model: Sylius\Component\Customer\Model\CustomerGroup
                interface: Sylius\Component\Customer\Model\CustomerGroupInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\CustomerBundle\Form\Type\CustomerGroupType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Customers in the Sylius platform* - concept documentation

SyliusFixturesBundle

Configurable fixtures management for Symfony applications.

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your composer.json and download the package.

If you have Composer installed globally.

```
$ composer require sylius/fixtures-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/fixtures-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel, usually at the end of bundle list.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        // Other bundles...
        new Sylius\Bundle\FixturesBundle\SyliusFixturesBundle(),
    );
}
```

Architecture

Flexibility is one of the key concepts of **SyliusFixturesBundle**. This article aims to explain what design decisions were made in order to achieve it.

Suites

Suites are collections of configured fixtures. They allow you to define different sets (for example - `staging`, `development` or `big_shop`) that can be loaded independently. They are defined through YAML configuration:

```yaml
sylius_fixtures:
    suites:
        my_suite_name: # Suite name as a key
            listeners: ~
            fixtures: ~
```

Fixtures

Fixtures are just plain old PHP objects, that change system state during their execution - they can either persist some entities in the database, upload some files, dispatch some events or do anything you think is needed.

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            fixtures:
                my_fixture: # Fixture name as a key
                    priority: 0 # The higher priority is, the sooner the fixture will
→be executed
                    options: ~ # Fixture options
```

They implement the `Sylius\Bundle\FixturesBundle\Fixture\FixtureInterface` and need to be registered under the `sylius_fixtures.fixture` tag in order to be used in suite configuration.

Note: The former interface extends the `ConfigurationInterface`, which is widely known from `Configuration` classes placed under `DependencyInjection` directory in Symfony bundles.

Using a fixture multiple times in a single suite

In order to use the same fixture multiple times in a single suite, it is needed to alias them:

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            regular_user: # Fixture alias as a key
                name: user # Fixture name
                options:
                    admin: false
                    amount: 10

            admin_user: # Fixture alias as a key
                name: user # Fixture name
                options:
                    admin: true
                    amount: 2
```

Both `regular_user` and `admin_user` are the aliases for `user` fixture. They will run the same fixture, but with different options being submitted.

Listeners

Listeners allow you to execute code at some point of fixtures loading.

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            listeners:
                my_listener: # Listener name as a key
                    priority: 0 # The higher priority is, the sooner the fixture will␣
↪be executed
                    options: ~ # Listener options
```

They implement at least one of four interfaces:

- `Sylius\Bundle\FixturesBundle\Listener\BeforeSuiteListenerInterface` - receives `Sylius\Bundle\FixturesBundle\Listener\SuiteEvent` as an argument

- `Sylius\Bundle\FixturesBundle\Listener\BeforeFixtureListenerInterface` - receives `Sylius\Bundle\FixturesBundle\Listener\FixtureEvent` as an argument

- `Sylius\Bundle\FixturesBundle\Listener\AfterFixtureListenerInterface` - receives `Sylius\Bundle\FixturesBundle\Listener\FixtureEvent` as an argument

- `Sylius\Bundle\FixturesBundle\Listener\AfterSuiteListenerInterface` - receives `Sylius\Bundle\FixturesBundle\Listener\SuiteEvent` as an argument

Note: The former interface extends the `ConfigurationInterface`, which is widely known from `Configuration` classes placed under `DependencyInjection` directory in Symfony bundles.

In order to be used in suite configuration, they need to be registered under the `sylius_fixtures.listener`.

Disabling listeners / fixtures in consecutive configurations

Given the following configuration coming from a third party (like Sylius if you're developing an application based on it):

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            listeners:
                first_listener: ~
                second_listener: ~
            fixtures:
                first_fixture: ~
                second_fixture: ~
```

It is possible to disable a listener or a fixture in a consecutive configuration by providing `false` as its value:

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            listeners:
                second_listener: false
            fixtures:
                second_fixture: false
```

These two configurations combined will be treated as a single configuration like:

```yaml
sylius_fixtures:
    suites:
        my_suite_name:
            listeners:
                first_listener: ~
            fixtures:
                first_fixture: ~
```

Custom fixture

Basic fixture

Let's create a fixture that loads all countries from the `Intl` library. We'll extend the `AbstractFixture` in order to skip the configuration part for now:

```php
namespace AppBundle\Fixture;

use Sylius\Bundle\FixturesBundle\Fixture\AbstractFixture;
use Sylius\Bundle\FixturesBundle\Fixture\FixtureInterface;

final class CountryFixture extends AbstractFixture implements FixtureInterface
{
    private $countryManager;

    public function __construct(ObjectManager $countryManager)
    {
        $this->countryManager = $countryManager;
    }
```

```php
    public function getName()
    {
        return 'country';
    }

    public function load(array $options)
    {
        $countriesCodes = array_keys(\Intl::getRegionBundle()->getCountryNames());

        foreach ($countriesCodes as $countryCode) {
            $country = new Country($countryCode);

            $this->countryManager->persist($country);
        }

        $this->countryManager->flush();
    }
}
```

The next step is to register this fixture:

```xml
<service id="app.fixture.country" class="AppBundle\Fixture\CountryFixture">
    <argument type="service" id="doctrine.orm.entity_manager" />
    <tag name="sylius_fixtures.fixture" />
</service>
```

Fixture is now registered and ready to use in your suite:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            fixtures:
                country: ~
```

Configurable fixture

Loading all countries may be useful, but what if you want to load only some defined countries in one suite and all the countries in the another one? You don't need to create multiple fixtures, a one configurable fixture will do the job!

```php
// ...

final class CountryFixture extends AbstractFixture implements FixtureInterface
{
    // ...

    public function load(array $options)
    {
        foreach ($options['countries'] as $countryCode) {
            $country = new Country($countryCode);

            $this->countryManager->persist($country);
        }

        $this->countryManager->flush();
```

```
    }

    protected function configureOptionsNode(ArrayNodeDefinition $optionsNode)
    {
        $optionsNode
            ->children()
            ->arrayNode('countries')
                ->performNoDeepMerging()
                ->defaultValue(array_keys(\Intl::getRegionBundle()->
↪getCountryNames()))
                ->prototype('scalar')
        ;
    }
}
```

Note: The `AbstractFixture` implements the `ConfigurationInterface::getConfigTreeBuilder()`
and exposes a handy `configureOptionsNode()` method to reduce the boilerplate. It is possible to test this
configuration using SymfonyConfigTest library. For examples of that tests have a look at Sylius Fixtures Configuration
Tests.

Now, it is possible for the fixture to create different outcomes by just changing its configuration:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            fixtures:
                country: ~ # Creates all countries
        my_another_suite:
            fixtures:
                country:
                    options: ~ # Still creates all countries
        my_customized_suite:
            fixtures:
                country:
                    options:
                        countries: # Creates only defined countries
                            - PL
                            - FR
                            - DE
```

Custom listener

Basic listener

Let's create a listener that removes the directory before loading the fixtures.

```php
namespace AppBundle\Listener;

use Sylius\Bundle\FixturesBundle\Listener\AbstractListener;
use Sylius\Bundle\FixturesBundle\Listener\BeforeSuiteListenerInterface;
use Sylius\Bundle\FixturesBundle\Listener\SuiteEvent;
use Symfony\Component\Filesystem\Filesystem;
```

```
final class DirectoryPurgerListener extends AbstractListener implements␣
↪ListenerInterface
{
    public function getName()
    {
        return 'directory_purger';
    }

    public function beforeSuite(SuiteEvent $suiteEvent, array $options)
    {
        (new Filesystem())->remove('/hardcoded/path/to/directory');
    }
}
```

The next step is to register this listener:

```xml
<service id="app.listener.directory_purger" class=
↪"AppBundle\Listener\DirectoryPurgerListener">
    <tag name="sylius_fixtures.listener" />
</service>
```

Listener is now registered and ready to use in your suite:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            listeners:
                directory_purger: ~
```

Configurable listener

Listener that removes a hardcoded directory isn't very useful. Allowing it to receive an array of directories would make this listener a lot more reusable.

```php
// ...

final class DirectoryPurgerListener extends AbstractListener implements␣
↪ListenerInterface
{
    // ...

    public function beforeSuite(SuiteEvent $suiteEvent, array $options)
    {
        (new Filesystem())->remove($options['directories']);
    }

    protected function configureOptionsNode(ArrayNodeDefinition $optionsNode)
    {
        $optionsNodeBuilder
            ->arrayNode('directories')
                ->performNoDeepMerging()
                ->prototype('scalar')
        ;
```

```
      }
}
```

Note: The `AbstractListener` implements the `ConfigurationInterface::getConfigTreeBuilder()` and exposes a handy `configureOptionsNode()` method to reduce the boilerplate. It is possible to test this configuration using [SymfonyConfigTest](#) library.

Now, it is possible to remove different directories in each suite:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            listener:
                directory_purger:
                    options:
                        directories:
                            - /custom/directory
                            - /another/custom/directory
        my_another_suite:
            listener:
                directory_purger:
                    options:
                        directories:
                            - /path/per/suite
```

Built-in listeners

SyliusFixturesBundle comes with a few useful listeners.

Logger (`logger`)

Provides output while running `sylius:fixtures:load` command.

```
# Without logger

$ bin/console sylius:fixtures:load my_suite
$ _

# With logger

$ bin/console sylius:fixtures:load my_suite
Running suite "my_suite"...
Running fixture "country"...
Running fixture "locale"...
Running fixture "currency"...
$ _
```

The logger does not have any configuration options. It can be enabled in such a way:

```yaml
sylius_fixtures:
    suites:
```

```yaml
        my_suite:
            listeners:
                logger: ~
```

ORM Purger (`orm_purger`)

Purges the relational database. Uses `delete` purge mode and the default entity manager if not configured otherwise.

Configuration options:

- `mode` - sets how database is purged, available values: `delete` (default), `truncate`

- `managers` - an array of entity managers' names used to purge the database, `[null]` by default

- `exclude` - an array of table/view names to be excluded from purge, `[]` by default

Example configuration:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            listeners:
                orm_purger:
                    options:
                        mode: truncate
                        managers:
                            - custom_manager
                        exclude:
                            - custom_entity_table_name
```

PHPCR / MongoDB Purger (`phpcr_purger` / `mongodb_purger`)

Purges the document database. Uses the default document manager if not configured otherwise.

Configuration options:

- `managers` - an array of document managers' names used to purge the database, `[null]` by default

Example configuration:

```yaml
sylius_fixtures:
    suites:
        my_suite:
            listeners:
                phpcr_purger:
                    options:
                        managers:
                            - custom_manager # Uses custom document manager
                mongodb_purger: ~ # Uses default document manager
```

Commands

Listing fixtures

To list all available suites and fixtures, use the `sylius:fixtures:list` command.

```
$ bin/console sylius:fixtures:list

Available suites:
 – default
 – dev
 – test
Available fixtures:
 – country
 – locale
 – currency
```

Loading fixtures

To load a suite, use the `sylius:fixtures:load [suite]` command.

```
$ bin/console sylius:fixtures:load default

Running suite "default"...
Running fixture "country"...
Running fixture "locale"...
Running fixture "currency"...
```

Summary

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Fixtures in the Sylius platform* - concept documentation

SyliusGridBundle

Displaying a grid with sorting and filtering options is a common task for many web applications. This bundle integrates the Sylius Grid component with the Symfony framework and allows you to display grids really easily.

Some of the features worth mentioning:

- Uses YAML to define the grid structure
- Supports different data sources: Doctrine ORM/ODM, native SQL query.
- Rich filter functionality, easy to define your own filter type with flexible form
- Each column type is configurable and you can create your own
- Automatic sorting

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/grid-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/grid-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add `SyliusResourceBundle` and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new Sylius\Bundle\GridBundle\SyliusGridBundle(),
    );
}
```

Congratulations! The bundle is now installed and ready to use. You need to define your first resource and grid!

Your First Grid

In order to use grids, we need to register your entity as a Sylius resource. Let us assume you have a Supplier model in your application, which represents a supplier of goods in your shop and has several fields, including name, description and enabled field.

In order to make it a Sylius resource, you need to configure it under `sylius_resource` node. If you don't have it yet create a file `app/config/resources.yml`, import it in the `app/config/config.yml`.

```yaml
# app/config/resources.yml
sylius_resource:
    resources:
        app.supplier:
            driver: doctrine/orm
            classes:
                model: AppBundle\Entity\Supplier
```

```yaml
# app/config/config.yml
imports:
    - { resource: "resources.yml" }
```

That's it! Your class is now a resource. In order to learn what does it mean, please refer to the *SyliusResourceBundle* documentation.

Grid Definition

Now we can configure our first grid:

Note: Remember that a grid is **the way objects of a desired entity are displayed on its index view**. Therefore only fields that are useful for identification of objects are available - only `string` and `twig` type. Then even though a Supplier has also a description field, it is not needed on index and can't be displayed here.

```yaml
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
            driver:
                name: doctrine/orm
                options:
                    class: AppBundle\Entity\Supplier
            fields:
                name:
                    type: string
                    label: sylius.ui.name
                enabled:
                    type: twig
                    label: sylius.ui.enabled
                    options:
                        template: SyliusUiBundle:Grid/Field:enabled.html.twig # This
                        →will be a checkbox field
```

Remember to import your grid in the `app/config/grids/grids.yml` file which has to be imported in the `app/config/config.yml`.

```yaml
# app/config/grids/grids.yml
imports:
    - { resource: 'admin/supplier.yml' }
```

```yaml
# app/config/config.yml
imports:
    - { resource: "grids/grids.yml" }
```

Generating The CRUD Routing

That's it. SyliusResourceBundle allows to generate a default CRUD interface including the grid we have just defined. Just put this in your routing configuration!

```yaml
# app/config/routing/admin/supplier.yml
app_admin_supplier:
    resource: |
        alias: app.supplier
        section: admin
        templates: SyliusAdminBundle:Crud
```

(continues on next page)

```
        except: ['show']
        redirect: update
        grid: app_admin_supplier
        vars:
            all:
                subheader: app.ui.supplier # define a translation key for your entity␣
↪subheader
            index:
                icon: 'file image outline' # choose an icon that will be displayed␣
↪next to the subheader
    type: sylius.resource
```

```
# app/config/routing/admin.yml
app_admin_supplier:
    resource: 'supplier.yml'
```

```
# app/config/routing.yml
app_admin:
    resource: 'routing/admin.yml'
    prefix: /admin
```

This will generate the following paths:

- */admin/suppliers/* - [GET] - Your grid.

- */admin/suppliers/new* - [GET/POST] - Creating new supplier.

- */admin/suppliers/{id}/edit* - [GET/PUT] - Editing an existing supplier.

- */admin/suppliers/{id}* - [DELETE] - Deleting specific supplier.

- */admin/suppliers/{id}* - [GET] - Displaying specific supplier.

Tip: In the Semantic UI documentation you can find all possible icons you can choose for your grid.

Tip: See *how to add links to your new entity administration in the administration menu*.

Tip: Adding translations to the grid (read more *here*):

```
# app/Resources/translations/messages.en.yml
app:
    ui:
        supplier: Supplier
        suppliers: Suppliers
    menu:
        admin:
            main:
                additional:
                    header: Additional
                    suppliers: Suppliers
```

After that your new grid should look like that when accessing the */admin/suppliers/new* path in order to create new object:

And when accessing index on the */admin/suppliers/* path it should look like that:

Defining Filters

In order to make searching for certain things in your grid you can use filters.

```yaml
sylius_grid:
    grids:
        app_admin_supplier:
            ...
            filters:
                name:
                    type: string
                enabled:
                    type: boolean
```

How will it look like in the admin panel?

What about filtering by fields of related entities? For instance if you would like to filter your suppliers by their country of origin, which is a property of the associated address entity.

This first requires a *custom repository method* for your grid query:

```
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
            driver:
                name: doctrine/orm
                options:
                    class: AppBundle\Entity\Supplier
                    repository:
                        method: mySupplierGridQuery
```

Note: The repository method has to return a queryBuilder object, since the query has to adjustable depending on the filters and sorting the user later applies. Furthermore, all sub entities you wish to use later for filtering have to be joined explicitely in the query.

Then you can set up your filter to accordingly:

```
sylius_grid:
    grids:
        app_admin_supplier:
                ...
            filters:
                    ...
                country:
                    type: string
                    label: origin
                    options:
                        fields: [address.country]
                    form_options:
                        type: contains
```

Default Sorting

You can define by which field you want the grid to be sorted and how.

```
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
                ...
            sorting:
                name: asc
                ...
```

Then at the fields level, define that the field can be used for sorting:

```
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
                ...
            fields:
                name:
```

(continues on next page)

```
                    type: string
                    label: sylius.ui.name
                    sortable: ~
                ...
```

If your field is not of a "simple" type, f.i. a twig template with a specific path, you get sorting working with the following definition:

```yaml
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
            ...
            fields:
                ....
                origin:
                    type: twig
                    options:
                        template: "@AppBundle/Grid/Fields/myCountryFlags.html.twig"
                    path: address.country
                    label: app.ui.country
                    sortable: address.country
                ...
```

Pagination

You can limit how many items are visible on each page by providing an array of integers into the `limits` parameter. The first element of the array will be treated as the default, so by configuring:

```yaml
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
                ...
            limits: [30, 12, 48]
            ...
```

you will see thirty suppliers per page, also you will have the possibility to change the number of elements to either 12 or 48.

Note: Pagination limits are set by default to 10, 25 and 50 items per page. In order to turn it off, configure *limits: ~*.

Actions Configuration

Next step is adding some actions to the grid: create, update and delete.

Note: There are two types of actions that can be added to a grid: `main` which "influence" the whole grid (like adding new objects) and `item` which influence one row of the grid (one object) like editing or deleting.

```
# app/config/grids/admin/supplier.yml
sylius_grid:
    grids:
        app_admin_supplier:
                ...
            actions:
                main:
                    create:
                        type: create
                item:
                    update:
                        type: update
                    delete:
                        type: delete
```

This activates such a view on the */admin/suppliers/* path:

Your grid is ready to use!

Field Types

This is the list of built-in field types.

String

Simplest column type, which basically renders the value at given path as a string.

By default it uses the name of the field, but you can specify the path alternatively. For example:

```
sylius_grid:
    grids:
        app_user:
            fields:
                email:
                    type: string
                    label: app.ui.email # each filed type can have a label, we
↪suggest using translation keys instead of messages
                    path: contactDetails.email
```

This configuration will display the value from `$user->getContactDetails()->getEmail()`.

DateTime

This column type works exactly the same way as *string*, but expects *DateTime* instance and outputs a formatted date
and time string.

```
sylius_grid:
    grids:
        app_user:
            fields:
                birthday:
                    type: datetime
                    label: app.ui.birthday
                    options:
                        format: 'Y:m:d H:i:s' # this is the default value, but you
↪can modify it
```

Twig (*twig*)

Twig column type is the most flexible from all of them, because it delegates the logic of rendering the value to Twig
templating engine. You just have to specify the template and it will be rendered with the `data` variable available to
you.

```
sylius_grid:
    grids:
        app_user:
            fields:
                name:
                    type: twig
                    label: app.ui.name
                    options:
                        template: :Grid/Column:_prettyName.html.twig
```

In the `:Grid/Column:_prettyName.html.twig` template, you just need to render the value for example as
you see below:

```
<strong>{{ data }}</strong>
```

If you wish to render more complex grid fields just redefine the path of the field to root – `path:   .` in the yaml and
you can access all attributes of the object instance:

```
<strong>{{ data.name }}</strong>
<p>{{ data.description|markdown }}</p>
```

Field configuration

Each field can be configured with several configuration keys, to make it more suitable to your grid requirements.

Name	Type	Description
type	string	Type of column. Default field types are described *here*.
label	string	Label displayed in the field header. By default, it is field name.
path	string	Path to property displayed in field (can be property of resource or one of its referenced objects).
position	int	Position of field in the grid index view
options	array	Array of field options (see below).

`options` field can contains following fields:

Name	Type	Description	Default
template	string	Available (and required) only for *twig* column type. Path to template that is used to render column value.	
format	string	Available only for *datetime* field type.	Y:m:d H:i:s

Filters

Filters on grids are a kind of search prepared for each grid. Having a grid of objects you can filter out only those with a specified name, or value etc. Here you can find the supported filters. Keep in mind you can very easily define your own ones!

String

Simplest filter type. It can filter by one or multiple fields.

```yaml
sylius_grid:
    grids:
        app_user:
            filters:
                username:
                    type: string
                email:
                    type: string
                firstName:
                    type: string
                lastName:
                    type: string
```

The filter allows the user to select following search options:

- contains

- not contains

- equal

- not equal

- starts with

- ends with

- empty

- not empty

- in

- not in

If you don't want display all theses matching possibilities, you can choose just one of them. Then only the input field will be displayed. You can achieve it like that:

```
sylius_grid:
    grids:
        app_user:
            filters:
                username:
                    type: string
                    form_options:
                        type: contains
```

By configuring a filter like above you will have only an input field for filtering users objects that `contain` a given string in their username.

Boolean

This filter checks if a value is true or false.

```
sylius_grid:
    grids:
        app_channel:
            filters:
                enabled:
                    type: boolean
```

Date

This filter checks if a chosen datetime field is between given dates.

```
sylius_grid:
    grids:
        app_order:
            filters:
                createdAt:
                    type: date
                completedAt:
                    type: date
```

Entity

This type filters by a chosen entity.

```
sylius_grid:
    grids:
        app_order:
            filters:
                channel:
```

```yaml
                type: entity
                form_options:
                    class: "%app.model.channel%"
            customer:
                type: entity
                form_options:
                    class: "%app.model.customer%"
```

Money

This filter checks if an amount is in range and in a specified currency

```yaml
sylius_grid:
    grids:
        app_order:
            filters:
                total:
                    type: money
                    form_options:
                        scale: 3
                    options:
                        currency_field: currencyCode
                        scale: 3
```

> **Warning:** Providing different `scale` between **form_options** and **options** may cause unwanted, and plausibly volatile results.

Exists

This filter checks if the specified field contains any value

```yaml
sylius_grid:
    grids:
        app_order:
            filters:
                date:
                    type: exists
                    options:
                        field: completedAt
```

Custom Filters

> **Tip:** If you need to create a custom filter, *read the docs here*.

Custom Field Type

There are certain cases when built-in field types are not enough. Sylius Grids allows to define new types with ease!

All you need to do is create your own class implementing FieldTypeInterface and register it as a service.

```php
<?php

namespace AppBundle\Grid\FieldType;

use Sylius\Component\Grid\Definition\Field;
use Sylius\Component\Grid\FieldTypes\FieldTypeInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class CustomType implements FieldTypeInterface
{
    public function render(Field $field, $data, array $options = [])
    {
        // Your rendering logic... Use Twig, PHP or even external api...
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver
            ->setDefaults([
                'dynamic' => false
            ])
            ->setAllowedTypes([
                'dynamic' => ['boolean']
            ])
        ;
    }

    public function getName()
    {
        return 'custom';
    }
}
```

That is all. Now register your new field type as a service.

```yaml
# app/config/services.yml
app.grid_field.custom:
    class: AppBundle\Grid\FieldType\CustomType
    tags:
        - { name: sylius.grid_field, type: custom }
```

Now you can use your new column type in the grid configuration!

```yaml
sylius_grid:
    grids:
        app_admin_supplier:
            driver:
                name: doctrine/orm
                options:
                    class: AppBundle\Entity\Supplier
            fields:
                name:
                    type: custom
                    label: sylius.ui.name
```

Custom Filter

Sylius Grids come with built-in filters, but there are use-cases where you need something more than basic filter. Grids allow you to define your own filter types!

To add a new filter, we need to create an appropriate class and form type.

```php
<?php

namespace AppBundle\Grid\Filter;

use Sylius\Component\Grid\Data\DataSourceInterface;
use Sylius\Component\Grid\Filtering\FilterInterface;

class SuppliersStatisticsFilter implements FilterInterface
{
    public function apply(DataSourceInterface $dataSource, $name, $data, array
→$options = [])
    {
        // Your filtering logic. DataSource is kind of query builder.
        // $data['stats'] contains the submitted value!
        // here is an example
        $dataSource->restrict($dataSource->getExpressionBuilder()->equals('stats',
→$data['stats']));
    }
}
```

And the form type:

```php
<?php

namespace AppBundle\Form\Type\Filter;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class SuppliersStatisticsFilterType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add(
            'stats',
            ChoiceType::class,
            ['choices' => range($options['range'][0], $options['range'][1])]
        );
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver
            ->setDefaults([
                'range' => [0, 10],
            ])
            ->setAllowedTypes('range', ['array'])
        ;
    }
```

```
}
```

Create a template for the filter, similar to the existing ones:

```twig
# app/Resources/views/Grid/Filter/suppliers_statistics.html.twig
{% form_theme form 'SyliusUiBundle:Form:theme.html.twig' %}

{{ form_row(form) }}
```

That is all. Now let's register your new filter type as service.

```yaml
# app/config/services.yml

services:
    app.grid.filter.suppliers_statistics:
        class: AppBundle\Grid\Filter\SuppliersStatisticsFilter
        tags:
            -
                name: sylius.grid_filter
                type: suppliers_statistics
                form_type: AppBundle\Form\Type\Filter\SuppliersStatisticsFilterType
```

Now you can use your new filter type in the grid configuration!

```yaml
sylius_grid:
    grids:
        app_tournament:
            driver: doctrine/orm
            resource: app.tournament
            filters:
                stats:
                    type: suppliers_statistics
                    form_options:
                        range: [0, 100]
    templates:
        filter:
            suppliers_statistics: 'AppBundle:Grid/Filter:suppliers_statistics.html.
→twig'
```

Custom Action

There are certain cases when built-in action types are not enough.

All you need to do is create your own action template and register it for the `sylius_grid`.

In the template we will specify the button's icon to be `mail` and its colour to be `purple`.

```twig
{% import '@SyliusUi/Macro/buttons.html.twig' as buttons %}

{% set path = options.link.url|default(path(options.link.route, options.link.
→parameters)) %}

{{ buttons.default(path, action.label, null, 'mail', 'purple') }}
```

Now configure the new action's template like below in the `app/config/config.yml`:

```yaml
# app/config/config.yml
sylius_grid:
    templates:
        action:
            contactSupplier: "@App/Grid/Action/contactSupplier.html.twig"
```

From now on you can use your new action type in the grid configuration!

Let's assume that you already have a route for contacting your suppliers, then you can configure the grid action:

```yaml
sylius_grid:
    grids:
        app_admin_supplier:
            driver:
                name: doctrine/orm
                options:
                    class: AppBundle\Entity\Supplier
            actions:
                item:
                    contactSupplier:
                        type: contactSupplier
                        label: Contact Supplier
                        options:
                            link:
                                route: app_admin_contact_supplier
                                parameters:
                                    id: resource.id
```

Custom Bulk Action

There are cases where pressing a button per item in a grid is not suitable. And there are also certain cases when built-in bulk action types are not enough.

All you need to do is create your own bulk action template and register it for the `sylius_grid`.

In the template we will specify the button's icon to be `export` and its colour to be `orange`.

```twig
{% import '@SyliusUi/Macro/buttons.html.twig' as buttons %}

{% set path = options.link.url|default(path(options.link.route)) %}

{{ buttons.default(path, action.label, null, 'export', 'orange') }}
```

Now configure the new action's template like below in the `app/config/config.yml`:

```yaml
# app/config/config.yml
sylius_grid:
    templates:
        bulk_action:
            export: "@App/Grid/BulkAction/export.html.twig"
```

From now on you can use your new bulk action type in the grid configuration!

Let's assume that you already have a route for exporting by injecting ids, then you can configure the grid action:

```yaml
sylius_grid:
    grids:
```

```
        app_admin_product:
            ...
            actions:
                bulk:
                    export:
                        type: export
                        label: Export Data
                        options:
                            link:
                                route: app_admin_product_export
                                parameters:
                                    format: csv
```

Configuration Reference

Here you will find all configuration options of `sylius_grid`.

```
sylius_grid:
    grids:
        app_user: # Your grid name
            driver:
                name: doctrine/orm
                options:
                    class: "%app.model.user%"
                    repository:
                        method: myCustomMethod
                        arguments:
                            id: resource.id
            sorting:
                name: asc
            limits: [10, 25, 50, 100]
            fields:
                name:
                    type: twig # Type of field
                    label: Name # Label
                    path: . # dot means a whole object
                    sortable: ~ | field path
                    position: 100
                    options:
                        template: :Grid/Column:_name.html.twig # Only twig column
                        vars:
                            labels: # a template of how does the label look like
                    enabled: true
            filters:
                name:
                    type: string # Type of filter
                    label: app.ui.name
                    enabled: true
                    template: ~
                    position: 100
                    options:
                        fields: { }
                    form_options:
                        type: contains # type of string filtering option, if you one␣
↪to have just one
```

```yaml
                default_value: ~
            enabled:
                type: boolean # Type of filter
                label: app.ui.enabled
                enabled: true
                template: ~
                position: 100
                options:
                    field: enabled
                form_options: { }
                default_value: ~
            date:
                type: date # Type of filter
                label: app.ui.created_at
                enabled: true
                template: ~
                position: 100
                options:
                    field: createdAt
                form_options: { }
                default_value: ~
            channel:
                type: entity # Type of filter
                label: app.ui.channel
                enabled: true
                template: ~
                position: 100
                options:
                    fields: [channel]
                form_options:
                    class: "%app.model.channel%"
                default_value: ~
        actions:
            main:
                create:
                    type: create
                    label: sylius.ui.create
                    enabled: true
                    icon: ~
                    position: 100
            item:
                update:
                    type: update
                    label: sylius.ui.edit
                    enabled: true
                    icon: ~
                    position: 100
                    options: { }
                delete:
                    type: delete
                    label: sylius.ui.delete
                    enabled: true
                    icon: ~
                    position: 100
                    options: { }
                show:
                    type: show
```

```yaml
                    label: sylius.ui.show
                    enabled: true
                    icon: ~
                    position: 100
                    options:
                        link:
                            route: app_user_show
                            parameters:
                                id: resource.id
                archive:
                    type: archive
                    label: sylius.ui.archive
                    enabled: true
                    icon: ~
                    position: 100
                    options:
                        restore_label: sylius.ui.restore
            bulk:
                delete:
                    type: delete
                    label: sylius.ui.delete
                    enabled: true
                    icon: ~
                    position: 100
                    options: { }
            subitem:
                addresses:
                    type: links
                    label: sylius.ui.manage_addresses
                    options:
                        icon: cubes
                        links:
                            index:
                                label: sylius.ui.list_addresses
                                icon: list
                                route: app_admin_user_address_index
                                visible: resource.hasAddress
                                parameters:
                                    userId: resource.id
                            create:
                                label: sylius.ui.generate
                                icon: random
                                route: app_admin_user_address_create
                                parameters:
                                    userId: resource.id
```

SyliusInventoryBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Flexible inventory management for Symfony applications.

With minimal configuration you can implement inventory tracking in your project.

It's fully customizable, but the default setup should be optimal for most use cases.

There is *StockableInterface* and *InventoryUnit* model inside the bundle.

There are services **AvailabilityChecker**, **InventoryOperator** and **InventoryChangeListener**.

You'll get familiar with them in later parts of this documentation.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/inventory-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/inventory-bundle
```

Adding required bundles to the kernel

First, you need to enable the bundle inside the kernel. If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to the kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        // ...
        new FOS\RestBundle\FOSRestBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new Sylius\Bundle\InventoryBundle\SyliusInventoryBundle(),
    );
}
```

Creating your entities

Let's assume we want to implement a book store application and track the books inventory.

You have to create a *Book* and an *InventoryUnit* entity, living inside your application code. We think that **keeping the app-specific bundle structure simple** is a good practice, so let's assume you have your `AppBundle` registered under `App\Bundle\AppBundle` namespace.

We will create *Book* entity.

```php
<?php

// src/App/AppBundle/Entity/Book.php
namespace App\AppBundle\Entity;

use Sylius\Component\Inventory\Model\StockableInterface;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="app_book")
 */
class Book implements StockableInterface
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string")
     */
    protected $isbn;

    /**
     * @ORM\Column(type="string")
     */
    protected $title;

    /**
     * @ORM\Column(type="integer")
     */
    protected $onHand;

    public function __construct()
    {
        $this->onHand = 1;
    }

    public function getId()
    {
        return $this->id;
    }

    public function getIsbn()
    {
        return $this->isbn;
    }

    public function setIsbn($isbn)
    {
        $this->isbn = $isbn;
    }
```

```php
    public function getSku()
    {
        return $this->getIsbn();
    }

    public function getTitle()
    {
        return $this->title;
    }

    public function setTitle($title)
    {
        $this->title = $title;
    }

    public function getInventoryName()
    {
        return $this->getTitle();
    }

    public function isInStock()
    {
        return 0 < $this->onHand;
    }

    public function getOnHand()
    {
        return $this->onHand;
    }

    public function setOnHand($onHand)
    {
        $this->onHand = $onHand;
    }
}
```

Note: This example shows the full power of *StockableInterface*.

In order to track the books inventory our *Book* entity must implement *StockableInterface*. Note that we added ->getSku() method which is alias to ->getIsbn(), this is the power of the interface, we now have full control over the entity mapping. In the same way ->getInventoryName() exposes the book title as the displayed name for our stockable entity.

The next step requires the creating of the *InventoryUnit* entity, let's do this now.

```php
<?php

// src/App/AppBundle/Entity/InventoryUnit.php
namespace App\AppBundle\Entity;

use Sylius\Component\Inventory\Model\InventoryUnit as BaseInventoryUnit;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
```

```php
 * @ORM\Table(name="app_inventory_unit")
 */
class InventoryUnit extends BaseInventoryUnit
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;
}
```

Note that we are using base model from Sylius component, which means inheriting some functionality inventory component provides. *InventoryUnit* holds the reference to stockable object, which is *Book* in our case. So, if we use the *InventoryOperator* to create inventory units, they will reference the given book entity.

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
sylius_inventory:
    driver: doctrine/orm
    resources:
        inventory_unit:
            classes:
                model: App\AppBundle\Entity\InventoryUnit
```

Updating database schema

Remember to update your database schema.

For "**doctrine/orm**" driver run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Here is a quick reference for the default models.

InventoryUnit

Each unit holds a reference to a stockable object and its state, which can be **sold** or **returned**. It also provides some handy shortcut methods like *isSold*.

Stockable

In order to be able to track stock levels in your application, you must implement *StockableInterface* or use the *Stockable* model. It uses the SKU to identify stockable and need to provide display name. It can get/set current stock level with *getOnHand* and *setOnHand* methods.

Using the services

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

When using the bundle, you have access to several handy services.

AvailabilityChecker

The name speaks for itself, this service checks availability for given stockable object. *AvailabilityChecker* relies on the current stock level.

There are two methods for checking availability. `->isStockAvailable()` just checks whether stockable object is available in stock and doesn't care about quantity. `->isStockSufficient()` checks if there is enough units in the stock for given quantity.

InventoryOperator

Inventory operator is the heart of this bundle. It can be used to manage stock levels and inventory units. Creating/destroying inventory units with a given state is also the operators job.

Twig Extension

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

There are two handy twig functions bundled in: *sylius_inventory_is_available* and *sylius_inventory_is_sufficient*.

They are simple proxies to the availability checker, and can be used to show if the stockable object is available/sufficient.

Here is a simple example, note that *product* variable has to be an instance of *StockableInterface*.

```
{% if not sylius_inventory_is_available(product) %}
    <span class="label label-important">out of stock</span>
{% endif %}
```

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration reference

```yaml
sylius_inventory:
    # The driver used for persistence layer.
    driver: ~
    # Enable or disbale tracking inventory
    track_inventory: true
    # The availability checker service id.
    checker: sylius.availability_checker.default
    # The inventory operator service id.
    operator: sylius.inventory_operator.default
    # Array of events for InventoryChangeListener
    events: ~
    resources:
        inventory_unit:
            classes:
                model:      Sylius\Component\Inventory\Model\InventoryUnit
                interface:  Sylius\Component\Inventory\Model\InventoryUnitInterface
                controller:
→Sylius\Bundle\InventoryBundle\Controller\InventoryUnitController
                repository: ~ # You can override the repository class here.
                factory:    Sylius\Component\Resource\Factory\Factory
        stockable:
            classes:
                model: ~ # The stockable model class.
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Inventory in the Sylius platform* - concept documentation

SyliusMailerBundle

Sending customizable e-mails has never been easier in Symfony.

You can configure different e-mail types in the YAML or in database. (and use YAML as fallback) This allows you to send out e-mails with one simple method call, providing an unique code and data.

The bundle supports adapters, by default e-mails are rendered using Twig and sent via Swiftmailer, but you can easily implement your own adapter and delegate the whole operation to external API.

This bundle provides easy integration of the *Sylius Mailer component* with any Symfony full-stack application.

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/mailer-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/mailer-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),

        new Sylius\Bundle\MailerBundle\SyliusMailerBundle(),
    );
}
```

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
sylius_mailer:
    sender:
        name: My website
        address: no-reply@my-website.com
```

Congratulations! The bundle is now installed and ready to use.

Your First Email

Let's say you want to send a notification to the website team when someone submits a new position to your movie catalog website!

You can do it in few simple steps:

Configure Your E-Mail

In your `app/config/config.yml`, under `sylius_mailer` you should configure the email:

```yaml
# app/config/config.yml

sylius_mailer:
    sender:
        name: Movie Database Example
        address: no-reply@movie-database-example.com
    emails:
        movie_added_notification:
            subject: A new movie {{ movie.title }} has been submitted
            template: AppBundle:Email:movieAddedNotification.html.twig
```

That's it! Your unique code is "movie_added_notification". Now, let's create the template.

Creating Your Template

In your `app/Resources/views/Email:movieAddedNotification.html.twig` put the following Twig code:

```twig
{% block subject %}
    A new movie {{ movie.title }} has been submitted
{% endblock %}

{% block body %}
    Hello Movie Database Example!

    A new movie has been submitted for review to your database.

    Title: {{ movie.title }}
    Added by {{ user.name }}

    Please review it and accept or reject!
{% endblock %}
```

That should be enough!

Sending The E-Mail

The service responsible for sending an e-mail has id `sylius.email_sender`. All you need to do is retrieve it from the container or inject to a listener:

```php
<?php

namespace App\AppBundle\Controller;

use Symfony\Component\HttpFoundation\Request;

class MovieController
{
    public function submitAction(Request $request)
    {
```

(continues on next page)

```
        // Your code.

        $this->get('sylius.email_sender')->send('movie_added_notification', array(
    'team@website.com'), array('movie' => $movie, 'user' => $this->getUser()));
    }
}
```

Listener example:

```php
<?php

namespace App\AppBundle\Controller;

use App\Event\MovieCreatedEvent;
use Sylius\Component\Mailer\Sender\SenderInterface;

class MovieNotificationListener
{
    private $sender;

    public function __construct(SenderInterface $sender)
    {
        $this->sender = $sender;
    }

    public function onMovieCreation(MovieCreatedEvent $event)
    {
        $movie = $event->getMovie();
        $user = $event->getUser();

        $this->sender->send('movie_added_notification', array('team@website.com'),
    array('movie' => $movie, 'user' => $user));
    }
}
```

We recommend using events approach, but you can send e-mails from anywhere in your application. Enjoy!

Using Custom Adapter

There are certain use cases, where you do not want to send the e-mail from your app, but delegate the task to an external API.

It is really simple with Adapters system!

Implement Your Adapter

Create your adapter class and add your custom logic for sending:

```php
<?php

namespace App\Mailer\Adapter;

use Sylius\Component\Mailer\Sender\Adapter\AbstractAdapter;
use Sylius\Component\Mailer\Model\EmailInterface;
```

```php
use Sylius\Component\Mailer\Renderer\RenderedEmail;

class CustomAdapter extends AbstractAdapter
{
    public function send(array $recipients, $senderAddress, $senderName,
↪RenderedEmail $renderedEmail, EmailInterface $email, array $data)
    {
        // Your custom logic.
    }
}
```

Register New Adapter In Container

In your `services.yml` file, simply add your adapter definition.

```yaml
services:
    app.email_sender.adapter.custom:
        parent: sylius.email_sender.adapter.abstract
        class: App\Mailer\Adapter\CustomAdapter
```

Configure The New Adapter

Now you just need to put service name under `sylius_mailer` configuration in `app/config/config.yml`.

```yaml
sylius_mailer:
    sender_adapter: app.email_sender.adapter.custom
```

That's it! Your new adapter will be used to send out e-mails. You can do whatever you want there!

Configuration reference

```yaml
sylius_mailer:
    sender_adapter: sylius.email_sender.adapter.swiftmailer # Adapter for sending e-
↪mails.
    renderer_adapter: sylius.email_renderer.adapter.twig # Adapter for rendering e-
↪mails.
    sender:
        name: # Required - default sender name.
        address: # Required - default sender e-mail address.
    templates: # Your templates available for selection in backend!
        label: Template path
        label: Template path
        label: Template path
    emails:
        your_email:
            subject: Subject of your email
            template: AppBundle:Email:yourEmail.html.twig
            enabled: true/false
            sender:
                name: Custom name
                address: Custom sender address for this e-mail
```

```
        your_another_email:
            subject: Subject of your another email
            template: AppBundle:Email:yourAnotherEmail.html.twig
            enabled: true/false
            sender:
                name: Custom name
                address: Custom sender address for this e-mail
```

Learn more

- *Emails in the Sylius platform* - concept documentation

SyliusOrderBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

This bundle is a foundation for sales order handling for Symfony projects. It allows you to use any model as the merchandise.

It also includes a super flexible adjustments feature, which serves as a basis for any taxation, shipping charges or discounts system.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/order-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/order-bundle
```

Adding required bundles to the kernel

First, you need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to the kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle()
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),

        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new Sylius\Bundle\MoneyBundle\SyliusMoneyBundle(),
        new Sylius\Bundle\OrderBundle\SyliusOrderBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Updating database schema

Remember to update your database schema.

For "**doctrine/orm**" driver run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

The Order, OrderItem and OrderItemUnit

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Here is a quick reference of what the default models can do for you.

Order basics

Each order has 2 main identifiers, an *ID* and a human-friendly *number*. You can access those by calling ->getId() and ->getNumber() respectively. The number is mutable, so you can change it by calling ->setNumber('E001') on the order instance.

```php
<?php
```

(continues on next page)

```php
$order->getId();
$order->getNumber();

$order->setNumber('E001');
```

Order totals

Note: All money amounts in Sylius are represented as "cents" - integers.

An order has 3 basic totals, which are all persisted together with the order.

The first total is the *items total*, it is calculated as the sum of all item totals (including theirs adjustments).

The second total is the *adjustments total*, you can read more about this in next chapter.

```php
<?php

echo $order->getItemsTotal(); // 1900.
echo $order->getAdjustmentsTotal(); // -250.

$order->calculateTotal();
echo $order->getTotal(); // 1650.
```

The main order total is a sum of the previously mentioned values. You can access the order total value using the `->getTotal()` method.

Note: It's not needed to call `calculateTotal()` method, as both `itemsTotal` and `adjustmentsTotal` are automatically updated after each operation that can influence their values.

Items management

The collection of items (Implementing the `Doctrine\Common\Collections\Collection` interface) can be obtained using the `->getItems()`. To add or remove items, you can simply use the `addItem` and `removeItem` methods.

```php
<?php

// $item1 and $item2 are instances of OrderItemInterface.
$order
    ->addItem($item)
    ->removeItem($item2)
;
```

OrderItem basics

An order item model has only the id as identifier, also it has the order to which it belongs, accessible via `->getOrder()` method.

The sellable object can be retrieved and set, using the following setter and getter - `->getProduct()` &
`->setVariant(ProductVariantInterface $variant)`.

```php
<?php

$item->setVariant($book);
```

Note: In most cases you'll use the **OrderBuilder** service to create your orders.

Just like for the order, the total is available via the same method, but the unit price is accessible using the
`->getUnitPrice()` Each item also can calculate its total, using the quantity (`->getQuantity()`) and the
unit price.

Warning: Concept of `OrderItemUnit` allows better management of `OrderItem`'s quantity. Because of that,
it's needed to use *OrderItemQuantityModifier* to handle quantity modification properly.

```php
<?php

$item = $itemRepository->createNew();
$item->setVariant($book);
$item->setUnitPrice(2000)

$orderItemQuantityModifier->modify($item, 4); //modifies item's quantity to 4

echo $item->getTotal(); // 8000.
```

An OrderItem can also hold adjustments.

Units management

Each element from `units` collection in `OrderItem` represents single, separate unit from order. It's total is sum of its
`item` unit price and totals' of each adjustments. Unit's can be added and removed using `addUnit` and `removeUnit`
methods from `OrderItem`, but it's highly recommended to use *OrderItemQuantityModifier*.

The Adjustments

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
it. You can help us making documentation up to date via Sylius Github. Thank you!

Adjustments are based on simple but powerful idea inspired by Spree adjustments. They serve as foundation for any
tax, shipping and discounts systems.

Adjustment model

Note: To be written. Learn more in *the Book*.

Using the services

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

When using the bundle, you have access to several handy services. You can use them to retrieve and persist orders.

Managers and Repositories

> **Note:** Sylius uses `Doctrine\Common\Persistence` interfaces.

You have access to following services which are used to manage and retrieve resources.

This set of default services is shared across almost all Sylius bundles, but this is just a convention. You're interacting with them like you usually do with own entities in your project.

```php
<?php

// ObjectManager which is capable of managing the resources.
// For *doctrine/orm* driver it will be EntityManager.
$this->get('sylius.manager.order');
$this->get('sylius.manager.order_item');
$this->get('sylius.manager.order_item_unit');
$this->get('sylius.manager.adjustment');

// ObjectRepository for the Order resource, it extends the base EntityRepository.
// You can use it like usual entity repository in project.
$this->get('sylius.repository.order');
$this->get('sylius.repository.order_item');
$this->get('sylius.repository.order_item_unit');
$this->get('sylius.repository.adjustment');

// Those repositories have some handy default methods, for example...
$item = $itemRepository->createNew();
$orderRepository->find(4);
$paginator = $orderRepository->createPaginator(array('confirmed' => false)); // Get
→Pagerfanta instance for all unconfirmed orders.
```

OrderItemQuantityModifier

`OrderItemQuantityModifier` should be used to modify `OrderItem` quantity, because of whole background units' logic, that needs to be done. This service handles this task, adding and removing proper amounts of units to `OrderItem`.

```php
<?php

$orderItemFactory = $this->get('sylius.factory.order_item');
$orderItemQuantityModifier = $this->get('sylius.order_item_quantity_modifier');

$orderItem = $orderItemFactory->createNew();
```

(continues on next page)

```php
$orderItem->getQuantity(); // default quantity of order item is 0

$orderItem->setUnitPrice(1000);

$orderItemQuantityModifier->modify($orderItem, 4);

$orderItem->getQuantity(); // after using modifier, quantity is as expected
$orderItem->getTotal();    // item's total is sum of all units' total (units have
↪been created by modifier)

$orderItemQuantityModifier->modify($orderItem, 2);

// OrderItemQuantityModifier can also reduce item's quantity and remove unnecessary
↪units

$orderItem->getQuantity(); // 2
$orderItem->getTotal();    // 2000
```

Summary

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration reference

```yaml
sylius_order:
    driver: doctrine/orm
    resources:
        order:
            classes:
                model: Sylius\Component\Core\Model\Order
                controller: Sylius\Bundle\CoreBundle\Controller\OrderController
                repository: Sylius\Bundle\CoreBundle\Doctrine\ORM\OrderRepository
                form: Sylius\Bundle\CoreBundle\Form\Type\Order\OrderType
                interface: Sylius\Component\Order\Model\OrderInterface
                factory: Sylius\Component\Resource\Factory\Factory
        order_item:
            classes:
                model: Sylius\Component\Core\Model\OrderItem
                form: Sylius\Bundle\CoreBundle\Form\Type\Order\OrderItemType
                interface: Sylius\Component\Order\Model\OrderItemInterface
                controller: Sylius\Bundle\OrderBundle\Controller\OrderItemController
                factory: Sylius\Component\Resource\Factory\Factory
        order_item_unit:
            classes:
                model: Sylius\Component\Core\Model\OrderItemUnit
                interface: Sylius\Component\Order\Model\OrderItemUnitInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Order\Factory\OrderItemUnitFactory
        adjustment:
            classes:
                model: Sylius\Component\Order\Model\Adjustment
```

```
                interface: Sylius\Component\Order\Model\AdjustmentInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
        order_sequence:
            classes:
                model: Sylius\Component\Order\Model\OrderSequence
                interface: Sylius\Component\Order\Model\OrderSequenceInterface
                factory: Sylius\Component\Resource\Factory\Factory
    expiration:
        cart: '2 days'
        order: '5 days'
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Processors

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Order processors are responsible of manipulating the orders to apply different predefined adjustments or other modifications based on order state.

Registering custom processors

Once you have your own *OrderProcessorInterface* implementation you need to register it as a service.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services
                               http://symfony.com/schema/dic/services/services-1.0.xsd
↪">

    <services>
        <service id="acme.order_processor.custom" class=
↪"Acme\ShopBundle\OrderProcessor\CustomOrderProcessor">
            <tag name="sylius.order_processor" priority="0" />
        </service>
    </services>
</container>
```

> **Note:** You can add your own processor to the *CompositeOrderProcessor* using *sylius.order_processor*

Using CompositeOrderProcessor

All processor services containing *sylius.order_processor* tag can be launched as follows:

In a controller:

```php
<?php

// Fetch order from DB
$order = ...;

// Get the processor from the container or inject the service
$orderProcessor = ...;

$orderProcessor->process($order);
```

Note: The *CompositeOrderProcessor* is named as ' sylius.order_processing.order_processor' in the container and contains all services tagged as *sylius.order_processor*

Learn more

- *Carts & Orders in the Sylius platform* - concept documentation

SyliusProductBundle

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/product-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/product-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new Sylius\Bundle\ProductBundle\SyliusProductBundle(),
        new Sylius\Bundle\AttributeBundle\SyliusAttributeBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new Sylius\Bundle\LocaleBundle\SyliusLocaleBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
stof_doctrine_extensions:
    orm:
        default:
            sluggable: true
            timestampable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Congratulations! The bundle is now installed and ready to use.

The Product

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Retrieving products

Retrieving a product from the database should always happen via repository, which always implements `Sylius\Bundle\ResourceBundle\Model\RepositoryInterface`. If you are using Doctrine, you're already familiar with this concept, as it extends the native Doctrine `ObjectRepository` interface.

Your product repository is always accessible via the `sylius.repository.product` service.

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');
}
```

Retrieving products is simple as calling proper methods on the repository.

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');

    $product = $repository->find(4); // Get product with id 4, returns null if not
→found.
    $product = $repository->findOneBy(array('slug' => 'my-super-product')); // Get
→one product by defined criteria.

    $products = $repository->findAll(); // Load all the products!
    $products = $repository->findBy(array('special' => true)); // Find products
→matching some custom criteria.
}
```

Product repository also supports paginating products. To create a Pagerfanta instance use the `createPaginator` method.

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');

    $products = $repository->createPaginator();
    $products->setMaxPerPage(3);
    $products->setCurrentPage($request->query->get('page', 1));

  // Now you can return products to the template and iterate over it to get products
→from the current page.
}
```

The paginator also can be created for specific criteria and with desired sorting.

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');

    $products = $repository->createPaginator(array('foo' => true), array('createdAt'
    ↪=> 'desc'));
    $products->setMaxPerPage(3);
    $products->setCurrentPage($request->query->get('page', 1));
}
```

Creating new product object

To create new product instance, you can simply call `createNew()` method on the factory.

```php
<?php

public function myAction(Request $request)
{
    $factory = $this->container->get('sylius.factory.product');
    $product = $factory->createNew();
}
```

Note: Creating a product via this factory method makes the code more testable, and allows you to change the product class easily.

Saving & removing product

To save or remove a product, you can use any `ObjectManager` which manages Product. You can always access it via alias `sylius.manager.product`. But it's also perfectly fine if you use `doctrine.orm.entity_manager` or other appropriate manager service.

```php
<?php

public function myAction(Request $request)
{
    $factory = $this->container->get('sylius.factory.product');
    $manager = $this->container->get('sylius.manager.product'); // Alias for
    ↪appropriate doctrine manager service.

    $product = $factory->createNew();

    $product
        ->setName('Foo')
        ->setDescription('Nice product')
    ;

    $manager->persist($product);
    $manager->flush(); // Save changes in database.
}
```

To remove a product, you also use the manager.

```php
<?php

public function myAction(Request $request)
{
    $repository = $this->container->get('sylius.repository.product');
    $manager = $this->container->get('sylius.manager.product');

    $product = $repository->find(1);

    $manager->remove($product);
    $manager->flush(); // Save changes in database.
}
```

Properties

A product can also have a set of defined Properties (*Attributes*), you'll learn about them in next chapter of this documentation.

Forms

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

The bundle ships with a set of useful form types for all models. You can use the defaults or *override them* with your own forms.

Product form

The product form type is named `sylius_product` and you can create it whenever you need, using the form factory.

```php
<?php

// src/Acme/ShopBundle/Controller/ProductController.php

namespace Acme\ShopBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DemoController extends Controller
{
    public function fooAction(Request $request)
    {
        $form = $this->get('form.factory')->create('sylius_product');
    }
}
```

The default product form consists of following fields.

Field	Type
name	text
description	textarea
metaDescription	text
metaKeywords	text

You can render each of these using the usual Symfony way `{{ form_row(form.description) }}`.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration reference

```yaml
sylius_product:
    driver: doctrine/orm
    resources:
        product:
            classes:
                model: Sylius\Component\Core\Model\Product
                repository: Sylius\Bundle\CoreBundle\Doctrine\ORM\ProductRepository
                form: Sylius\Bundle\CoreBundle\Form\Type\Product\ProductType
                interface: Sylius\Component\Product\Model\ProductInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Product\Factory\ProductFactory
            translation:
                classes:
                    model: Sylius\Component\Core\Model\ProductTranslation
                    form:
→Sylius\Bundle\CoreBundle\Form\Type\Product\ProductTranslationType
                    interface:
→Sylius\Component\Product\Model\ProductTranslationInterface
                    controller:
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    factory: Sylius\Component\Resource\Factory\Factory
        product_variant:
            classes:
                model: Sylius\Component\Core\Model\ProductVariant
                repository:
→Sylius\Bundle\ProductBundle\Doctrine\ORM\ProductVariantRepository
                form: Sylius\Bundle\CoreBundle\Form\Type\Product\ProductVariantType
                interface: Sylius\Component\Product\Model\ProductVariantInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Product\Factory\ProductVariantFactory
        product_option:
            classes:
                repository:
→Sylius\Bundle\ProductBundle\Doctrine\ORM\ProductOptionRepository
                model: Sylius\Component\Product\Model\ProductOption
                interface: Sylius\Component\Product\Model\ProductOptionInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
```

(continues on next page)

```
                    factory: Sylius\Component\Resource\Factory\TranslatableFactory
                    form: Sylius\Bundle\ProductBundle\Form\Type\ProductOptionType
            translation:
                classes:
                    model: Sylius\Component\Product\Model\ProductOptionTranslation
                    interface:
→Sylius\Component\Product\Model\ProductOptionTranslationInterface
                    controller:
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    factory: Sylius\Component\Resource\Factory\Factory
                    form:
→Sylius\Bundle\ProductBundle\Form\Type\ProductOptionTranslationType
        product_option_value:
            classes:
                model: Sylius\Component\Product\Model\ProductOptionValue
                interface: Sylius\Component\Product\Model\ProductOptionValueInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\TranslatableFactory
                form: Sylius\Bundle\ProductBundle\Form\Type\ProductOptionValueType
            translation:
                classes:
                    model:
→Sylius\Component\Product\Model\ProductOptionValueTranslation
                    interface:
→Sylius\Component\Product\Model\ProductOptionValueTranslationInterface
                    controller:
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    factory: Sylius\Component\Resource\Factory\Factory
                    form:
→Sylius\Bundle\ProductBundle\Form\Type\ProductOptionValueTranslationType
        product_association:
            classes:
                model: Sylius\Component\Product\Model\ProductAssociation
                interface: Sylius\Component\Product\Model\ProductAssociationInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\ProductBundle\Form\Type\ProductAssociationType
        product_association_type:
            classes:
                model: Sylius\Component\Product\Model\ProductAssociationType
                interface:
→Sylius\Component\Product\Model\ProductAssociationTypeInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\ProductBundle\Form\Type\ProductAssociationTypeType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Products in the Sylius platform* - concept documentation

SyliusPromotionBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Promotions system for Symfony applications.

With minimal configuration you can introduce promotions and coupons into your project. The following types of promotions are available and **totally mixable**:

- percentage discounts

- fixed amount discounts

- promotions limited by time

- promotions limited by a maximum number of usages

- promotions based on coupons

This means you can for instance create the following promotions :

- 20$ discount for New Year orders having more than 3 items

- 8% discount for Christmas orders over 100 EUR

- first 3 orders have 100% discount

- 5% discount this week with the coupon code *WEEK5*

- 40€ discount with the code you have received by mail

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your `composer.json` and download the package.

If you have Composer installed globally.

```
$ composer require sylius/promotion-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/promotion-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add `SyliusResourceBundle` and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new Sylius\Bundle\PromotionBundle\SyliusPromotionBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Promotion Subject configuration

Note: You need to have a class that is *registered as a sylius_resource*. It can be for example a `CarRentalOrderClass`.

- Make your `CarRentalOrder` class implement the `PromotionSubjectInterface`.

Put its configuration inside your `app/config/config.yml`.

```yaml
# app/config/config.yml
sylius_promotion:
    resources:
        promotion_subject:
            classes:
                model: AppBundle\Entity\CarRentalOrder
```

And configure doctrine extensions which are used by the bundle.

```yaml
# app/config/config.yml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Congratulations! The bundle is now installed and ready to use.

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

All the models of this bundle are defined in `Sylius\Component\Promotion\Model`.

PromotionRule

A `PromotionRule` is used to check if your order is eligible to the promotion. A promotion can have none, one or several rules. `SyliusPromotionBundle` comes with 2 types of rules :

- cart quantity rule : quantity of the order is checked
- item total rule : the amount of the order is checked

A rule is configured via the `configuration` attribute which is an array serialized into database. For cart quantity rules, you have to configure the `count` key, whereas the `amount` key is used for item total rules. Configuration is always strict, which means, that if you set `count` to **4** for cart quantity rule, orders with equal or more than **4** quantity will be eligible.

PromotionAction

An `PromotionAction` defines the nature of the discount. Common actions are :

- percentage discount
- fixed amount discount

An action is configured via the `configuration` attribute which is an array serialized into database. For percentage discount actions, you have to configure the `percentage` key, whereas the `amount` key is used for fixed discount rules.

PromotionCoupon

A `PromotionCoupon` is a ticket having a `code` that can be exchanged for a financial discount. A promotion can have none, one or several coupons.

A coupon is considered as valid if the method `isValid()` returns `true`. This method checks the number of times this coupon can be used (attribute `usageLimit`), the number of times this has already been used (attribute `used`) and the coupon expiration date (attribute `expiresAt`). If `usageLimit` is not set, the coupon will be usable an unlimited times.

PromotionSubjectInterface

A `PromotionSubjectInterface` is the object you want to apply the promotion on. For instance, in Sylius Standard, a `Sylius\Component\Core\Model\Order` can be subject to promotions.

By implementing `PromotionSubjectInterface`, your object will have to define the following methods : - `getPromotionSubjectItemTotal()` should return the amount of your order - `getPromotionSubjectItemCount()` should return the number of items of your order - `getPromotionCoupon()` should return the coupon linked to your order. If you do not want to use coupon, simply return `null`.

Promotion

The `Promotion` is the main model of this bundle. A promotion has a `name`, a `description` and :

- can have none, one or several rules

- should have at least one action to be effective

- can be based on coupons

- can have a limited number of usages by using the attributes `usageLimit` and `used`. When `used` reaches `usageLimit` the promotion is no longer valid. If `usageLimit` is not set, the promotion will be usable an unlimited times.

- can be limited by time by using the attributes `startsAt` and `endsAt`

How are rules checked?

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Everything related to this subject is located in `Sylius\Component\Promotion\Checker`.

Rule checkers

New rules can be created by implementing `RuleCheckerInterface`. This interface provides the method `isEligible` which aims to determine if the promotion subject respects the current rule or not.

I told you before that `SyliusPromotionBundle` ships with 2 types of rules : cart quantity rule and item total rule.

Cart quantity rule is defined via the service `sylius.promotion_rule_checker.cart_quantity` which uses the class `CartQuantityRuleChecker`. The method `isEligible` checks here if the promotion subject has the minimum quantity (method `getPromotionSubjectItemCount()` of `PromotionSubjectInterface`) required by the rule.

Item total rule is defined via the service `sylius.promotion_rule_checker.item_total` which uses the class `ItemTotalRuleChecker`. The method `isEligible` checks here if the promotion subject has the minimum amount (method `getPromotionSubjectItemTotal()` of `PromotionSubjectInterface`) required by the rule.

The promotion eligibility checker service

To be eligible to a promotion, a subject must :

1. respect all the rules related to the promotion

2. respect promotion dates if promotion is limited by time

3. respect promotions usages count if promotion has a limited number of usages

4. if a coupon is provided with this order, it must be valid and belong to this promotion

The service `sylius.promotion_eligibility_checker` checks all these constraints for you with the method `isEligible()` which returns `true` or `false`. This service uses the class `CompositePromotionEligibilityChecker`.

How actions are applied?

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Everything related to this subject is located in `Sylius\Component\Promotion\Action`.

Actions

Actions can be created by implementing `PromotionActionCommandInterface`. This interface provides the method `execute` which aim is to apply a promotion to its subject. It also provides the method `getConfigurationFormType` which has to return the form name related to this action.

Actions have to be defined as services and have to use the tag named `sylius.promotion_action` with the attributes `type` and `label`.

As `SyliusPromotionBundle` is totally independent, it does not provide actions out of the box.

> **Note:** `Sylius\Component\Core\Promotion\Action\FixedDiscountPromotionActionCommand` from `Sylius/Sylius-Standard` is an example of action for a fixed amount discount. The related service is called `sylius.promotion_action.fixed_discount`.

> **Note:** `Sylius\Component\Core\Promotion\Action\PercentageDiscountPromotionActionCommand` from `Sylius/Sylius-Standard` is an example of action for a discount based on percentage. The related service is called `sylius.promotion_action.percentage_discount`.

Learn more about actions in the *promotions concept documentation* and in the *Cookbook*.

Applying actions to promotions

We have seen above how actions can be created. Now let's see how they are applied to their subject.

The `PromotionApplicator` is responsible of this via its method `apply`. This method will `execute` all the registered actions of a promotion on a subject.

How promotions are applied?

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

By using the *promotion eligibility checker* and the *promotion applicator checker* services, the promotion processor applies all the possible promotions on a subject.

The promotion processor is defined via the service `sylius.promotion_processor` which uses the class `Sylius\Component\Promotion\Processor\PromotionProcessor`. Basically, it calls the method `apply` of the promotion applicator for all the active promotions that are eligible to the given subject.

Coupon based promotions

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Coupon based promotions require special needs that are covered by this documentation.

Coupon generator

`SyliusPromotionBundle` provides a way of generating coupons for a promotion : the coupon generator. Provided as a service `sylius.promotion_coupon_generator` via the class `Sylius\Component\Promotion\Generator\PromotionCouponGenerator`, its goal is to generate unique coupon codes.

PromotionCoupon controller

The `Sylius\Bundle\PromotionBundle\Controller\PromotionCouponController` provides a method for generating new coupons.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

```yaml
sylius_promotion:
    driver: doctrine/orm
    resources:
        promotion_subject:
            classes:
                model: Sylius\Component\Core\Model\Order
        promotion:
            classes:
                model:      Sylius\Component\Promotion\Model\Promotion
                interface:  Sylius\Component\Promotion\Model\PromotionInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\PromotionBundle\Form\Type\PromotionType
        promotion_rule:
            classes:
                factory: Sylius\Component\Core\Factory\PromotionRuleFactory
                model: Sylius\Component\Promotion\Model\PromotionRule
                interface: Sylius\Component\Promotion\Model\PromotionRuleInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\PromotionBundle\Form\Type\PromotionRuleType
        promotion_coupon:
```

```
        classes:
            model:      Sylius\Component\Promotion\Model\PromotionAction
            interface:  Sylius\Component\Promotion\Model\PromotionActionInterface
            controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
            repository: ~
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\PromotionBundle\Form\Type\PromotionActionType
    promotion_action:
        classes:
            model:      Sylius\Component\Promotion\Model\Coupon
            interface:  Sylius\Component\Promotion\Model\CouponInterface
            controller:␣
→Sylius\Bundle\PromotionBundle\Controller\PromotionCouponController
            repository: ~
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\PromotionBundle\Form\Type\PromotionActionType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Promotions in the Sylius platform* - concept documentation

SyliusResourceBundle

There are plenty of things you need to handle for every single Resource in your web application.

Several "Admin Generators" are available for Symfony, but we needed something really simple, that will allow us to have reusable controllers but preserve the performance and standard Symfony workflow. We did not want to generate any code or write "Admin" class definitions in PHP. The big goal was to have exactly the same workflow as with writing controllers manually but without actually creating them!

Another idea was not to limit ourselves to a single persistence backend. `Resource` component provides us with generic purpose persistence services and you can use this bundle with multiple persistence backends. So far we support:

- Doctrine ORM
- Doctrine MongoDB ODM
- Doctrine PHPCR ODM
- InMemory
- ElasticSearch (via an extension)

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your `composer.json` and download the package.

If you have Composer installed globally.

```
$ composer require sylius/resource-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/resource-bundle
```

Adding Required Bundles to The Kernel

You need to enable the bundle and its dependencies in the kernel:

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Bazinga\Bundle\HateoasBundle\BazingaHateoasBundle(),
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
    );
}
```

That's it! Now you can configure your first resource.

Configuring Your Resources

Now you need to configure your first resource. Let's assume you have a *Book* entity in your application and it has simple fields:

- id
- title
- author
- description

Tip: You can see a full exemplary configuration of a typical resource *here, in the "How to add a custom model?" cookbook*.

Implement the ResourceInterface in your model class.

```php
<?php

namespace AppBundle\Entity;

use Sylius\Component\Resource\Model\ResourceInterface;
```

(continues on next page)

```php
class Book implements ResourceInterface
{
    // Most of the time you have the code below already in your class.
    protected $id;

    public function getId()
    {
        return $this->id;
    }
}
```

Configure the class as a resource.

In your `app/config/config.yml` add:

```yaml
sylius_resource:
    resources:
        app.book:
            classes:
                model: AppBundle\Entity\Book
```

That's it! Your Book entity is now registered as Sylius Resource.

You can also configure several doctrine drivers.

Note: Remember that the `doctrine/orm` driver is used by default.

```yaml
sylius_resource:
    drivers:
        - doctrine/orm
        - doctrine/phpcr-odm
    resources:
        app.book:
            classes:
                model: AppBundle\Entity\Book
        app.article:
            driver: doctrine/phpcr-odm
            classes:
                model: AppBundle\Document\ArticleDocument
```

Generate API routing.

Tip: Learn more about using Sylius REST API in these articles: *REST API Reference*, *How to use Sylius API? - Cookbook*.

Add the following lines to `app/config/routing.yml`:

```
app_book:
    resource: |
        alias: app.book
    type: sylius.resource_api
```

After that a full JSON/XML CRUD API is ready to use. Sounds crazy? Spin up the built-in server and give it a try:

```
$ php bin/console server:run
```

You should see something like:

```
Server running on http://127.0.0.1:8000

Quit the server with CONTROL-C.
```

Now, in a separate Terminal window, call these commands:

```
$ curl -i -X POST -H "Content-Type: application/json" -d '{"title": "Lord of The Rings
↪", "author": "J. R. R. Tolkien", "description": "Amazing!"}' http://localhost:8000/
↪books/
$ curl -i -X GET -H "Accept: application/json" http://localhost:8000/books/
```

As you can guess, other CRUD actions are available through this API.

Generate web routing.

What if you want to render HTML pages? That's easy! Update the routing configuration:

```
app_book:
    resource: |
        alias: app.book
    type: sylius.resource
```

This will generate routing for HTML views.

Run the `debug:router` command to see available routes:

```
$ php bin/console debug:router

---------------------- --------------- -------- ------ ------------------------
Name                   Method          Scheme   Host   Path
---------------------- --------------- -------- ------ ------------------------
app_book_show          GET             ANY      ANY    /books/{id}
app_book_index         GET             ANY      ANY    /books/
app_book_create        GET|POST        ANY      ANY    /books/new
app_book_update        GET|PUT|PATCH   ANY      ANY    /books/{id}/edit
app_book_delete        DELETE          ANY      ANY    /books/{id}
---------------------- --------------- -------- ------ ------------------------
```

Tip: Do you need **views** for your newly created entity? Read more about *Grids*, which are a separate bundle of Sylius, but may be very useful for views generation.

You can configure more options for the routing generation but you can also define each route manually to have it fully configurable. Continue reading to learn more!

Services

When you register an entity as a resource, several services are registered for you. For the `app.book` resource, the following services are available:

- `app.controller.book` - instance of `ResourceController`;

- `app.factory.book` - instance of *FactoryInterface*;

- `app.repository.book` - instance of *RepositoryInterface*;

- `app.manager.book` - alias to an appropriate Doctrine's `ObjectManager`.

Routing

SyliusResourceBundle ships with a custom route loader that can save you some time.

Generating Generic CRUD Routing

To generate a full CRUD routing, simply configure it in your `app/config/routing.yml`:

```yaml
app_book:
    resource: |
        alias: app.book
    type: sylius.resource
```

Results in the following routes:

```
$ php bin/console debug:router


----------------------- --------------- -------- ------ ------------------------
Name                    Method          Scheme   Host   Path
----------------------- --------------- -------- ------ ------------------------
app_book_show           GET             ANY      ANY    /books/{id}
app_book_index          GET             ANY      ANY    /books/
app_book_create         GET|POST        ANY      ANY    /books/new
app_book_update         GET|PUT|PATCH   ANY      ANY    /books/{id}/edit
app_book_delete         DELETE          ANY      ANY    /books/{id}
```

Using a Custom Path

By default, Sylius will use a plural form of the resource name, but you can easily customize the path:

```yaml
app_book:
    resource: |
        alias: app.book
        path: library
    type: sylius.resource
```

Results in the following routes:

```
$ php bin/console debug:router


----------------------- --------------- -------- ------ ------------------------
```

(continues on next page)

```
Name                    Method          Scheme   Host   Path
----------------------- --------------- -------- ------ -------------------------
app_book_show           GET             ANY      ANY    /library/{id}
app_book_index          GET             ANY      ANY    /library/
app_book_create         GET|POST        ANY      ANY    /library/new
app_book_update         GET|PUT|PATCH   ANY      ANY    /library/{id}/edit
app_book_delete         DELETE          ANY      ANY    /library/{id}
```

Generating API CRUD Routing

To generate a full API-friendly CRUD routing, add these YAML lines to your `app/config/routing.yml`:

```yaml
app_book:
    resource: |
        alias: app.book
    type: sylius.resource_api
```

Results in the following routes:

```
$ php bin/console debug:router


----------------------- --------------- -------- ------ -------------------------
Name                    Method          Scheme   Host   Path
----------------------- --------------- -------- ------ -------------------------
app_book_show           GET             ANY      ANY    /books/{id}
app_book_index          GET             ANY      ANY    /books/
app_book_create         POST            ANY      ANY    /books/
app_book_update         PUT|PATCH       ANY      ANY    /books/{id}
app_book_delete         DELETE          ANY      ANY    /books/{id}
```

Excluding Routes

If you want to skip some routes, simply use `except` configuration:

```yaml
app_book:
    resource: |
        alias: app.book
        except: ['delete', 'update']
    type: sylius.resource
```

Results in the following routes:

```
$ php bin/console debug:router


----------------------- --------------- -------- ------ -------------------------
Name                    Method          Scheme   Host   Path
----------------------- --------------- -------- ------ -------------------------
app_book_show           GET             ANY      ANY    /books/{id}
app_book_index          GET             ANY      ANY    /books/
app_book_create         GET|POST        ANY      ANY    /books/new
```

Generating Only Specific Routes

If you want to generate only some specific routes, simply use the `only` configuration:

```
app_book:
    resource: |
        alias: app.book
        only: ['show', 'index']
    type: sylius.resource
```

Results in the following routes:

```
$ php bin/console debug:router

---------------------- --------------- -------- ------ ------------------------
Name                   Method          Scheme   Host   Path
---------------------- --------------- -------- ------ ------------------------
app_book_show          GET             ANY      ANY    /books/{id}
app_book_index         GET             ANY      ANY    /books/
```

Generating Routing for a Section

Sometimes you want to generate routing for different "sections" of an application:

```
app_admin_book:
    resource: |
        alias: app.book
        section: admin
    type: sylius.resource
    prefix: /admin

app_library_book:
    resource: |
        alias: app.book
        section: library
        only: ['show', 'index']
    type: sylius.resource
    prefix: /library
```

The generation results in the following routes:

```
$ php bin/console debug:router

---------------------- --------------- -------- ------ ------------------------
Name                   Method          Scheme   Host   Path
---------------------- --------------- -------- ------ ------------------------
app_admin_book_show    GET             ANY      ANY    /admin/books/{id}
app_admin_book_index   GET             ANY      ANY    /admin/books/
app_admin_book_create  GET|POST        ANY      ANY    /admin/books/new
app_admin_book_update  GET|PUT|PATCH   ANY      ANY    /admin/books/{id}/edit
app_admin_book_delete  DELETE          ANY      ANY    /admin/books/{id}
app_library_book_show  GET             ANY      ANY    /library/books/{id}
app_library_book_index GET             ANY      ANY    /library/books/
```

Using Custom Templates

By default, `ResourceController` will use the templates namespace you have configured for the resource. You can easily change that per route, but it is also easy when you generate the routing:

```yaml
app_admin_book:
    resource: |
        alias: app.book
        section: admin
        templates: Admin/Book
    type: sylius.resource
    prefix: /admin
```

Following templates will be used for actions:

- `:app/Resources/views/Admin/Book:show.html.twig`

- `:app/Resources/views/Admin/Book:index.html.twig`

- `:app/Resources/views/Admin/Book:create.html.twig`

- `:app/Resources/views/Admin/Book:update.html.twig`

Using a Custom Form

If you want to use a custom form:

```yaml
app_book:
    resource: |
        alias: app.book
        form: AppBundle/Form/Type/AdminBookType
    type: sylius.resource
```

`create` and `update` actions will use AppBundle/Form/Type/AdminBookType form type.

Note: Remember, that if your form type has some dependencies you have to declare it as a service and tag with **name: form.type**. You can read more about it here

Using a Custom Redirect

By default, after successful resource creation or update, Sylius will redirect to the `show` route and fallback to `index` if it does not exist. If you want to change that behavior, use the following configuration:

```yaml
app_book:
    resource: |
        alias: app.book
        redirect: update
    type: sylius.resource
```

API Versioning

One of the ResourceBundle dependencies is JMSSerializer, which provides a useful functionality of object versioning. It is possible to take an advantage of it almost out of the box. If you would like to return only the second version of

your object serializations, use the following snippet:

```
app_book:
    resource: |
        alias: app.book
        serialization_version: 2
    type: sylius.resource_api
```

What is more, you can use a path variable to dynamically change your request. You can achieve this by setting a path prefix when importing file or specify it in the path option.

```
app_book:
    resource: |
        alias: app.book
        serialization_version: $version
    type: sylius.resource_api
```

Note: Remember that a dynamically resolved *books* prefix is no longer available when you specify `path`, and it has to be defined manually.

Using a Custom Criteria

Sometimes it is convenient to add some additional constraint when resolving resources. For example, one could want to present a list of all books from some library (which id would be a part of path). Assuming that the path prefix is */libraries/{libraryId}*, if you would like to list all books from this library, you could use the following snippet:

```
app_book:
    resource: |
        alias: app.book
        criteria:
            library: $libraryId
    type: sylius.resource
```

Which will result in the following routes:

```
$ php bin/console debug:router

----------------------- --------------- -------- ------ ----------------------------
↪----------
Name                    Method          Scheme   Host   Path
----------------------- --------------- -------- ------ ----------------------------
↪----------
app_book_show           GET             ANY      ANY    /libraries/{libraryId}/books/
↪{id}
app_book_index          GET             ANY      ANY    /libraries/{libraryId}/books/
app_book_create         GET|POST        ANY      ANY    /libraries/{libraryId}/books/
↪new
app_book_update         GET|PUT|PATCH   ANY      ANY    /libraries/{libraryId}/books/
↪{id}/edit
app_book_delete         DELETE          ANY      ANY    /libraries/{libraryId}/books/
↪{id}
```

Using a Custom Identifier

As you could notice the generated routing resolves resources by the id field. But sometimes it is more convenient to use a custom identifier field instead, let's say a code (or any other field of your choice which can uniquely identify your resource). If you want to look for books by isbn, use the following configuration:

```yaml
app_book:
    resource: |
        identifier: isbn
        alias: app.book
        criteria:
            isbn: $isbn
    type: sylius.resource
```

Which will result in the following routes:

```
$ php bin/console debug:router


---------------------- --------------- -------- ------ ----------------------
Name                    Method          Scheme   Host   Path
---------------------- --------------- -------- ------ ----------------------
app_book_show           GET             ANY      ANY    /books/{isbn}
app_book_index          GET             ANY      ANY    /books/
app_book_create         GET|POST        ANY      ANY    /books/new
app_book_update         GET|PUT|PATCH   ANY      ANY    /books/{isbn}/edit
app_book_delete         DELETE          ANY      ANY    /books/{isbn}
```

Forms

Have you noticed how Sylius generates forms for you? Of course, for many use-cases you may want to create a custom form.

Custom Resource Form

Create a FormType class for your resource

```php
<?php

namespace AppBundle\Form\Type;

use Sylius\Bundle\ResourceBundle\Form\Type\AbstractResourceType;
use Symfony\Component\Form\FormBuilderInterface;

class BookType extends AbstractResourceType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        // Build your custom form, with all fields that you need
        $builder->add('title', TextType::class);
    }
```

(continues on next page)

```php
    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix()
    {
        return 'app_book';
    }
}
```

Note: The getBlockPrefix method returns the prefix of the template block name for this type.

Register the FormType as a service

Warning: the registration of a form type is only needed when the form is extending the `AbstractResourceType` or when it has some custom constructor dependencies.

```yaml
app.book.form.type:
    class: AppBundle\Form\Type\BookType
    tags:
        - { name: form.type }
    arguments: ['%app.model.book.class%', '%app.book.form.type.validation_groups%']
```

Configure the form for your resource

```yaml
sylius_resource:
    resources:
        app.book:
            classes:
                model: AppBundle\Entity\Book
                form: AppBundle\Form\Type\BookType
```

That's it. Your new class will be used for all forms!

Getting a Single Resource

Your newly created controller service supports basic CRUD operations and is configurable via routing.

The simplest action is **showAction**. It is used to display a single resource. To use it, the only thing you need to do is register a proper route.

Let's assume that you have a app.book resource registered. To display a single Book, define the following routing:

```yaml
# app/config/routing.yml

app_book_show:
    path: /books/{id}
```

```
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
```

Done! Now when you go to `/books/3`, ResourceController will use the repository (`app.repository.book`) to find a Book with the given id (3). If the requested book resource does not exist, it will throw a `404 Not Found` exception.

When a Book is found, the default template will be rendered - `App:Book:show.html.twig` (like you configured it in the `config.yml`) with the Book result as the `book` variable. That's the most basic usage of the simple `showAction`.

Using a Custom Template

Okay, but what if you want to display the same Book resource, but with a different representation in a view?

```
# routing.yml

app_admin_book_show:
    path: /admin/books/{id}
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
        _sylius:
            template: Admin/Book/show.html.twig
```

Nothing more to do here, when you go to `/admin/books/3`, the controller will try to find the Book and render it using the custom template you specified under the route configuration. Simple, isn't it?

Overriding Default Criteria

Displaying books by id can be boring... and let's say we do not want to allow viewing disabled books. There is a solution for that!

```
# routing.yml

app_book_show:
    path: /books/{title}
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
        _sylius:
            criteria:
                title: $title
                enabled: true
```

With this configuration, the controller will look for a book with the given title and exclude disabled books. Internally, it simply uses the `$repository->findOneBy(array $criteria)` method to look for the resource.

Using Custom Repository Methods

By default, resource repository uses **findOneBy(array $criteria)**, but in some cases it's not enough - for example - you want to do proper joins or use a custom query. Creating yet another action to change the method called could be

a solution but there is a better way. The configuration below will use a custom repository method to get the resource.

```yaml
# routing.yml

app_book_show:
    path: /books/{author}
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
        _sylius:
            repository:
                method: findOneNewestByAuthor
                arguments: [$author]
```

Internally, it simply uses the `$repository->findOneNewestByAuthor($author)` method, where `author` is taken from the current request.

Using Custom Repository Service

If you would like to use your own service to get the resource, then try the following configuration:

```yaml
# app/config/routing.yml

app_book_show:
    path: /books/{author}
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
        _sylius:
            repository:
                method: ["expr:service('app.repository.custom_book_repository')",
"findOneNewestByAuthor"]
                arguments: [$author]
```

With this configuration, method `findOneNewestByAuthor` from service with ID `app.repository.custom_book_repository` will be called to get the resource.

Configuration Reference

```yaml
# routing.yml

app_book_show:
    path: /books/{author}
    methods: [GET]
    defaults:
        _controller: app.controller.book:showAction
        _sylius:
            template: Book/show.html.twig
            repository:
                method: findOneNewestByAuthor
                arguments: [$author]
            criteria:
                enabled: true
            serialization_groups: [Custom, Details]
            serialization_version: 1.0.2
```

Getting a Collection of Resources

To get a paginated list of Books, we will use **indexAction** of our controller. In the default scenario, it will return an instance of paginator, with a list of Books.

```yaml
# app/config/routing.yml

app_book_index:
    path: /books
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
```

When you go to `/books`, the ResourceController will use the repository (`app.repository.book`) to create a paginator. The default template will be rendered - `App:Book:index.html.twig` with the paginator as the `books` variable.

A paginator can be a simple array, if you disable the pagination, otherwise it is an instance of `Pagerfanta\Pagerfanta` which is a library used to manage the pagination.

Overriding the Template and Criteria

Just like for the **showAction**, you can override the default template and criteria.

```yaml
# app/config/routing.yml

app_book_index_inactive:
    path: /books/disabled
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
        _sylius:
            criteria:
                enabled: false
            template: Book/disabled.html.twig
```

This action will render a custom template with a paginator only for disabled Books.

Sorting

Except filtering, you can also sort Books.

```yaml
# app/config/routing.yml

app_book_index_top:
    path: /books/top
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
        _sylius:
            sortable: true
            sorting:
                score: desc
            template: Book/top.html.twig
```

Under that route, you can paginate over the Books by their score.

Using a Custom Repository Method

You can define your own repository method too, you can use the same way explained in show_resource.

Note: If you want to paginate your resources you need to use `EntityRepository::getPaginator($queryBuilder)`. It will transform your doctrine query builder into `Pagerfanta\Pagerfanta` object.

Changing the "Max Per Page" Option of Paginator

You can also control the "max per page" for paginator, using `paginate` parameter.

```yaml
# app/config/routing.yml

app_book_index_top:
    path: /books/top
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
        _sylius:
            paginate: 5
            sortable: true
            sorting:
                score: desc
            template: Book/top.html.twig
```

This will paginate 5 books per page, where 10 is the default.

Disabling Pagination - Getting a Simple Collection

Pagination is handy, but you do not always want to do it, you can disable pagination and simply request a collection of resources.

```yaml
# app/config/routing.yml

app_book_index_top3:
    path: /books/top
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
        _sylius:
            paginate: false
            limit: 3
            sortable: true
            sorting:
                score: desc
            template: Book/top3.html.twig
```

That action will return the top 3 books by score, as the `books` variable.

Configuration Reference

```yaml
# app/config/routing.yml

app_book_index:
    path: /{author}/books
    methods: [GET]
    defaults:
        _controller: app.controller.book:indexAction
        _sylius:
            template: Author/books.html.twig
            repository:
                method: createPaginatorByAuthor
                arguments: [$author]
            criteria:
                enabled: true
                author.name: $author
            paginate: false # Or: 50
            limit: 100 # Or: false
            serialization_groups: [Custom, Details]
            serialization_version: 1.0.2
```

Creating Resources

To display a form, handle its submission or to create a new resource via API, you should use the **createAction** of your **app.controller.book** service.

```yaml
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
```

Done! Now when you go to /books/new, the ResourceController will use the factory (app.factory.book) to create a new book instance. Then it will try to create an app_book form, and set the newly created book as its data.

Submitting the Form

You can use exactly the same route to handle the submit of the form and create the book.

```html
<form method="post" action="{{ path('app_book_create') }}">
```

On submit, the create action with method POST, will bind the request on the form, and if it is valid it will use the right manager to persist the resource. Then, by default it redirects to app_book_show to display the created book, but you can easily change that behavior - you'll see this in further sections.

When validation fails, it will render the form just like previously with the error messages displayed.

Changing the Template

Just like for the **show** and **index** actions, you can customize the template per route.

```
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            template: Book/create.html.twig
```

Using Custom Form

You can also use custom form type on per route basis. Following Symfony3 conventions forms types are resolved by FQCN. Below you can see the usage for specifying a custom form.

```
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            form: AppBundle\Form\BookType
```

Passing Custom Options to Form

What happens when you need pass some options to the form? Well, there's a configuration for that!

Below you can see the usage for specifying custom options, in this case, `validation_groups`, but you can pass any option accepted by the form.

```
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            form:
                type: app_book_custom
                options:
                    validation_groups: [sylius, my_custom_group]
```

Using Custom Factory Method

By default, `ResourceController` will use the `createNew` method with no arguments to create a new instance of your object. However, this behavior can be modified. To use a different method of your factory, you can simply configure the `factory` option.

```
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            factory: createNewWithAuthor
```

Additionally, if you want to provide your custom method with arguments from the request, you can do so by adding more parameters.

```
# app/config/routing.yml

app_book_create:
    path: /books/{author}/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            factory:
                method: createNewWithAuthor
                arguments: [$author]
```

With this configuration, `$factory->createNewWithAuthor($request->get('author'))` will be called to create new resource within the `createAction`.

Using Custom Factory Service

If you would like to use your own service to create the resource, then try the following configuration:

```
# app/config/routing.yml

app_book_create:
    path: /{authorId}/books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            factory:
                method: ["expr:service('app.factory.custom_book_factory')",
→"createNewByAuthorId"]
                arguments: $authorId
```

With this configuration, service with id "app.factory.custom_book_factory" will be called to create new resource within the `createNewByAuthorId` method and the author id from the url as argument.

Custom Redirect After Success

By default the controller will try to get the id of the newly created resource and redirect to the "show" route. You can easily change that behaviour. For example, to redirect to the index list after successfully creating a new resource - you can use the following configuration.

```yaml
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            redirect: app_book_index
```

You can also perform more complex redirects, with parameters. For example:

```yaml
# app/config/routing.yml

app_book_create:
    path: /genre/{genreId}/books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            redirect:
                route: app_genre_show
                parameters: { id: $genreId }
```

In addition to the request parameters, you can access some of the newly created objects properties, using the `resource.` prefix.

```yaml
# app/config/routing.yml

app_book_create:
    path: /books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            redirect:
                route: app_book_show
                parameters: { title: resource.title }
```

With this configuration, the `title` parameter for route `app_book_show` will be obtained from your newly created book.

Custom Event Name

By default, there are two events dispatched during resource creation, one before adding it do database, the other after successful addition. The pattern is always the same - `{applicationName}.{resourceName}.pre/ post_create`. However, you can customize the last part of the event, to provide your own action name.

```yaml
# app/config/routing.yml

app_book_customer_create:
    path: /customer/books/new
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
```

(continues on next page)

```
        _sylius:
            event: customer_create
```

This way, you can listen to `app.book.pre_customer_create` and `app.book.post_customer_create` events. It's especially useful, when you use `ResourceController:createAction` in more than one route.

Configuration Reference

```yaml
# app/config/routing.yml

app_genre_book_add:
    path: /{genreName}/books/add
    methods: [GET, POST]
    defaults:
        _controller: app.controller.book:createAction
        _sylius:
            template: Book/addToGenre.html.twig
            form: app_new_book
            event: book_create
            factory:
                method: createForGenre
                arguments: [$genreName]
            criteria:
                group.name: $genreName
            redirect:
                route: app_book_show
                parameters: { title: resource.title }
```

Updating Resources

To display an edit form of a particular resource, change it or update it via API, you should use the **updateAction** action of your **app.controller.book** service.

```yaml
# app/config/routing.yml

app_book_update:
    path: /books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
```

Done! Now when you go to `/books/5/edit`, ResourceController will use the repository (`app.repository.book`) to find the book with id == **5**. If found it will create the `app_book` form, and set the existing book as data.

Submitting the Form

You can use exactly the same route to handle the submit of the form and updating the book.

```html
<form method="post" action="{{ path('app_book_update', {'id': book.id}) }}">
    <input type="hidden" name="_method" value="PUT" />
```

On submit, the update action with method PUT, will bind the request on the form, and if it is valid it will use the right manager to persist the resource. Then, by default it redirects to `app_book_show` to display the updated book, but like for creation of the resource - it's customizable.

When validation fails, it will simply render the form again, but with error messages.

Changing the Template

Just like for other actions, you can customize the template.

```yaml
# app/config/routing.yml

app_book_update:
    path: /books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            template: Admin/Book/update.html.twig
```

Using Custom Form

Same way like for **createAction** you can override the default form.

```yaml
# app/config/routing.yml

app_book_update:
    path: /books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            form: AppBundle\Form\BookType
```

Passing Custom Options to Form

Same way like for **createAction** you can pass options to the form.

Below you can see how to specify custom options, in this case, `validation_groups`, but you can pass any option accepted by the form.

```yaml
# app/config/routing.yml

app_book_update:
    path: /books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            form:
                type: app_book_custom
                options:
                    validation_groups: [sylius, my_custom_group]
```

Overriding the Criteria

By default, the **updateAction** will look for the resource by id. You can easily change that criteria.

```
# app/config/routing.yml

app_book_update:
    path: /books/{title}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            criteria: { title: $title }
```

Custom Redirect After Success

By default the controller will try to get the id of resource and redirect to the "show" route. To change that, use the following configuration.

```
# app/config/routing.yml

app_book_update:
    path: /books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            redirect: app_book_index
```

You can also perform more complex redirects, with parameters. For example:

```
# app/config/routing.yml

app_book_update:
    path: /genre/{genreId}/books/{id}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            redirect:
                route: app_genre_show
                parameters: { id: $genreId }
```

Custom Event Name

By default, there are two events dispatched during resource update, one before setting new data, the other after successful update. The pattern is always the same - {applicationName}.{resourceName}.pre/post_update. However, you can customize the last part of the event, to provide your own action name.

```
# app/config/routing.yml

app_book_customer_update:
    path: /customer/book-update/{id}
```

(continues on next page)

```
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            event: customer_update
```

This way, you can listen to `app.book.pre_customer_update` and `app.book.post_customer_update` events. It's especially useful, when you use `ResourceController:updateAction` in more than one route.

[API] Returning resource or no content

Depending on your app approach it can be useful to return a changed object or only the `204 HTTP Code`, which indicates that everything worked smoothly. Sylius, by default is returning the `204 HTTP Code`, which indicates an empty response. If you would like to receive a whole object as a response you should set a *return_content* option to true.

```
# app/config/routing.yml

app_book_update:
    path: /books/{title}/edit
    methods: [GET, PUT]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            criteria: { title: $title }
            return_content: true
```

Warning: The *return_content* flag is available for the *applyStateMachineTransitionAction* method as well. But these are the only ones which can be configured this way. It is worth noticing, that the *applyStateMachineTransitionAction* returns a default *200 HTTP Code* response with a fully serialized object.

Configuration Reference

```
# app/config/routing.yml

app_book_update:
    path: /genre/{genreId}/books/{title}/edit
    methods: [GET, PUT, PATCH]
    defaults:
        _controller: app.controller.book:updateAction
        _sylius:
            template: Book/editInGenre.html.twig
            form: app_book_custom
            event: book_update
            repository:
                method: findBookByTitle
                arguments: [$title, expr:service('app.context.book')]
            criteria:
                enabled: true
                genreId: $genreId
            redirect:
```

```
                    route: app_book_show
                    parameters: { title: resource.title }
            return_content: true
```

Deleting Resources

Deleting a resource is simple.

```yaml
# app/config/routing.yml

app_book_delete:
    path: /books/{id}
    methods: [DELETE]
    defaults:
        _controller: app.controller.book:deleteAction
```

Calling an Action with DELETE method

Currently browsers do not support the "DELETE" http method. Fortunately, Symfony has a very useful feature. You can make a POST call with parameter override, which will force the framework to treat the request as the specified method.

```html
<form method="post" action="{{ path('app_book_delete', {'id': book.id}) }}">
    <input type="hidden" name="_method" value="DELETE" />
    <button type="submit">
        Delete
    </button>
</form>
```

On submit, the delete action with the method DELETE, will remove and flush the resource. Then, by default it redirects to app_book_index to display the books index, but just like for the other actions - it's customizable.

Overriding the Criteria

By default, the **deleteAction** will look for the resource by id. However, you can easily change that. For example, if you want to delete a book that belongs to a particular genre, not only by its id.

```yaml
# app/config/routing.yml

app_book_delete:
    path: /genre/{genreId}/books/{id}
    methods: [DELETE]
    defaults:
        _controller: app.controller.book:deleteAction
        _sylius:
            criteria:
                id: $id
                genre: $genreId
```

There are no magic hacks behind that, it simply takes parameters from request and builds the criteria array for the findOneBy repository method.

Custom Redirect After Success

By default the controller will redirect to the "index" route after successful action. To change that, use the following configuration.

```yaml
# app/config/routing.yml

app_book_delete:
    path: /genre/{genreId}/books/{id}
    methods: [DELETE]
    defaults:
        _controller: app.controller.book:deleteAction
        _sylius:
            redirect:
                route: app_genre_show
                parameters: { id: $genreId }
```

Custom Event Name

By default, there are two events dispatched during resource deletion, one before removing, the other after successful removal. The pattern is always the same - {applicationName}.{resourceName}.pre/post_delete. However, you can customize the last part of the event, to provide your own action name.

```yaml
# app/config/routing.yml

app_book_customer_delete:
    path: /customer/book-delete/{id}
    methods: [DELETE]
    defaults:
        _controller: app.controller.book:deleteAction
        _sylius:
            event: customer_delete
```

This way, you can listen to app.book.pre_customer_delete and app.book.post_customer_delete events. It's especially useful, when you use ResourceController:deleteAction in more than one route.

Configuration Reference

```yaml
# app/config/routing.yml

app_genre_book_remove:
    path: /{genreName}/books/{id}/remove
    methods: [DELETE]
    defaults:
        _controller: app.controller.book:deleteAction
        _sylius:
            event: book_delete
            repository:
                method: findByGenreNameAndId
                arguments: [$genreName, $id]
            criteria:
                genre.name: $genreName
                id: $id
```

(continues on next page)

```
        redirect:
            route: app_genre_show
            parameters: { genreName: $genreName }
```

Configuration Reference

```
sylius_resource:
    resources:
        app.book:
            driver: doctrine/orm
            classes:
                model: # Required!
                interface: ~
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\ResourceBundle\Form\Type\DefaultResourceType
                    validation_groups: [sylius]
            options:
                object_manager: default
            templates:
                form: Book/_form.html.twig
            translation:
                classes:
                    model: ~
                    interface: ~
                    controller:␣
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    repository: ~
                    factory: Sylius\Component\Resource\Factory\Factory
                    form: Sylius\Bundle\ResourceBundle\Form\Type\DefaultResourceType
                        validation_groups: [sylius]
                templates:
                    form: Book/Translation/_form.html.twig
                options: ~
```

Routing Generator Configuration Reference

```
app_book:
    resource: |
        alias: app.book
        path: library
        identifier: code
        criteria:
            code: $code
        section: admin
        templates: :Book
        form: AppBundle/Form/Type/SimpleBookType
        redirect: create
        except: ['show']
        only: ['create', 'index']
        serialization_version: 1
    type: sylius.resource
```

Learn more

- *Resource Layer in the Sylius platform* - concept documentation

SyliusShippingBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

SyliusShippingBundle is the shipment management component for Symfony e-commerce applications.

If you need to manage shipments, shipping methods and deal with complex cost calculation, this bundle can help you a lot!

Your products or whatever you need to deliver, can be categorized under unlimited set of categories. You can display appropriate shipping methods available to the user, based on object category, weight, dimensions and anything you can imagine.

Flexible shipping cost calculation system allows you to create your own calculator services.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/shipping-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/shipping-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
```

(continues on next page)

```php
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Sylius\Bundle\ShippingBundle\SyliusShippingBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
sylius_shipping:
    driver: doctrine/orm # Configure the Doctrine ORM driver used in documentation.
```

Configure doctrine extensions which are used by this bundle.

```yaml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Routing configuration

Add the following to your `app/config/routing.yml`.

```yaml
sylius_shipping:
    resource: "@SyliusShipping/Resources/config/routing.yml"
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

Warning: This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

The ShippableInterface

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In order to handle your merchandise through the Sylius shipping engine, your models need to implement **ShippableInterface**.

Implementing the interface

Let's assume that you have a **Book** entity in your application.

First step is to implement the simple interface, which contains few simple methods.

```php
namespace Acme\Bundle\ShopBundle\Entity;

use Sylius\Component\Shipping\Model\ShippableInterface;
use Sylius\Component\Shipping\Model\ShippingCategoryInterface;

class Book implements ShippableInterface
{
    private $shippingCategory;

    public function getShippingCategory()
    {
        return $this->shippingCategory;
    }

    public function setShippingCategory(ShippingCategoryInterface $shippingCategory) /
→/ This method is not required.
    {
        $this->shippingCategory = $shippingCategory;

        return $this;
    }

    public function getShippingWeight()
    {
        // return integer representing the object weight.
    }

    public function getShippingWidth()
    {
        // return integer representing the book width.
    }

    public function getShippingHeight()
    {
        // return integer representing the book height.
    }

    public function getShippingDepth()
    {
        // return integer representing the book depth.
```

(continues on next page)

```
    }
}
```

Second and last task is to define the relation inside `Resources/config/doctrine/Book.orm.xml` of your
bundle.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://doctrine-project.org/schemas/orm/
↪doctrine-mapping
                                      http://doctrine-project.org/schemas/orm/
↪doctrine-mapping.xsd">

    <entity name="Acme\ShopBundle\Entity\Book" table="acme_book">
        <!-- your mappings... -->

        <many-to-one field="shippingCategory" target-entity=
↪"Sylius\Bundle\ShippingBundle\Model\ShippingCategoryInterface">
            <join-column name="shipping_category_id" referenced-column-name="id"␣
↪nullable="false" />
        </many-to-one>
    </entity>

</doctrine-mapping>
```

Done! Now your **Book** model can be used in Sylius shippingation engine.

Forms

If you want to add a shipping category selection field to your model form, simply use the
`sylius_shipping_category_choice` type.

```php
namespace Acme\ShopBundle\Form\Type;

use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\AbstractType;

class BookType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title', 'text')
            ->add('shippingCategory', 'sylius_shipping_category_choice')
        ;
    }
}
```

The ShippingSubjectInterface

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

The find available shipping methods or calculate shipping cost you need to use object implementing `ShippingSubjectInterface`.

The default **Shipment** model is already implementing `ShippingSubjectInterface`.

Interface methods

- The `getShippingMethod` returns a `ShippingMethodInterface` instance, representing the method.

- The `getShippingItemCount` provides you with the count of items to ship.

- The `getShippingItemTotal` returns the total value of shipment, if applicable. The default **Shipment** model returns 0.

- The `getShippingWeight` returns the total shipment weight.

- The `getShippables` returns a collection of unique `ShippableInterface` instances.

The Shipping Categories

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Every shippable object needs to have a shipping category assigned. The **ShippingCategory** model is extremely simple and described below.

Attribute	Description
id	Unique id of the shipping category
name	Name of the shipping category
description	Human friendly description of the classification
createdAt	Date when the category was created
updatedAt	Date of the last shipping category update

The Shipping Method

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ShippingMethod model represents the way that goods need to be shipped. An example of shipping method may be "DHL Express" or "FedEx World Shipping".

Attribute	Description
id	Unique id of the shipping method
name	Name of the shipping method
category	Reference to **ShippingCategory** (optional)
categoryRequirement	Category requirement
calculator	Name of the cost calculator
configuration	Configuration for the calculator
createdAt	Date when the method was created
updatedAt	Date of the last shipping method update

Calculating shipping cost

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Calculating shipping cost is as simple as using the `sylius.shipping_calculator` service and calling `calculate` method on `ShippingSubjectInterface`.

Let's calculate the cost of existing shipment.

```php
public function myAction()
{
    $calculator = $this->get('sylius.shipping_calculator');
    $shipment = $this->get('sylius.repository.shipment')->find(5);

    echo $calculator->calculate($shipment); // Returns price in cents. (integer)
}
```

What has happened?

- The delegating calculator gets the **ShippingMethod** from the **ShippingSubjectInterface** (Shipment).

- Appropriate **Calculator** instance is loaded, based on the **ShippingMethod.calculator** parameter.

- The `calculate(ShippingSubjectInterface, array $configuration)` is called, where configuration is taken from **ShippingMethod.configuration** attribute.

Default calculators

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Default calculators can be sufficient solution for many use cases.

Flat rate

The `flat_rate` calculator, charges concrete amount per shipment.

Per item rate

The `per_item_rate` calculator, charges concrete amount per shipment item.

More calculators

Depending on community contributions and Sylius resources, more default calculators can be implemented, for example `weight_range_rate`.

Custom calculators

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sylius ships with several default calculators, but you can easily register your own.

Simple calculators

All shipping cost calculators implement `CalculatorInterface`. In our example we'll create a calculator which calls an external API to obtain the shipping cost.

```php
# src/AppBundle/Shipping/Calculator/DHLCalculator.php
<?php

declare(strict_types=1);

namespace AppBundle\Shipping\Calculator;

use Sylius\Component\Shipping\Calculator\CalculatorInterface;
use Sylius\Component\Shipping\Model\ShipmentInterface;

final class DHLCalculator implements CalculatorInterface
{
    /**
     * @var DHLService
     */
    private $dhlService;

    /**
     * @param DHLService $dhlService
     */
    public function __construct(DHLService $dhlService)
    {
        $this->dhlService = $dhlService;
    }

    /**
     * {@inheritdoc}
     */
    public function calculate(ShipmentInterface $subject, array $configuration): int
    {
```

(continues on next page)

```php
        return $this->dhlService->getShippingCostForWeight($subject->
↪getShippingWeight());
    }

    /**
     * {@inheritdoc}
     */
    public function getType(): string
    {
        return 'dhl';
    }
}
```

Now, you need to register your new service in container and tag it with `sylius.shipping_calculator`.

```yaml
services:
    app.shipping_calculator.dhl:
        class: AppBundle\Shipping\Calculator\DHLCalculator
        arguments: ['@app.dhl_service']
        tags:
            - { name: sylius.shipping_calculator, calculator: dhl, label: "DHL" }
```

That would be all. This new option ("DHL") will appear on the **ShippingMethod** creation form, in the "calculator" field.

Configurable calculators

You can also create configurable calculators, meaning that you can have several **ShippingMethod**'s using same type of calculator, with different settings.

Let's modify the **DHLCalculator**, so that it charges 0 if shipping more than X items. First step is to create a form type which will be displayed if our calculator is selected.

```php
# src/AppBundle/Form/Type/Shipping/Calculator/DHLConfigurationType.php
<?php

declare(strict_types=1);

namespace AppBundle\Form\Type\Shipping\Calculator;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\IntegerType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Constraints\Type;

final class DHLConfigurationType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
```

```php
            ->add('limit', IntegerType::class, [
                'label' => 'Free shipping above total items',
                'constraints' => [
                    new NotBlank(),
                    new Type(['type' => 'integer']),
                ]
            ])
        ;
    }

    /**
     * {@inheritdoc}
     */
    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver
            ->setDefaults([
                'data_class' => null,
                'limit' => 10,
            ])
            ->setAllowedTypes('limit', 'integer')
        ;
    }

    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix(): string
    {
        return 'app_shipping_calculator_dhl';
    }
}
```

We also need to register the form type in the container and set this form type in the definition of the calculator.

```yaml
services:
    app.shipping_calculator.dhl:
        class: AppBundle\Shipping\Calculator\DHLCalculator
        arguments: ['@app.dhl_service']
        tags:
            - { name: sylius.shipping_calculator, calculator: dhl, form_type:
→AppBundle\Form\Type\Shipping\Calculator\DHLConfigurationType, label: "DHL" }

    app.form.type.shipping_calculator.dhl:
        class: AppBundle\Form\Type\Shipping\Calculator\DHLConfigurationType
        tags:
            - { name: form.type }
```

Perfect, now we're able to use the configuration inside the `calculate` method.

```php
# src/AppBundle/Shipping/Calculator/DHLCalculator.php
<?php

declare(strict_types=1);

namespace AppBundle\Shipping\Calculator;
```

```php
use Sylius\Component\Shipping\Calculator\CalculatorInterface;
use Sylius\Component\Shipping\Model\ShipmentInterface;

final class DHLCalculator implements CalculatorInterface
{
    /**
     * @var DHLService
     */
    private $dhlService;

    /**
     * @param DHLService $dhlService
     */
    public function __construct(DHLService $dhlService)
    {
        $this->dhlService = $dhlService;
    }

    /**
     * {@inheritdoc}
     */
    public function calculate(ShipmentInterface $subject, array $configuration): int
    {
        if ($subject->getShippingUnitCount() > $configuration['limit']) {
            return 0;
        }

        return $this->dhlService->getShippingCostForWeight($subject->
→getShippingWeight());
    }

    /**
     * {@inheritdoc}
     */
    public function getType(): string
    {
        return 'dhl';
    }
}
```

Your new configurable calculator is ready to use. When you select the "DHL" calculator in **ShippingMethod** form, configuration fields will appear automatically.

Resolving available shipping methods

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In many use cases, you want to decide which shipping methods are available for user. Sylius has a dedicated service which serves this purpose.

ShippingMethodsResolver

This service also works with the `ShippingSubjectInterface`. To get all shipping methods which support given subject, simply call the `getSupportedMethods` function.

```php
public function myAction()
{
    $resolver = $this->get('sylius.shipping_methods_resolver');
    $shipment = $this->get('sylius.repository.shipment')->find(5);

    foreach ($resolver->getSupportedMethods($shipment) as $method) {
        echo $method->getName();
    }
}
```

You can also pass the criteria array to initially filter the shipping methods pool.

```php
public function myAction()
{
    $country = $this->getUser()->getCountry();
    $resolver = $this->get('sylius.shipping_methods_resolver');
    $shipment = $this->get('sylius.repository.shipment')->find(5);

    foreach ($resolver->getSupportedMethods($shipment, array('country' => $country))
→as $method) {
        echo $method->getName();
    }
}
```

In forms

To display a select field with all the available methods for given subject, you can use the `sylius_shipping_method_choice` type. It supports two special options, required `subject` and optional `criteria`.

```php
<?php

class ShippingController extends Controller
{
    public function selectMethodAction(Request $request)
    {
        $shipment = $this->get('sylius.repository.shipment')->find(5);

        $form = $this->get('form.factory')->create(ShippingMethodChoiceType::class,
→null, array('subject' => $shipment));
    }
}
```

This form type internally calls the **ShippingMethodsResolver** service and creates a list of available methods.

Shipping method requirements

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sylius has a very flexible system for displaying only the right shipping methods to the user.

Shipping categories

Every **ShippableInterface** can hold a reference to **ShippingCategory**. The **ShippingSubjectInterface** (or **ShipmentInterface**) returns a collection of shippables.

ShippingMethod has an optional shipping category setting as well as **categoryRequirement** which has 3 options. If this setting is set to null, categories system is ignored.

"Match any" requirement

With this requirement, the shipping method will support any shipment (or shipping subject) which contains at least one shippable with the same category.

"Match all" requirement

All shippables have to reference the same category as the **ShippingMethod**.

"Match none" requirement

None of the shippables can have the same shipping category.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration Reference

```yaml
sylius_shipping:
    # The driver used for persistence layer.
    driver: ~
    classes:
        shipment:
            classes:
                model:      Sylius\Component\Shipping\Model\Shipment
                interface:  Sylius\Component\Shipping\Model\ShipmentInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
```

(continues on next page)

```
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\ShippingBundle\Form\Type\ShipmentType
    shipment_item:
        classes:
            model:      Sylius\Component\Shipping\Model\ShipmentItem
            interface:  Sylius\Component\Shipping\Model\ShipmentItemInterface
            controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
            repository: ~
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\ShippingBundle\Form\Type\ShipmentItemType
    shipping_method:
        classes:
            model:      Sylius\Component\Shipping\Model\ShippingMethod
            interface:  Sylius\Component\Shipping\Model\ShippingMethodInterface
            controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
            repository: ~
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\ShippingBundle\Form\Type\ShippingMethodType
        translation:
            classes:
                model:      ␣
→Sylius\Component\Shipping\Model\ShippingMethodTranslation
                interface:  ␣
→Sylius\Component\Shipping\Model\ShippingMethodTranslationInterface
                controller:␣
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\Factory
                form:␣
→Sylius\Bundle\ShippingBundle\Form\Type\ShippingMethodTranslationType
    shipping_category:
        classes:
            model:      Sylius\Component\Shipping\Model\ShippingCategory
            interface:  Sylius\Component\Shipping\Model\ShippingCategoryInterface
            controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
            repository: ~
            factory:    Sylius\Component\Resource\Factory\Factory
            form: Sylius\Bundle\ShippingBundle\Form\Type\ShippingCategoryType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Shipments in the Sylius platform* - concept documentation

SyliusTaxationBundle

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Calculating and applying taxes is a common task for most of ecommerce applications. **SyliusTaxationBundle** is a reusable taxation component for Symfony. You can integrate it into your existing application and enable the tax calculation logic for any model implementing the `TaxableInterface`.

It supports different tax categories and customizable tax calculators - you're able to easily implement your own calculator services. The default implementation handles tax included in and excluded from the price.

As with any Sylius bundle, you can override all the models, controllers, repositories, forms and services.

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with [Composer](), a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download package.

If you have [Composer installed globally]().

```
$ composer require sylius/taxation-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/taxation-bundle
```

Adding required bundles to the kernel

First, you need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Sylius\Bundle\TaxationBundle\SyliusTaxationBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Container configuration

Put this configuration inside your `app/config/config.yml`.

```yaml
sylius_taxation:
    driver: doctrine/orm # Configure the Doctrine ORM driver used in documentation.
```

And configure doctrine extensions which are used by this bundle:

```yaml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

The TaxableInterface

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In order to calculate the taxes for a model in your application, it needs to implement the `TaxableInterface` It is a very simple interface, with only one method - the `getTaxCategory()`, as every taxable has to belong to a specific tax category.

Implementing the interface

Let's assume that you have a **Server** entity in your application. Every server has it's price and other parameters, but you would like to calculate the tax included in price. You could calculate the math in a simple method, but it's not enough when you have to handle multiple tax rates, categories and zones.

First step is to implement the simple interface.

```php
namespace AcmeBundle\Entity;

use Sylius\Component\Taxation\Model\TaxCategoryInterface;
use Sylius\Component\Taxation\Model\TaxableInterface;

class Server implements TaxableInterface
{
    private $name;
```

(continues on next page)

```php
    private $taxCategory;

    public function getName()
    {
        return $this->name;
    }

    public function setName($name)
    {
        $this->name = $name;
    }

    public function getTaxCategory()
    {
        return $this->taxCategory;
    }

    public function setTaxCategory(TaxCategoryInterface $taxCategory) // This method
→is not required.
    {
        $this->taxCategory = $taxCategory;
    }
}
```

Second and last task is to define the relation inside `Resources/config/doctrine/Server.orm.xml` of your bundle.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://doctrine-project.org/schemas/orm/
→doctrine-mapping
                                    http://doctrine-project.org/schemas/orm/
→doctrine-mapping.xsd">

    <entity name="AcmeBundle\Entity\Server" table="acme_server">
        <!-- your mappings... -->

        <many-to-one field="taxCategory" target-entity=
→"Sylius\Component\Taxation\Model\TaxCategoryInterface">
            <join-column name="tax_category_id" referenced-column-name="id" nullable=
→"false" />
        </many-to-one>
    </entity>

</doctrine-mapping>
```

Done! Now your **Server** model can be used in Sylius taxation engine.

Forms

If you want to add a tax category selection field to your model form, simply use the `sylius_tax_category_choice` type.

```
namespace AcmeBundle\Form\Type;

use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\AbstractType;

class ServerType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', 'text')
            ->add('taxCategory', 'sylius_tax_category_choice')
        ;
    }

    public function getName()
    {
        return 'acme_server';
    }
}
```

Configuring taxation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

To start calculating taxes, we need to configure the system. In most cases, the configuration process is done via web interface, but you can also do it programatically.

Creating the tax categories

First step is to create a new tax category.

```php
<?php

public function configureAction()
{
    $taxCategoryFactory = $this->container->get('sylius.factory.tax_category');
    $taxCategoryManager = $this->container->get('sylius.manager.tax_category');

    $clothingTaxCategory = $taxCategoryFactory->createNew();
    $clothingTaxCategory->setName('Clothing');
    $clothingTaxCategory->setDescription('T-Shirts and this kind of stuff.');

    $foodTaxCategory = $taxCategoryFactory->createNew();
    $foodTaxCategory->setName('Food');
    $foodTaxCategory->setDescription('Yummy!');

    $taxCategoryManager->persist($clothingTaxCategory);
    $taxCategoryManager->persist($foodTaxCategory);
```

(continues on next page)

```php
    $taxCategoryManager->flush();
}
```

Categorizing the taxables

Second thing to do is to classify the taxables, in our example we'll get two products and assign the proper categories to them.

```php
<?php

public function configureAction()
{
    $tshirtProduct = // ...
    $bananaProduct = // ... Some logic behind loading entities.

    $taxCategoryRepository = $this->container->get('sylius.repository.tax_category');

    $clothingTaxCategory = $taxCategoryRepository->findOneBy(['name' => 'Clothing']);
    $foodTaxCategory = $taxCategoryRepository->findOneBy(['name' => 'Food']);

    $tshirtProduct->setTaxCategory($clothingTaxCategory);
    $bananaProduct->setTaxCategory($foodTaxCategory);

    // Save the product entities.
}
```

Configuring the tax rates

Finally, you have to create appropriate tax rates for each of categories.

```php
<?php

public function configureAction()
{
    $taxCategoryRepository = $this->container->get('sylius.repository.tax_category');

    $clothingTaxCategory = $taxCategoryRepository->findOneBy(['name' => 'Clothing']);
    $foodTaxCategory = $taxCategoryRepository->findOneBy(['name' => 'Food']);

    $taxRateFactory = $this->container->get('sylius.factory.tax_rate');
    $taxRateManager = $this->container->get('sylius.repository.tax_rate');

    $clothingTaxRate = $taxRateFactory->createNew();
    $clothingTaxRate->setCategory($clothingTaxCategory);
    $clothingTaxRate->setName('Clothing Tax');
    $clothingTaxRate->setCode('CT');
    $clothingTaxRate->setAmount(0.08);
    $clothingTaxRate->setCalculator('default');

    $foodTaxRate = $taxRateFactory->createNew();
    $foodTaxRate->setCategory($foodTaxCategory);
    $foodTaxRate->setName('Food');
    $foodTaxRate->setCode('F');
```

```
    $foodTaxRate->setAmount(0.12);
    $foodTaxRate->setCalculator('default');

    $taxRateManager->persist($clothingTaxRate);
    $taxRateManager->persist($foodTaxRate);

    $taxRateManager->flush();
}
```

Done! See the *"Calculating Taxes" chapter* to see how to apply these rates.

Calculating taxes

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

> **Warning:** When using the CoreBundle (i.e: full stack Sylius framework), the taxes are already calculated at each cart change. It is implemented by the `TaxationProcessor` class, which is called by the `OrderTaxationListener`.

In order to calculate tax amount for given taxable, we need to find out the applicable tax rate. The tax rate resolver service is available under `sylius.tax_rate_resolver` id, while the delegating tax calculator is accessible via `sylius.tax_calculator` name.

Resolving rate and using calculator

```php
<?php

namespace Acme\ShopBundle\Taxation

use Acme\ShopBundle\Entity\Order;
use Sylius\Bundle\TaxationBundle\Calculator\CalculatorInterface;
use Sylius\Bundle\TaxationBundle\Resolver\TaxRateResolverInterface;

class TaxApplicator
{
    private $calculator;
    private $taxRateResolver;

    public function __construct(
        CalculatorInterface $calculator,
        TaxRateResolverInterface $taxRateResolver,
    )
    {
        $this->calculator = $calculator;
        $this->taxRateResolver = $taxRateResolver;

    }
```

```php
    public function applyTaxes(Order $order)
    {
        $tax = 0;

        foreach ($order->getItems() as $item) {
            $taxable = $item->getProduct();
            $rate = $this->taxRateResolver->resolve($taxable);

            if (null === $rate) {
                continue; // Skip this item, there is no matching tax rate.
            }

            $tax += $this->calculator->calculate($item->getTotal(), $rate);
        }

        $order->setTaxTotal($tax); // Set the calculated taxes.
    }
}
```

Using custom tax calculators

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Every **TaxRate** model holds a *calculator* variable with the name of the tax calculation service, used to compute the tax amount. While the default calculator should fit for most common use cases, you're free to define your own implementation.

Creating the calculator

All tax calculators implement the `CalculatorInterface`. In our example we'll create a simple fee calculator. First, you need to create a new class.

```php
# src/AppBundle/Taxation/Calculator/FeeCalculator.php
<?php

declare(strict_types=1);

namespace AppBundle\Taxation\Calculator;

use Sylius\Component\Taxation\Calculator\CalculatorInterface;
use Sylius\Component\Taxation\Model\TaxRateInterface;

final class FeeCalculator implements CalculatorInterface
{
    /**
     * {@inheritdoc}
     */
    public function calculate(float $base, TaxRateInterface $rate): float
    {
        return $base * ($rate->getAmount() + 0.15 * 0.30);
```

```
    }
}
```

Now, you need to register your new service in container and tag it with `sylius.tax_calculator`.

```yaml
services:
    app.tax_calculator.fee:
        class: AppBundle\Taxation\Calculator\FeeCalculator
        tags:
            - { name: sylius.tax_calculator, calculator: fee, label: "Fee" }
```

That would be all. This new option ("Fee") will appear on the **TaxRate** creation form, in the "calculator" field.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration Reference

```yaml
sylius_taxation:
    # The driver used for persistence layer.
    driver: ~
    resources:
        tax_category:
            classes:
                model:      Sylius\Component\Taxation\Model\TaxCategory
                interface:  Sylius\Component\Taxation\Model\TaxCategoryInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\Factory
                form:       Sylius\Bundle\TaxationBundle\Form\Type\TaxCategoryType
        tax_rate:
            classes:
                model:      Sylius\Component\Taxation\Model\TaxRate
                interface:  Sylius\Component\Taxation\Model\TaxRateInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\Factory
                form:       Sylius\Bundle\TaxationBundle\Form\Type\TaxRateType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Taxation in the Sylius platform* - concept documentation

SyliusTaxonomyBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
> it. You can help us making documentation up to date via Sylius Github. Thank you!

Flexible categorization system for Symfony applications.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
> it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with [Composer](), a dependency manager for PHP. Use the following command to add the
bundle to your *composer.json* and download the package.

If you have [Composer installed globally]().

```
$ composer require sylius/taxonomy-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/taxonomy-bundle
```

Note: This version is compatible only with Symfony 2.3 or newer. Please see the CHANGELOG file in the repository,
to find version to use with older vendors.

Adding required bundles to the kernel

First, you need to enable the bundle inside the kernel. If you're not using any other Sylius bundles, you will also need
to add *SyliusResourceBundle* and its dependencies to the kernel. Don't worry, everything was automatically installed
via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Sylius\Bundle\TaxonomyBundle\SyliusTaxonomyBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
```

```
        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Container configuration

Configure doctrine extensions which are used in the taxonomy bundle:

```yaml
stof_doctrine_extensions:
    orm:
        default:
            tree: true
            sluggable: true
            sortable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Taxons

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Taxons

Retrieving taxons from database should always happen via repository, which implements `Sylius\Bundle\ResourceBundle\Model\RepositoryInterface`. If you are using Doctrine, you're already familiar with this concept, as it extends the native Doctrine `ObjectRepository` interface.

Your taxon repository is always accessible via `sylius.repository.taxon` service.

Taxon contains methods which allow you to retrieve the child taxons. Let's look at our example tree.

```
| Categories
|--  T-Shirts
|    |-- Men
|    `-- Women
|--  Stickers
|--  Mugs
`--  Books
```

To get a collection of child taxons under taxon, use the `findChildren` method.

```php
<?php

public function myAction(Request $request)
{
    // Find the parent taxon
    $taxonRepository = $this->container->get('sylius.repository.taxon');
    $taxon = $taxonRepository->findOneByName('Categories');

    $taxonRepository = $this->container->get('sylius.repository.taxon');
    $taxons = $taxonRepository->findChildren($taxon);
}
```

`$taxons` variable will now contain a list (ArrayCollection) of taxons in following order: T-Shirts, Men, Women, Stickers, Mugs, Books.

Categorization

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In this example, we will use taxonomies to categorize products and build a nice catalog.

We think that **keeping the app-specific bundle structure simple** is a good practice, so let's assume you have your `ShopBundle` registered under `Acme\ShopBundle` namespace.

```php
<?php

// src/Acme/ShopBundle/Entity/Product.php
namespace Acme\ShopBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;

class Product
{
    protected $taxons;

    public function __construct()
    {
        $this->taxons = new ArrayCollection();
    }

    public function getTaxons()
    {
        return $this->taxons;
    }

    public function setTaxons(Collection $taxons)
    {
        $this->taxons = $taxons;
    }
}
```

You also need to define the doctrine mapping with a many-to-many relation between Product and Taxons. Your product entity mapping should live inside `Resources/config/doctrine/Product.orm.xml` of your bundle.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://doctrine-project.org/schemas/orm/
↪doctrine-mapping

                                     http://doctrine-project.org/schemas/orm/
↪doctrine-mapping.xsd">

    <entity name="Acme\ShopBundle\Entity\Product" table="sylius_product">
        <id name="id" column="id" type="integer">
            <generator strategy="AUTO" />
        </id>

        <!-- Your other mappings. -->

        <many-to-many field="taxons" target-entity=
↪"Sylius\Component\Taxonomy\Model\TaxonInterface">
            <join-table name="sylius_product_taxon">
                <join-columns>
                    <join-column name="product_id" referenced-column-name="id"
↪nullable="false" />
                </join-columns>
                <inverse-join-columns>
                    <join-column name="taxon_id" referenced-column-name="id" nullable=
↪"false" />
                </inverse-join-columns>
            </join-table>
        </many-to-many>
    </entity>

</doctrine-mapping>
```

Product is just an example where we have many to many relationship with taxons, which will make it possible to categorize products and list them by taxon later.

You can classify any other model in your application the same way.

Creating your forms

To be able to apply taxonomies on your products, or whatever you are categorizing or tagging, it is handy to use *sylius_taxon_choice* form type:

```php
<?php

// src/Acme/ShopBundle/Form/ProductType.php
namespace Acme\ShopBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ProductType extends AbstractType
{
```

```php
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('taxons', 'sylius_taxon_choice');
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver
            ->setDefaults(array(
                'data_class' => 'Acme\ShopBundle\Entity\Product'
            ))
        ;
    }
}
```

This *sylius_taxon_choice* type will add a select input field for each taxonomy, with select option for each taxon.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration Reference

```yaml
sylius_taxonomy:
    # The driver used for persistence layer.
    driver: ~
    resources:
        taxon:
            classes:
                model:      Sylius\Component\Taxonomy\Model\Taxon
                interface:  Sylius\Component\Taxonomy\Model\TaxonInterface
                controller: Sylius\Bundle\ResourceBundle\Controller\ResourceController
                repository: ~
                factory:    Sylius\Component\Resource\Factory\TranslatableFactory
                form: Sylius\Bundle\TaxonomyBundle\Form\Type\TaxonType
            translation:
                classes:
                    model:      Sylius\Component\Taxonomy\Model\TaxonTranslation
                    interface: ␣
→Sylius\Component\Taxonomy\Model\TaxonTranslationInterface
                    controller:␣
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                    repository: ~
                    factory:    Sylius\Component\Resource\Factory\Factory
                    form: Sylius\Bundle\TaxonomyBundle\Form\Type\TaxonTranslationType
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Taxons in the Sylius platform* - concept documentation

SyliusThemeBundle

Flexible theming system for Symfony applications.

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/theme-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/theme-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel, usually at the end of bundle list.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
        new Symfony\Bundle\TwigBundle\TwigBundle(),

        // Other bundles...
        new Sylius\Bundle\ThemeBundle\SyliusThemeBundle(),
    );
}
```

Note: Please register the bundle after *FrameworkBundle*. This is important as we override default templating, translation and assets logic.

Configuring bundle

In order to store your themes metadata in the filesystem, add the following configuration:

```
sylius_theme:
    sources:
        filesystem: ~
```

Your first theme

This tutorial assumes that the *filesystem* source. Make sure it's enabled with the default options:

```
sylius_theme:
    sources:
        filesystem: ~
```

Themes location and definition

Private themes should be added to `app/themes` directory by default. Every theme should have a default configuration located in `composer.json` file. The only required parameter is `name`, but it is worth to define other ones (*have a look at theme configuration reference*).

```
{
  "name": "vendor/default-theme"
}
```

When adding or removing a theme, it's necessary to rebuild the container (same as adding new translation files in Symfony) by clearing the cache (`bin/console cache:clear`).

Theme structure

Themes can override and add both bundle resources and app resources. When your theme configuration is in `SampleTheme/theme.json`, app resources should be located at `SampleTheme/views` for templates, `SampleTheme/translations` for translations and `SampleTheme/public` for assets. Same comes with the bundle resources, eg. for `FOSUserBundle` the paths should be located at `SampleTheme/FOSUserBundle/views`, `SampleTheme/FOSUserBundle/translations` and `SampleTheme/FOSUserBundle/public` respectively.

```
AcmeTheme
├── AcmeBundle
│   ├── public
│   │   └── asset.jpg
│   ├── translations
│   │   └── messages.en.yml
│   └── views
│       └── template.html.twig
├── composer.json
├── translations
│   └── messages.en.yml
└── views
    └── template.html.twig
```

Enabling themes

Themes are enabled on the runtime and uses the theme context to define which one is currently used. There are two ways to enable your theme:

Custom theme context

Implement `Sylius\Bundle\ThemeBundle\Context\ThemeContextInterface`, register it as a service and replace the default theme context with the new one by changing ThemeBundle configuration:

```
sylius_theme:
    context: acme.theme_context # theme context service id
```

Request listener and settable theme context

Create an event listener and register it as listening for `kernel.request` event.

```php
use Sylius\Bundle\ThemeBundle\Context\SettableThemeContext;
use Sylius\Bundle\ThemeBundle\Repository\ThemeRepositoryInterface;
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpKernel\HttpKernelInterface;

final class ThemeRequestListener
{
    /**
     * @var ThemeRepositoryInterface
     */
    private $themeRepository;

    /**
     * @var SettableThemeContext
     */
    private $themeContext;

    /**
     * @param ThemeRepositoryInterface $themeRepository
     * @param SettableThemeContext $themeContext
     */
    public function __construct(ThemeRepositoryInterface $themeRepository,
→SettableThemeContext $themeContext)
    {
        $this->themeRepository = $themeRepository;
        $this->themeContext = $themeContext;
    }

    /**
     * @param GetResponseEvent $event
     */
    public function onKernelRequest(GetResponseEvent $event)
    {
        if (HttpKernelInterface::MASTER_REQUEST !== $event->getRequestType()) {
            // don't do anything if it's not the master request
            return;
        }
```

(continues on next page)

```
        $this->themeContext->setTheme(
            $this->themeRepository->findOneByName('sylius/cool-theme')
        );
    }
}
```

Theme assets

When creating a new theme, any templates not in your own theme are taken from the default SyliusShopBundle views - otherwise you'd need to copy all the files. But watch out! Assets like javascript resources are not loaded this way. If you install some assets you will need to link them to your theme files by using this command:

```
$ php bin/console sylius:theme:assets:install
```

Important changes

`SyliusThemeBundle` changes the way vanilla Symfony works a lot. Templates and translations will never behave the same as they were.

Templates

Changed loading order (priority descending):

- **App templates:**
    - `<Theme>/views` **(NEW!)**
    - `app/Resources/views`
- **Bundle templates:**
    - `<Theme>/<Bundle name>/views` **(NEW!)**
    - `app/Resources/<Bundle name>/views`
    - `<Bundle>/Resources/views`

Translations

Changed loading order (priority descending):

- `<Theme>/translations` **(NEW!)**
- `<Theme>/<Bundle name>/translations` **(NEW!)**
- `app/Resources/translations`
- `app/Resources/<Bundle name>/translations`
- `<Bundle>/Resources/translations`

Assets

Theme assets are installed by `sylius:theme:assets:install` command, which is supplementary for and should be used after `assets:install`.

The command run with `--symlink` or `--relative` parameters creates symlinks for every installed asset file, not for entire asset directory (eg. if `AcmeBundle/Resources/public/asset.js` exists, it creates symlink `web/bundles/acme/asset.js` leading to `AcmeBundle/Resources/public/asset.js` instead of symlink `web/bundles/acme/` leading to `AcmeBundle/Resources/public/`). When you create a new asset or delete an existing one, it is required to rerun this command to apply changes (just as the hard copy option works).

Assetic

Nothing has changed, `ThemeBundle` is not and will not be integrated with `Assetic`.

Configuration sources

To discover themes that are defined in the application, ThemeBundle uses configuration sources.

Existing configuration sources

Filesystem configuration source

Filesystem configuration source loads theme definitions from files placed under specified directories.

By default it seeks for `composer.json` files that exists under `%kernel.project_dir%/app/themes` directory, which usually is resolved to `app/themes`.

Configuration reference

```yaml
sylius_theme:
    sources:
        filesystem:
            enabled: false
            filename: composer.json
            scan_depth: null
            directories:
                - "%kernel.project_dir%/app/themes"
```

Note: Like every other source, `filesystem` is disabled if not specified otherwise. To enable it and use the default configuration, use the following configuration:

```yaml
sylius_theme:
    sources:
        filesystem: ~
```

Tip: Scanning for the configuration file inside themes directories is recursive with unlimited directory depth by default, which can result in slow performance when a lot of files are placed inside themes (e.g. a *node_modules* folder).

Define the optional *scan_depth* (integer) setting to the configuration to restrict scanning for the theme configuration file to a specific depth.

Test configuration source

Test configuration source provides an interface that can be used to add, remove and access themes in test environment. They are stored in the cache directory and if used with Behat, they are persisted across steps but not across scenarios.

Configuration reference

This source does not have any configuration options. To enable it, use the following configuration:

```
sylius_theme:
    sources:
        test: ~
```

Usage

In order to use tests, have a look at `sylius.theme.test_theme_configuration_manager` service (implementing TestThemeConfigurationManagerInterface). You can:

- add a theme: `void add(array $configuration)`
- remove a theme: `void remove(string $themeName)`
- remove all themes: `void clear()`

Creating custom configuration source

If there is no existing configuration source that fulfills your needs, you can also *create a new one*.

Creating custom configuration source

If your needs can't be fulfilled by built-in configuration sources, you can create a custom one in a few minutes!

Configuration provider

The configuration provider contains the core logic of themes configurations retrieval.

It requires only one method - `getConfigurations()` which receives no arguments and returns an array of configuration arrays.

```
use Sylius\Bundle\ThemeBundle\Configuration\ConfigurationProviderInterface;

final class CustomConfigurationProvider implements ConfigurationProviderInterface
{
    /**
     * {@inheritdoc}
     */
```

(continues on next page)

```php
    public function getConfigurations()
    {
        return [
            [
                'name' => 'theme/name',
                'path' => '/theme/path',
                'title' => 'Theme title',
            ],
        ];
    }
}
```

Configuration source factory

The configuration source factory is the glue between your brand new configuration provider and ThemeBundle.

It provides an easy way to allow customization of your configuration source and defines how the configuration provider is constructed.

```php
use Sylius\Bundle\ThemeBundle\Configuration\ConfigurationSourceFactoryInterface;
use Symfony\Component\Config\Definition\Builder\ArrayNodeDefinition;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Definition;

final class CustomConfigurationSourceFactory implements
→ConfigurationSourceFactoryInterface
{
    /**
     * {@inheritdoc}
     */
    public function buildConfiguration(ArrayNodeDefinition $node)
    {
        $node
            ->children()
                ->scalarNode('option')
        ;
    }

    /**
     * {@inheritdoc}
     */
    public function initializeSource(ContainerBuilder $container, array $config)
    {
        return new Definition(CustomConfigurationProvider::class, [
            $config['option'], // pass an argument configured by end user to
→configuration provider
        ]);
    }

    /**
     * {@inheritdoc}
     */
    public function getName()
    {
        return 'custom';
```

```
    }
}
```

Note: Try not to define any public services in the container inside `initializeSource()` - it will prevent Symfony from cleaning it up and will remain in the compiled container even if not used.

The last step is to tell ThemeBundle to use the source factory defined before. It can be done in your bundle definition:

```php
use Sylius\Bundle\ThemeBundle\DependencyInjection\SyliusThemeExtension;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\HttpKernel\Bundle\Bundle;

final class AcmeBundle extends Bundle
{
    /**
     * {@inheritdoc}
     */
    public function build(ContainerBuilder $container)
    {
        /** @var SyliusThemeExtension $themeExtension */
        $themeExtension = $container->getExtension('sylius_theme');
        $themeExtension->addConfigurationSourceFactory(new
→CustomConfigurationSourceFactory());
    }
}
```

Usage

Configuration source is set up, it will start providing themes configurations as soon as it is enabled in ThemeBundle:

```yaml
sylius_theme:
    sources:
        custom: ~
```

Theme inheritance

While you can't set two themes active at once, you can make use of multiple inheritance. Eg.:

```json
{
    "name": "vendor/child-theme",
    "extra": {
        "sylius-theme": {
            "title": "Child theme",
            "parents": ["vendor/first-parent-theme", "vendor/second-parent-theme"]
        }
    }
}
```

```json
{
    "name": "vendor/first-parent-theme",
    "extra": {
```

```json
        "sylius-theme": {
            "title": "First parent theme",
            "parents": ["vendor/grand-parent-theme"]
        }
    }
}
```

```json
{
    "name": "vendor/grand-parent-theme",
    "extra": {
        "sylius-theme": {
            "title": "Grandparent theme"
        }
    }
}
```

```json
{
    "name": "vendor/second-parent-theme",
    "extra": {
        "sylius-theme": {
            "title": "Second parent theme",
        }
    }
}
```

Configuration showed below will result in given order:

- Child theme

- First parent theme

- Grandparent theme

- Second parent theme

Grandparent theme gets overrided by first parent theme. First parent theme and second parent theme get overrided by child theme.

Theme configuration reference

```json
{
    "name": "vendor/sylius-theme",
    "title": "Great Sylius theme!",
    "description": "Optional description",
    "authors": [
        {
            "name": "Kamil Kokot",
            "email": "kamil@kokot.me",
            "homepage": "http://kamil.kokot.me",
            "role": "Developer"
        }
    ],
    "parents": [
        "vendor/common-sylius-theme",
        "another-vendor/not-so-cool-looking-sylius-theme"
```

```
    ]
}
```

> **Warning:** Theme configuration was meant to be mixed with the one from Composer. Fields `name`, `description` and `authors` are shared between these by default. To use different values for Composer & ThemeBundle, have a look below.

Composer integration

```json
{
    "name": "vendor/sylius-theme",
    "type": "sylius-theme",
    "description": "Composer package description",
    "authors": [
        {
            "name": "Kamil Kokot"
        }
    ],
    "extra": {
        "sylius-theme": {
            "description": "Theme description",
            "parents": [
                "vendor/other-sylius-theme"
            ]
        }
    }
}
```

> **Note:** By configuring Composer package along with theme we do not have to duplicate fields like `name` or `authors`, but we are free to overwrite them in any time, just like the `description` field in example above. The theme configuration is complementary to the Composer configuration and results in perfectly valid `composer.json`.

Summary

Tests

```
$ composer install
$ bin/phpspec run -f pretty
$ bin/phpunit
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

SyliusUserBundle

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

A solution for user management system inside of a Symfony application.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use the following command to add the bundle to your *composer.json* and download the package.

If you have Composer installed globally.

```
$ composer require sylius/user-bundle
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/user-bundle
```

Adding required bundles to the kernel

You need to enable the bundle inside the kernel.

If you're not using any other Sylius bundles, you will also need to add *SyliusResourceBundle* and its dependencies to kernel. Don't worry, everything was automatically installed via Composer.

```php
<?php

// app/AppKernel.php

public function registerBundles()
{
    $bundles = array(
        new FOS\RestBundle\FOSRestBundle(),
        new JMS\SerializerBundle\JMSSerializerBundle($this),
        new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
        new WhiteOctober\PagerfantaBundle\WhiteOctoberPagerfantaBundle(),
        new Bazinga\Bundle\HateoasBundle\BazingaHateoasBundle(),
        new winzou\Bundle\StateMachineBundle\winzouStateMachineBundle(),
        new Sylius\Bundle\ResourceBundle\SyliusResourceBundle(),
```

(continues on next page)

```
        new Sylius\Bundle\MailerBundle\SyliusMailerBundle(),
        new Sylius\Bundle\UserBundle\SyliusUserBundle(),

        // Other bundles...
        new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
    );
}
```

Configure Doctrine extensions

Configure doctrine extensions which are used by the bundle.

```
# app/config/config.yml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
```

Updating database schema

Run the following command.

```
$ php bin/console doctrine:schema:update --force
```

> **Warning:** This should be done only in **dev** environment! We recommend using Doctrine migrations, to safely update your schema.

Congratulations! The bundle is now installed and ready to use.

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Configuration reference

```
sylius_user:
    driver: doctrine/orm
    resources:
        admin:
            user:
                classes:
                    model: Sylius\Component\Core\Model\AdminUser
                    repository: Sylius\Bundle\UserBundle\Doctrine\ORM\UserRepository
                    form: Sylius\Bundle\CoreBundle\Form\Type\User\AdminUserType
                    interface: Sylius\Component\User\Model\UserInterface
```

```
                controller: Sylius\Bundle\UserBundle\Controller\UserController
                factory: Sylius\Component\Resource\Factory\Factory
            templates: 'SyliusUserBundle:User'
            resetting:
                token:
                    ttl: P1D
                    length: 16
                    field_name: passwordResetToken
                pin:
                    length: 4
                    field_name: passwordResetToken
            verification:
                token:
                    length: 16
                    field_name: emailVerificationToken
    shop:
        user:
            classes:
                model: Sylius\Component\Core\Model\ShopUser
                repository: Sylius\Bundle\CoreBundle\Doctrine\ORM\UserRepository
                form: Sylius\Bundle\CoreBundle\Form\Type\User\ShopUserType
                interface: Sylius\Component\User\Model\UserInterface
                controller: Sylius\Bundle\UserBundle\Controller\UserController
                factory: Sylius\Component\Resource\Factory\Factory
            templates: 'SyliusUserBundle:User'
            resetting:
                token:
                    ttl: P1D
                    length: 16
                    field_name: passwordResetToken
                pin:
                    length: 4
                    field_name: passwordResetToken
            verification:
                token:
                    length: 16
                    field_name: emailVerificationToken
    oauth:
        user:
            classes:
                model: Sylius\Component\User\Model\UserOAuth
                interface: Sylius\Component\User\Model\UserOAuthInterface
                controller:␣
→Sylius\Bundle\ResourceBundle\Controller\ResourceController
                factory: Sylius\Component\Resource\Factory\Factory
                form: Sylius\Bundle\UserBundle\Form\Type\UserType
            templates: 'SyliusUserBundle:User'
            resetting:
                token:
                    ttl: P1D
                    length: 16
                    field_name: passwordResetToken
                pin:
                    length: 4
                    field_name: passwordResetToken
            verification:
                token:
```

```
                     length: 16
                     field_name: emailVerificationToken
```

Bug tracking

This bundle uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Users & Customers in the Sylius platform* - concept documentation
- *SyliusAddressingBundle*
- *SyliusAttributeBundle*
- *SyliusCustomerBundle*
- *SyliusFixturesBundle*
- *SyliusGridBundle*
- *SyliusInventoryBundle*
- *SyliusMailerBundle*
- *SyliusOrderBundle*
- *SyliusProductBundle*
- *SyliusPromotionBundle*
- *SyliusResourceBundle*
- *SyliusShippingBundle*
- *SyliusTaxationBundle*
- *SyliusTaxonomyBundle*
- *SyliusThemeBundle*
- *SyliusUserBundle*

9.1.2 Sylius Components Documentation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We provide a set of well-tested and decoupled PHP libraries.

The components are the foundation of the Sylius platform, but they can also be used standalone with any PHP application even if you use a framework different than Symfony.

These packages solve common E-Commerce and web application problems. Have a look around this documentation to see if you will find them useful!

We recommend checking out *Components General Guide*, which will get you started in minutes.

Components General Guide

All Sylius components have very similar structure and this guide will introduce you these conventions.

Through this documentation, you will learn how to install and use them in any PHP application.

How to Install and Use the Sylius Components

If you're starting a new project (or already have a project) that will use one or more components, the easiest way to integrate everything is with Composer. Composer is smart enough to download the component(s) that you need and take care of autoloading so that you can begin using the libraries immediately.

This article will take you through using *Taxation*, though this applies to using any component.

Using the Taxation Component

1. If you're creating a new project, create a new empty directory for it.

```
$ mkdir project/
$ cd project/
```

2. Open a terminal and use Composer to grab the library.

Tip: if you don't have it already present on your system. Depending on how you install, you may end up with a `composer.phar` file in your directory. In that case, no worries! Just run `php composer.phar require sylius/taxation`.

```
$ composer require sylius/taxation
```

The name `sylius/taxation` is written at the top of the documentation for whatever component you want.

3. Write your code!

Once Composer has downloaded the component(s), all you need to do is include the `vendor/autoload.php` file that was generated by Composer. This file takes care of autoloading all of the libraries so that you can use them immediately. Open your favorite code editor and start coding:

```php
<?php

// Sample script.php file.

require_once __DIR__.'/vendor/autoload.php';

use Sylius\Component\Taxation\Calculator\DefaultCalculator;
use Sylius\Component\Taxation\Model\TaxRate;

$calculator = new DefaultCalculator();

$taxRate = new TaxRate();
$taxRate->setAmount(0.23);

$taxAmount = $calculator->calculate(100, $taxRate);

echo $taxAmount; // Outputs "23".
```

You can open the "script.php" file in browser or run it via console:

```
$ php script.php
```

Using all of the Components

If you want to use all of the Sylius Components, then instead of adding them one by one, you can include the `sylius/sylius` package:

```
$ composer require sylius/sylius
```

Now what?

Check out *What is a Resource?*, which will give you basic understanding about how all Sylius components look and work like.

Enjoy!

What is a Resource?

We refer to data models as "Resources". In the simplest words, some examples that you will find in Sylius:

- Product
- TaxRate
- Order
- OrderItem
- ShippingMethod
- PaymentMethod

As you can already guess, there are many more Resources in Sylius. It is a really simple but powerful concept that allows us to create a nice abstraction to handle all the complex logic of E-Commerce. When using Components, you will have access to the resources provided by them out-of-the-box.

What is more, you will be able to create your own, custom Resources and benefit from all features provided by Sylius.

Now what?

Learn how we handle *Creating Resources* via Factory pattern.

Creating Resources

Every resource provided by a Sylius component should be created via a factory.

Some resources use the default resource class while some use custom implementations to provide extra functionality.

Using Factory To Create New Resource

To create new resources you should use the default factory implementation.

```php
<?php

use Sylius\Component\Product\Model\Product;
use Sylius\Component\Resource\Factory\Factory;

$factory = new Factory(Product::class);

$product = $factory->createNew();
```

That's it! The `$product` variable will hold a clean instance of the Product model.

Why Even Bother?

"Hey! This is same as `$product = new Product();`!"

Yes, and no. Every Factory implements FactoryInterface and this allows you to abstract the way that resources are created. It also makes testing much simpler because you can mock the Factory and use it as a test double in your service.

What is more, thanks to usage of Factory pattern, Sylius is able to easily swap the default Product (or any other resource) model with your custom implementation, without changing code.

Note: For more detailed information go to Sylius API Factory.

> **Caution:** In a concrete Component's documentation we will use `new` keyword to create resources - just to keep things simpler to read.

Addressing

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sylius Addressing component for PHP E-Commerce applications which provides you with basic Address, Country, Province and Zone models.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/addressing` on Packagist);

- Use the official Git repository (https://github.com/Sylius/Addressing).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ZoneMatcher

Zones are not very useful by themselves, but they can take part in e.g. a complex taxation/shipping system. This service is capable of getting a *Zone* specific for given *Address*.

It uses a collaborator implementing Doctrine's ObjectRepository interface to obtain all available zones, compare them with given *Address* and return best fitted *Zone*.

First lets make some preparations.

```php
<?php

use Sylius\Component\Addressing\Model\Address;
use Sylius\Component\Addressing\Model\Zone;
use Sylius\Component\Addressing\Model\ZoneInterface;
use Sylius\Component\Addressing\Model\ZoneMember;
use Sylius\Component\Resource\Repository\InMemoryRepository;

$zoneRepository = new InMemoryRepository(ZoneInterface::class);
$zone = new Zone();
$zoneMember = new ZoneMember();

$address = new Address();
$address->setCountry('US');

$zoneMember->setCode('US');
$zoneMember->setBelongsTo($zone);

$zone->addMember($zoneMember);

$zoneRepository->add($zone);
```

Now that we have all the needed parts lets match something.

```php
<?php

use Sylius\Component\Addressing\Matcher\ZoneMatcher;

$zoneMatcher = new ZoneMatcher($zoneRepository);

$zoneMatcher->match($address); // returns the best matching zone
                               // for the address given, in this case $zone
```

ZoneMatcher can also return all zones containing given *Address*.

```
<?php

$zoneMatcher->matchAll($address); // returns all zones containing given $address
```

To be more specific you can provide a `scope` which will narrow the search only to zones with same corresponding property.

```
<?php

$zone->setScope('earth');

$zoneMatcher->match($address, 'earth'); // returns $zone
$zoneMatcher->matchAll($address, 'mars'); // returns null as there is no
                                          // zone with 'mars' scope
```

Note: This service implements the *ZoneMatcherInterface*.

Caution: Throws \InvalidArgumentException.

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Address

The customer's address is represented by an **Address** model. It should contain all data concerning customer's address and as default has the following properties:

Property	Description
id	Unique id of the address
firstName	Customer's first name
lastName	Customer's last name
phoneNumber	Customer's phone number
company	Company name
country	Country's ISO code
province	Province's code
street	Address' street
city	Address' city
postcode	Address' postcode
createdAt	Date when address was created
updatedAt	Date of last address' update

Note: This model implements the *AddressInterface*.

For more detailed information go to Sylius API Address.

Country

The geographical area of a country is represented by a **Country** model. It should contain all data concerning a country and as default has the following properties:

Property	Description
id	Unique id of the country
code	Country's ISO code
provinces	Collection of **Province** objects
enabled	Indicates whether country is enabled

Note: This model implements the *CountryInterface* and *CodeAwareInterface*.

For more detailed information go to Sylius API Country.

Province

Smaller area inside a country is represented by a **Province** model. You can use it to manage provinces or states and assign it to an address as well. It should contain all data concerning a province and as default has the following properties:

Property	Description
id	Unique id of the province
code	Unique code of the province
name	Province's name
country	The **Country** this province is assigned to

Note: This model implements the *ProvinceInterface* and *CodeAwareInterface*.

For more detailed information go to Sylius API Province.

Zone

The geographical area is represented by a **Zone** model. It should contain all data concerning a zone and as default has the following properties:

Property	Description
id	Unique id of the zone
code	Unique code of the zone
name	Zone's name
type	Zone's type
scope	Zone's scope
members	All of the **ZoneMember** objects assigned to this zone

Note: This model implements the *ZoneInterface* and *CodeAwareInterface*.

For more detailed information go to Sylius API Zone.

ZoneMember

In order to add a specific location to a **Zone**, an instance of **ZoneMember** must be created with that location's code. On default this model has the following properties:

Property	Description
id	Unique id of the zone member
code	Unique code of affiliated member i.e. country's code
belongsTo	The **Zone** this member is assigned to

Note: This model implements *ZoneMemberInterface* and *CodeAwareInterface*.

For more detailed information go to Sylius API ZoneMember.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

AddressInterface

This interface should be implemented by models representing the customer's address.

Note: This interface extends *TimestampableInterface*.

For more detailed information go to Sylius API AddressInterface.

CountryInterface

This interfaces should be implemented by models representing a country.

Note: This interface extends *ToggleableInterface*.

For more detailed information go to Sylius API CountryInterface.

ProvinceInterface

This interface should be implemented by models representing a part of a country.

Note: For more detailed information go to Sylius API ProvinceInterface.

ZoneInterface

This interface should be implemented by models representing a single zone.

It also holds all the *Zone Types*.

Note: For more detailed information go to Sylius API ZoneInterface.

ZoneMemberInterface

This interface should be implemented by models that represent an area a specific zone contains, e.g. all countries in the European Union.

Note: For more detailed information go to Sylius API ZoneMemberInterface.

Service Interfaces

RestrictedZoneCheckerInterface

A service implementing this interface should be able to check if given *Address* is in a restricted zone.

Note: For more detailed information go to Sylius API RestrictedZoneCheckerInterface.

ZoneMatcherInterface

This interface should be implemented by a service responsible of finding the best matching zone, and all zones containing the provided *Address*.

Note: For more detailed information go to Sylius API ZoneMatcherInterface.

Zone Types

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

There are three zone types available by default:

Related constant	Type
TYPE_COUNTRY	country
TYPE_PROVINCE	province
TYPE_ZONE	zone

Note: All of the above types are constant fields in the *ZoneInterface*.

Learn more

- *Addresses in the Sylius platform* - concept documentation

Attribute

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Handling of dynamic attributes on PHP models is a common task for most of modern business applications. Sylius component makes it easier to handle different types of attributes and attach them to any object by implementing a simple interface.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/attribute` on Packagist).
- Use the official Git repository (https://github.com/Sylius/Attribute).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Creating an attributable class

In the following example you will see a minimalistic implementation of the *AttributeSubjectInterface*.

```php
<?php

namespace App\Model;

use Sylius\Component\Attribute\Model\AttributeSubjectInterface;
use Sylius\Component\Attribute\Model\AttributeValueInterface;
use Doctrine\Common\Collections\Collection;

class Shirt implements AttributeSubjectInterface
{
    /**
     * @var AttributeValueInterface[]
     */
    private $attributes;

    /**
     * {@inheritdoc}
     */
    public function getAttributes()
    {
        return $this->attributes;
    }

    /**
     * {@inheritdoc}
     */
    public function setAttributes(Collection $attributes)
    {
        foreach ($attributes as $attribute) {
            $this->addAttribute($attribute);
        }
    }

    /**
     * {@inheritdoc}
     */
    public function addAttribute(AttributeValueInterface $attribute)
    {
        if (!$this->hasAttribute($attribute)) {
            $attribute->setSubject($this);
            $this->attributes[] = $attribute;
        }
    }

    /**
     * {@inheritdoc}
     */
    public function removeAttribute(AttributeValueInterface $attribute)
    {
        if ($this->hasAttribute($attribute)){
            $attribute->setSubject(null);
            $key = array_search($attribute, $this->attributes);
            unset($this->attributes[$key]);
        }
    }

    /**
```

```php
     * {@inheritdoc}
     */
    public function hasAttribute(AttributeValueInterface $attribute)
    {
        return in_array($attribute, $this->attributes);
    }

    /**
     * {@inheritdoc}
     */
    public function hasAttributeByName($attributeName)
    {
        foreach ($this->attributes as $attribute) {
            if ($attribute->getName() === $attributeName) {
                return true;
            }
        }

        return false;
    }

    /**
     * {@inheritdoc}
     */
    public function getAttributeByName($attributeName)
    {
        foreach ($this->attributes as $attribute) {
            if ($attribute->getName() === $attributeName) {
                return $attribute;
            }
        }

        return null;
    }

    /**
     * {@inheritdoc}
     */
    public function hasAttributeByCodeAndLocale($attributeCode, $localeCode = null)
    {

    }

    /**
     * {@inheritdoc}
     */
    public function getAttributeByCodeAndLocale($attributeCode, $localeCode = null)
    {

    }
}
```

Note: An implementation similar to the one above has been done in the *Product* model.

Adding attributes to an object

Once we have our class we can characterize it with attributes.

```php
<?php

use App\Model\Shirt;
use Sylius\Component\Attribute\Model\Attribute;
use Sylius\Component\Attribute\Model\AttributeValue;
use Sylius\Component\Attribute\AttributeType\TextAttributeType;
use Sylius\Component\Attribute\Model\AttributeValueInterface;

$attribute = new Attribute();
$attribute->setName('Size');
$attribute->setType(TextAttributeType::TYPE);
$attribute->setStorageType(AttributeValueInterface::STORAGE_TEXT);

$smallSize = new AttributeValue();
$mediumSize = new AttributeValue();

$smallSize->setAttribute($attribute);
$mediumSize->setAttribute($attribute);

$smallSize->setValue('S');
$mediumSize->setValue('M');

$shirt = new Shirt();

$shirt->addAttribute($smallSize);
$shirt->addAttribute($mediumSize);
```

Or you can just add all attributes needed using a class implementing Doctrine's Collection interface, e.g. the Array-Collection class.

> **Warning:** Beware! It's really important to set proper attribute storage type, which should reflect value type that is set in *AttributeValue*.

```php
<?php

use Doctrine\Common\Collections\ArrayCollection;

$attributes = new ArrayCollection();

$attributes->add($smallSize);
$attributes->add($mediumSize);

$shirt->setAttributes($attributes);
```

> **Note:** Notice that you don't actually add an *Attribute* to the subject, instead you need to add every *AttributeValue* assigned to the attribute.

Accessing attributes

```php
<?php

$shirt->getAttributes(); // returns an array containing all set attributes

$shirt->hasAttribute($smallSize); // returns true
$shirt->hasAttribute($hugeSize); // returns false
```

Accessing attributes by name

```php
<?php

$shirt->hasAttributeByName('Size'); // returns true

$shirt->getAttributeByName('Size'); // returns $smallSize
```

Removing an attribute

```php
<?php

$shirt->hasAttribute($smallSize); // returns true

$shirt->removeAttribute($smallSize);

$shirt->hasAttribute($smallSize); // now returns false
```

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Attribute

Every attribute is represented by the **Attribute** model which by default has the following properties:

Property	Description
id	Unique id of the attribute
type	Attribute's type ('text' by default)
name	Attribute's name
configuration	Attribute's configuration
validation	Attribute's validation configuration
values	Collection of attribute values
storageType	Defines how attribute value should be stored in database
createdAt	Date when attribute was created
updatedAt	Date of last attribute update

Note: This model uses the *Using TranslatableTrait* and implements the *AttributeInterface*.

For more detailed information go to Sylius API Attribute.

> **Attention:** Attribute's type is an alias of AttributeType service.

AttributeValue

This model binds the subject and the attribute, it is used to store the value of the attribute for the subject. It has the following properties:

Property	Description
id	Unique id of the attribute value
subject	Reference to attribute's subject
attribute	Reference to an attribute
value	Attribute's value (not mapped)
text	Value of attribute stored as text
boolean	Value of attribute stored as boolean
integer	Value of attribute stored as integer
float	Value of attribute stored as float
datetime	Value of attribute stored as datetime
date	Value of attribute stored as date

> **Attention:** `Value` property is used only as proxy, that stores data in proper field. It's crucial to set attribute value in field, that is mapped as attribute's storage type.

Note: This model implements the *AttributeValueInterface*.

For more detailed information go to Sylius API AttributeValue.

AttributeTranslation

The attribute's name for different locales is represented by the **AttributeTranslation** model which has the following properties:

Property	Description
id	Unique id of the attribute translation
name	Attribute's name for given locale

Note: This model extends the *Implementing AbstractTranslation* class and implements the *AttributeTranslationInterface*.

For more detailed information go to Sylius API AttributeTranslation.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

AttributeInterface

This interface should be implemented by models used for describing a product's attribute.

Note: This interface extends the *TimestampableInterface* and the *AttributeTranslationInterface*.

For more detailed information go to Sylius API AttributeInterface.

AttributeValueInterface

This interface should be implemented by models used for binding an *Attribute* with a model implementing the *AttributeSubjectInterface* e.g. the *Product*.

Note: For more detailed information go to Sylius API AttributeValueInterface.

AttributeTranslationInterface

This interface should be implemented by models maintaining a single translation of an *Attribute* for specified locale.

Note: For more detailed information go to Sylius API AttributeTranslationInterface.

AttributeSubjectInterface

This interface should be implemented by models you want to characterize with various *AttributeValue* objects.

It will ask you to implement the management of *AttributeValue* models.

Note: For more detailed information go to Sylius API AttributeSubjectInterface.

AttributeTypeInterface

This interface should be implemented by models used for describing a product's attribute type.

Learn more

- *Attributes in the Sylius platform* - concept documentation

Channel

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sale channels management implementation in PHP.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (sylius/channel on Packagist);
- Use the official Git repository (https://github.com/Sylius/Channel).

Then, require the vendor/autoload.php file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ChannelContext

The **ChannelContext** allows you to manage the currently used sale channel.

```php
<?php

use Sylius\Component\Channel\Context\ChannelContext;
use Sylius\Component\Channel\Model\Channel;

$channel = new Channel();
$channelContext = new ChannelContext();

$channelContext->setChannel($channel);

$channelContext->getChannel(); // will return the $channel object
```

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Channel

Sale channel is represented by a **Channel** model. It should have everything concerning channel's data and as default has the following properties:

Property	Description
id	Unique id of the channel
code	Channel's code
name	Channel's name
description	Channel's description
url	Channel's URL
color	Channel's color
enabled	Indicates whether channel is available
createdAt	Date of creation
updatedAt	Date of update

> **Note:** This model implements *ChannelInterface*.
>
> For more detailed information go to Sylius API Channel.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

ChannelInterface

This interface should be implemented by every custom sale channel model.

> **Note:** This interface extends *TimestampableInterface* and *CodeAwareInterface*.
>
> For more detailed information go to Sylius API ChannelInterface.

ChannelAwareInterface

This interface should be implemented by models associated with a specific sale channel.

Note: For more detailed information go to Sylius API ChannelAwareInterface.

ChannelsAwareInterface

This interface should be implemented by models associated with multiple channels.

Note: For more detailed information go to Sylius API ChannelsAwareInterface.

Service Interfaces

Learn more

- *Channels in the Sylius platform* - concept documentation

Currency

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Managing different currencies, exchange rates and converting cash amounts for PHP applications.

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/currency` on Packagist);

- Use the official Git repository (https://github.com/Sylius/Currency).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Getting a Currency name

```php
<?php

use Sylius\Component\Currency\Model\Currency;

$currency = new Currency();
$currency->setCode('USD');

$currency->getName(); // Returns 'US Dollar'.
```

The `getName` method uses Symfony's Intl class to convert currency's code into a human friendly form.

Note: The output of `getName` may vary as the name is generated accordingly to the set locale.

CurrencyConverter

The **CurrencyConverter** allows you to convert a value accordingly to the exchange rate of specified currency.

This behaviour is used just for displaying the *approximate* value in another currency than the base currency of the channel.

Note: This service implements the *CurrencyConverterInterface*.

For more detailed information go to Sylius API CurrencyConverter.

Caution: Throws *UnavailableCurrencyException*.

CurrencyProvider

The **CurrencyProvider** allows you to get all available currencies.

```php
<?php

use Sylius\Component\Currency\Provider\CurrencyProvider;
use Sylius\Component\Resource\Repository\InMemoryRepository;

$currencyRepository = new InMemoryRepository();
$currencyProvider = new CurrencyProvider($currencyRepository);

$currencyProvider->getAvailableCurrencies(); // Returns an array of Currency objects.
```

The `getAvailableCurrencies` method retrieves all currencies which `enabled` property is set to true and have been inserted in the given repository.

Note: This service implements the *CurrencyProviderInterface*.

For more detailed information go to Sylius API CurrencyProvider.

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Currency

Every currency is represented by a **Currency** model which by default has the following properties:

Method	Description
id	Unique id of the currency
code	Currency's code
createdAt	Date of creation
updatedAt	Date of last update

> **Note:** This model implements *CurrencyInterface*.
>
> For more detailed information go to Sylius API Currency.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

CurrencyInterface

This interface provides you with basic management of a currency's code, name, exchange rate and whether the currency should be enabled or not.

> **Note:** This interface extends *CodeAwareInterface* and *TimestampableInterface*.
>
> For more detailed information go to Sylius API CurrencyInterface.

Service Interfaces

CurrenciesAwareInterface

Any container used to store, and manage currencies should implement this interface.

> **Note:** For more detailed information go to Sylius API CurrenciesAwareInterface.

CurrencyContextInterface

This interface should be implemented by a service used for managing the currency name. It also contains the default storage key:

Related constant	Storage key
STORAGE_KEY	_sylius_currency

Note: For more detailed information go to Sylius API CurrencyContextInterface.

CurrencyConverterInterface

This interface should be implemented by any service used to convert the amount of money from one currency to another, according to their exchange rates.

Note: For more detailed information go to Sylius API CurrencyConverterInterface.

CurrencyProviderInterface

This interface allows you to implement one fast service which gets all available currencies from any container you would like.

Note: For more detailed information go to Sylius API CurrencyProviderInterface.

UnavailableCurrencyException

This exception is thrown when you try converting to a currency which is not present in the provided repository.

Note: This exception extends the \InvalidArgumentException.

Learn more

- *Currencies in the Sylius platform* - concept documentation

Grid

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sylius component used for describing data grids. Decoupled from Symfony and useful for any kind of system, which needs to provide user with grid functionality.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

We assume you're familiar with Composer, a dependency manager for PHP. Use following command to add the component to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/grid
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/grid
```

Summary

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Bug tracking

This component uses GitHub issues. If you have found bug, please create an issue.

Inventory

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Inventory management for PHP applications.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/inventory` on Packagist);

> • Use the official Git repository (https://github.com/Sylius/Inventory).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Stockable Object

The first thing you should do it is implementing stockable object. Example implementation:

```php
<?php

class Product implements StockableInterface
{
    /**
     * Get stock keeping unit.
     *
     * @return mixed
     */
    public function getSku()
    {
        // TODO: Implement getSku() method.
    }

    /**
     * Get inventory displayed name.
     *
     * @return string
     */
    public function getInventoryName()
    {
        // TODO: Implement getInventoryName() method.
    }

    /**
     * Simply checks if there any stock available.
     *
     * @return Boolean
     */
    public function isInStock()
    {
        // TODO: Implement isInStock() method.
    }

    /**
     * Get stock on hold.
     *
     * @return integer
     */
    public function getOnHold()
```

(continues on next page)

```
    {
        // TODO: Implement getOnHold() method.
    }

    /**
     * Set stock on hold.
     *
     * @param integer
     */
    public function setOnHold($onHold)
    {
        // TODO: Implement setOnHold() method.
    }

    /**
     * Get stock on hand.
     *
     * @return integer
     */
    public function getOnHand()
    {
        // TODO: Implement getOnHand() method.
    }

    /**
     * Set stock on hand.
     *
     * @param integer $onHand
     */
    public function setOnHand($onHand)
    {
        // TODO: Implement setOnHand() method.
    }
}
```

InventoryOperator

The **InventoryOperator** provides basic operations on your inventory.

```php
<?php

use Sylius\Component\Inventory\Operator\InventoryOperator;
use Sylius\Component\Inventory\Checker\AvailabilityChecker;
use Sylius\Component\Resource\Repository\InMemoryRepository;

$inMemoryRepository = new InMemoryRepository(); // Repository model.
$product = new Product(); // Stockable model.
$eventDispatcher; // It gives a possibility to hook before or after each operation.
// If you are not familiar with events, check the symfony Event Dispatcher.

$availabilityChecker = new AvailabilityChecker(false);
$inventoryOperator = new InventoryOperator($availabilityChecker, $eventDispatcher);

$product->getOnHand(); // Output will be 0.
$inventoryOperator->increase($product, 5);
```

```php
$product->getOnHand(); // Output will be 5.

$product->getOnHold(); // Output will be 0.
$inventoryOperator->hold($product, 4);
$product->getOnHold(); // Output will be 4.

$inventoryOperator->release($product, 3);
$product->getOnHold(); // Output will be 1.
```

Decrease

```php
<?php

use Sylius\Component\Inventory\Operator\InventoryOperator;
use Sylius\Component\Inventory\Checker\AvailabilityChecker;
use Doctrine\Common\Collections\ArrayCollection;
use Sylius\Component\Inventory\Model\InventoryUnit;
use Sylius\Component\Inventory\Model\InventoryUnitInterface;

$inventoryUnitRepository; // Repository model.
$product = new Product(); // Stockable model.
$eventDispatcher; // It gives possibility to hook before or after each operation.
// If you are not familiar with events. Check symfony event dispatcher.

$availabilityChecker = new AvailabilityChecker(false);
$inventoryOperator = new InventoryOperator($availabilityChecker, $eventDispatcher);
$inventoryUnit1 = new InventoryUnit();
$inventoryUnit2 = new InventoryUnit();
$inventoryUnits = new ArrayCollection();
$product->getOnHand(); // Output will be 5.

$inventoryUnit1->setStockable($product);
$inventoryUnit1->setInventoryState(InventoryUnitInterface::STATE_SOLD);

$inventoryUnit2->setStockable($product);

$inventoryUnits->add($inventoryUnit1);
$inventoryUnits->add($inventoryUnit2);

count($inventoryUnits); // Output will be 2.
$inventoryOperator->decrease($inventoryUnits);
$product->getOnHand(); // Output will be 4.
```

Caution: All methods in **InventoryOperator** throw InvalidArgumentException or InsufficientStockException if an error occurs.

Note: For more detailed information go to Sylius API InventoryOperator.

Hint: To understand how events work check Symfony EventDispatcher.

NoopInventoryOperator

In some cases, you may want to have unlimited inventory, this operator will allow you to do that.

Hint: This operator is based on the null object pattern. For more detailed information go to Null Object pattern.

Note: For more detailed information go to Sylius API NoopInventoryOperator.

AvailabilityChecker

The **AvailabilityChecker** checks availability of a given stockable object. To characterize an object which is an **AvailabilityChecker**, it needs to implement the *AvailabilityCheckerInterface*. Second parameter of the ->isStockSufficient() method gives a possibility to check for a given quantity of a stockable.

```php
<?php

use Sylius\Component\Inventory\Checker\AvailabilityChecker;

$product = new Product(); // Stockable model.
$product->getOnHand(); // Output will be 5
$product->getOnHold(); // Output will be 4

$availabilityChecker = new AvailabilityChecker(false);
$availabilityChecker->isStockAvailable($product); // Output will be true.
$availabilityChecker->isStockSufficient($product, 5); // Output will be false.
```

InventoryUnitFactory

The **InventoryUnitFactory** creates a collection of new inventory units.

```php
<?php

use Sylius\Component\Inventory\Factory\InventoryUnitFactory;
use Sylius\Component\Inventory\Model\InventoryUnitInterface;

$inventoryUnitRepository; // Repository model.
$product = new Product(); // Stockable model.

$inventoryUnitFactory = new InventoryUnitFactory($inventoryUnitRepository);

$inventoryUnits = $inventoryUnitFactory->create($product, 10,
→InventoryUnitInterface::STATE_RETURNED);
// Output will be collection of inventory units.

$inventoryUnits[0]->getStockable(); // Output will be your's stockable model.
$inventoryUnits[0]->getInventoryState(); // Output will be 'returned'.
count($inventoryUnits); // Output will be 10.
```

Note: For more detailed information go to Sylius API InventoryUnitFactory.

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

InventoryUnit

InventoryUnit object represents an inventory unit. InventoryUnits have the following properties:

Property	Description
id	Unique id of the inventory unit
stockable	Reference to any stockable unit. (Implements *StockableInterface*)
inventoryState	State of the inventory unit (e.g. "checkout", "sold")
createdAt	Date when inventory unit was created
updatedAt	Date of last change

> **Note:** This model implements the *InventoryUnitInterface* For more detailed information go to Sylius API InventoryUnit.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

InventoryUnitInterface

This interface should be implemented by model representing a single InventoryUnit.

> **Hint:** It also contains the default *State Machine*.

> **Note:** This interface extends *TimestampableInterface*.
>
> For more detailed information go to Sylius API InventoryUnitInterface.

StockableInterface

This interface provides basic operations for any model that can be stored.

Note: For more detailed information go to Sylius API StockableInterface.

Service Interfaces

AvailabilityCheckerInterface

This interface provides methods for checking availability of stockable objects.

Note: For more detailed information go to Sylius API AvailabilityCheckerInterface.

InventoryUnitFactoryInterface

This interface is implemented by services responsible for creating collection of new inventory units.

Note: For more detailed information go to Sylius API InventoryUnitFactoryInterface.

State Machine

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Inventory Unit States

Sylius itself uses a complex state machine system to manage all states of the business domain. This component has some sensible default states defined in the **InventoryUnitInterface**.

All new **InventoryUnit** instances have the state `checkout` by default, which means they are in the cart and wait for verification.

The following states are defined:

Related constant	State	Description
STATE_CHECKOUT	checkout	Item is in the cart
STATE_ONHOLD	onhold	Item is hold (e.g. waiting for the payment)
STATE_SOLD	sold	Item has been sold and is no longer in the warehouse
STATE_RETURNED	returned	Item has been sold, but returned and is in stock

---

Tip: Please keep in mind that these states are just default, you can define and use your own. If you use this component with *SyliusInventoryBundle* and Symfony, you will have full state machine configuration at your disposal.

Inventory Unit Transitions

There are the following order's transitions by default:

Related constant	Transition
SYLIUS_HOLD	hold
SYLIUS_SELL	sell
SYLIUS_RELEASE	release
SYLIUS_RETURN	return

There is also the default graph name included:

Related constant	Name
GRAPH	sylius_inventory_unit

Note: All of above transitions and the graph are constant fields in the **InventoryUnitTransitions** class.

Learn more

- *Inventory in the Sylius platform* - concept documentation

Locale

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Managing different locales for PHP apps.

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/locale` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Locale).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

LocaleContext

In the Locale component there are three LocaleContexts defined: `* CompositeLocaleContext * ImmutableLocaleContext * ProviderBasedLocaleContext`

CompositeLocaleContext

It is a composite of different contexts available in your application, which are prioritized while being injected here (the one with highest priority is used). It has the `getLocaleCode()` method available, that helps you to get the currently used locale.

LocaleProvider

The **LocaleProvider** allows you to get all available locales.

```php
<?php

use Sylius\Component\Locale\Provider\LocaleProvider;

$locales = new InMemoryRepository();

$localeProvider = new LocaleProvider($locales);

$localeProvider->getAvailableLocalesCodes() //Output will be a collection of
→available locales
$localeProvider->isLocaleAvailable('en') //It will check if that locale is enabled
```

> **Note:** For more detailed information go to Sylius API LocaleProvider.

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Locale

Locale represents one locale available in the application. It uses Symfony Intl component to return locale name. Locale has the following properties:

Property	Description
id	Unique id of the locale
code	Locale's code
createdAt	Date when locale was created
updatedAt	Date of last change

Hint: This model has one const `STORAGE_KEY` it is key used to store the locale in storage.

Note: This model implements the *LocaleInterface* For more detailed information go to Sylius API Locale.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

LocaleInterface

This interface should be implemented by models representing a single **Locale**.

Note: This interface extends *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API LocaleInterface.

LocalesAwareInterface

This interface provides basic operations for locale management. If you want to have locales in your model just implement this interface.

Note: For more detailed information go to Sylius API LocalesAwareInterface.

Service Interfaces

LocaleContextInterface

This interface is implemented by the service responsible for managing the current locale.

Note: For more detailed information go to Sylius API LocaleContextInterface.

LocaleProviderInterface

This interface is implemented by the service responsible for providing you with a list of available locales.

Note: For more detailed information go to Sylius API LocaleProviderInterface.

Learn more

- *Locales in the Sylius platform* - concept documentation

Mailer

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sylius Mailer component abstracts the process of sending e-mails. It also provides interface to configure various parameters for unique e-mails.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/mailer` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Mailer).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Sender

SenderAdapter

This is an abstraction layer that allows you to create your own logic of sending an email.

```php
<?php

use Sylius\Component\Mailer\Sender\Adapter\AbstractAdapter as BaseSenderAdapter;
use Sylius\Component\Mailer\Model\EmailInterface;
use Sylius\Component\Mailer\Model\Email;

class SenderAdapter extends BaseSenderAdapter
{
    /**
     * Send an e-mail.
     *
     * @param array  $recipients
     * @param string $senderAddress
     * @param string $senderName
     * @param RenderedEmail $renderedEmail
     * @param EmailInterface $email
     */
    public function send(array $recipients, $senderAddress, $senderName,
→RenderedEmail $renderedEmail, EmailInterface $email, array $data)
    {
        // TODO: Implement send() method.
    }
}

$email = new Email();

$email->setCode('christmas_party_invitation');
$email->setContent('Hi, we would like to invite you to christmas party');
$email->setSubject('Christmas party');
$email->setSenderAddress('mike.ehrmantraut@gmail.com');
$email->setSenderName('Mike Ehrmantraut');

$senderAdapter = new SenderAdapter();
$rendererAdapter = new RendererAdapter();

$renderedEmail = $rendererAdapter->render($email, $data);

$senderAdapter->send(array('john.doe@gmail.com'), $email->getSenderAddress(), $email->
→getSenderName(), $renderedEmail, $email, array())
```

Sender

This service collects those two adapters **SenderAdapter**, **RendererAdapter** and deals with process of sending an email.

```php
<?php

use Sylius\Component\Mailer\Provider\DefaultSettingsProvider;
use Sylius\Component\Mailer\Provider\EmailProvider;
use Sylius\Component\Mailer\Sender\Sender;

$sender = new Sender($rendererAdapter, $senderAdapter, $emailProvider,
→$defaultSettingsProvider);

$sender->send('christmas_party_invitation', array('mike.ehrmantraut@gmail.com'));
```

Renderer

RendererAdapter

This is an abstraction layer that allows you to create your own logic of rendering an email object.

```php
<?php

use Sylius\Component\Mailer\Renderer\Adapter\AbstractAdapter as BaseRendererAdapter;
use Sylius\Component\Mailer\Model\EmailInterface;
use Sylius\Component\Mailer\Model\Email;

class RendererAdapter extends BaseRendererAdapter
{
    /**
     * Render an e-mail.
     *
     * @param EmailInterface $email
     * @param array $data
     *
     * @return RenderedEmail
     */
    public function render(EmailInterface $email, array $data = array())
    {
        // TODO: Implement render() method.

        return new RenderedEmail($subject, $body);
    }
}

$email = new Email();

$email->setCode('christmas_party_invitation');
$email->setContent('Hi, we would like to invite you to christmas party');
$email->setSubject('Christmas party');
$email->setSenderAddress('mike.ehrmantraut@gmail.com');
$email->setSenderName('Mike Ehrmantraut');

$rendererAdapter = new RendererAdapter();
$renderedEmail = $rendererAdapter->render($email, $data); // It will render an email
→object based on your implementation.

$renderedEmail->getBody(); // Output will be Hi, we would .....
$renderedEmail->getSubject(); // Output will be Christmas party.
```

Hint: Renderer should return RenderedEmail

DefaultSettingsProvider

The **DefaultSettingsProvider** is service which provides you with default sender address and sender name.

```php
<?php
```

```php
use Sylius\Component\Mailer\Provider\DefaultSettingsProvider;

$defaultSettingsProvider = new DefaultSettingsProvider('Mike Ehrmantraut', 'mike.
↪ehrmantraut@gmail.com');

$defaultSettingsProvider->getSenderAddress(); // mike.ehrmantraut@gmail.com
$defaultSettingsProvider->getSenderName(); // Output will be Mike Ehrmantraut
```

EmailProvider

The **EmailProvider** allows you to get specific email from storage.

```php
<?php

use Sylius\Component\Mailer\Provider\EmailProvider;
use Sylius\Component\Resource\Repository\InMemoryRepository;

$inMemoryRepository = new InMemoryRepository();

$configuration = array(
    'christmas_party_invitation' => array(
        'subject' => 'Christmas party',
        'template' => 'email.html.twig',
        'enabled' => true,
        'sender' => array(
            'name' => 'John',
            'address' => 'john.doe@gmail.com',
        ),
    ),
);

$emailProvider = new EmailProvider($inMemoryRepository, $configuration);

$email = $emailProvider->getEmail('christmas_party_invitation'); // This method will
↪search for an email in your storage or in given configuration.

$email->getCode(); // Output will be christmas_party_invitation.
$email->getSenderAddress(); // Output will be john.doe@gmail.com.
$email->getSenderName(); // Output will be John.
```

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Email

Email object represents an email. Email has the following properties:

Property	Description
id	Unique id of the email
code	Code of the email
enabled	Indicates whether email is available
subject	Subject of the email message
content	Content of the email message
template	Template of the email
senderName	Name of a sender
senderAddress	Address of a sender
createdAt	Date when the email was created
updatedAt	Date of last change

Note: This model implements the *EmailInterface* For more detailed information go to Sylius API Email.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

EmailInterface

This interface should be implemented by model representing a single type of Email.

Note: This interface extends *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API EmailInterface.

Service Interfaces

DefaultSettingsProviderInterface

This interface provides methods for retrieving default sender name nad address.

Note: For more detailed information go to Sylius API DefaultSettingsProviderInterface.

EmailProviderInterface

This interface provides methods for retrieving an email from storage.

Note: For more detailed information go to Sylius API EmailProviderInterface.

Sender

The **Sender** it is way of sending emails

AdapterInterface

This interface provides methods for sending an email. This is an abstraction layer to provide flexibility of mailer component. The Adapter is injected into sender thanks to this you are free to inject your own logic of sending an email, one thing you should do is just implement this interface.

SenderInterface

This interface provides methods for sending an email.

Renderer

AdapterInterface

This interface provides methods for rendering an email. The Adapter is inject into sender for rendering email's content.

Learn more

- *Emails in the Sylius platform* - concept documentation

Order

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

E-Commerce PHP library for creating and managing sales orders.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/order` on Packagist);

- Use the official Git repository (https://github.com/Sylius/Order).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Order

Every order has 2 main identifiers, an ID and a human-friendly number. You can access those by calling `->getId()` and `->getNumber()` respectively. The number is mutable, so you can change it by calling `->setNumber('E001')` on the order instance.

Order Totals

> **Note:** All money amounts in Sylius are represented as "cents" - integers. An order has 3 basic totals, which are all persisted together with the order. The first total is the *items total*, it is calculated as the sum of all item totals. The second total is the *adjustments total*, you can read more about this in next chapter.

```php
<?php

echo $order->getItemsTotal(); //Output will be 1900.
echo $order->getAdjustmentsTotal(); //Output will be -250.

$order->calculateTotal();
echo $order->getTotal(); //Output will be 1650.
```

The main order total is a sum of the previously mentioned values. You can access the order total value using the `->getTotal()` method.

Recalculation of totals can happen by calling `->calculateTotal()` method, using the simplest math. It will also update the item totals.

Items Management

The collection of items (Implementing the `Doctrine\Common\Collections\Collection` interface) can be obtained using the `->getItems()`. To add or remove items, you can simply use the `addItem` and `removeItem` methods.

```php
<?php

use Sylius\Component\Order\Model\Order;
use Sylius\Component\Order\Model\OrderItem;

$order = new Order();
```

(continues on next page)

```php
$item1 = new OrderItem();
$item1->setName('Super cool product');
$item1->setUnitPrice(1999); // 19.99!
$item1->setQuantity(2);

$item2 = new OrderItem();
$item2->setName('Interesting t-shirt');
$item2->setUnitPrice(2549); // 25.49!

$order->addItem($item1);
$order->addItem($item2);
$order->removeItem($item1);
```

Order Item

An order item model has only the **id** property as identifier and it has the order reference, accessible via
->getOrder() method.

Order Item totals

Just like for the order, the total is available via the same method, but the unit price is accessible using the
->getUnitPrice() Each item also can calculate its total, using the quantity (->getQuantity()) and the
unit price.

```php
<?php

use Sylius\Component\Order\Model\OrderItem;

$item = new OrderItem();
$item->setUnitPrice(2000);
$item->setQuantity(4);
$item->calculateTotal();

$item->getTotal(); //Output will be 8000.
```

Applying adjustments to OrderItem

An OrderItem can also hold adjustments.

```php
<?php

use Sylius\Component\Order\Model\OrderItem;
use Sylius\Component\Order\Model\Adjustment;

$adjustment = new Adjustment();
$adjustment->setAmount(1200);
$adjustment->setType('tax');

$item = new OrderItem();
$item->addAdjustment($adjustment);
```

```php
$item->setUnitPrice(2000);
$item->setQuantity(2);
$item->calculateTotal();


$item->getTotal(); //Output will be 5200.
```

Adjustments

Neutral Adjustments

In some cases, you may want to use **Adjustment** just for displaying purposes. For example, when your order items have the tax already included in the price.

Every **Adjustment** instance has the `neutral` property, which indicates if it should be counted against object total.

```php
<?php

use Sylius\Component\Order\Order;
use Sylius\Component\Order\OrderItem;
use Sylius\Component\Order\Adjustment;

$order = new Order();
$tshirt = new OrderItem();
$tshirt->setUnitPrice(4999);

$shippingFees = new Adjustment();
$shippingFees->setAmount(1000);

$tax = new Adjustment();
$tax->setAmount(1150);
$tax->setNeutral(true);

$order->addItem($tshirt);
$order->addAdjustment($shippingFees);
$order->addAdjustment($tax);

$order->calculateTotal();
$order->getTotal();  // Output will be 5999.
```

Negative Adjustments

Adjustments can also have negative amounts, which means that they will decrease the order total by certain amount. Let's add a 5$ discount to the previous example.

```php
<?php

use Sylius\Component\Order\Order;
use Sylius\Component\Order\OrderItem;
use Sylius\Component\Order\Adjustment;

$order = new Order();
$tshirt = new OrderItem();
$tshirt->setUnitPrice(4999);
```

```php
$shippingFees = new Adjustment();
$shippingFees->setAmount(1000);

$tax = new Adjustment();
$tax->setAmount(1150);
$tax->setNeutral(true);

$discount = new Adjustment();
$discount->setAmount(-500);

$order->addItem($tshirt);
$order->addAdjustment($shippingFees);
$order->addAdjustment($tax);
$order->addAdjustment($discount);
$order->calculateTotal();
$order->getTotal(); // Output will be 5499.
```

Locked Adjustments

You can also lock an adjustment, this will ensure that it won't be deleted from order or order item.

```php
<?php

use Sylius\Component\Order\Order;
use Sylius\Component\Order\OrderItem;
use Sylius\Component\Order\Adjustment;

$order = new Order();
$tshirt = new OrderItem();
$tshirt->setUnitPrice(4999);

$shippingFees = new Adjustment();
$shippingFees->setAmount(1000);
$shippingFees->lock();

$discount = new Adjustment();
$discount->setAmount(-500);

$order->addItem($tshirt);
$order->addAdjustment($shippingFees);
$order->addAdjustment($discount);
$order->removeAdjustment($shippingFees);
$order->calculateTotal();
$order->getTotal(); // Output will be 5499.
```

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Order

Order object represents order. Orders have the following properties:

Property	Description
id	Unique id of the order
checkoutCompletedAt	The time at which checkout was completed
number	Number is human-friendly identifier
notes	Additional information about order
items	Collection of items
itemsTotal	Total value of items in order (default 0)
adjustments	Collection of adjustments
adjustmentsTotal	Total value of adjustments (default 0)
total	Calculated total (items + adjustments)
state	State of the order (e.g. "cart", "pending")
createdAt	Date when order was created
updatedAt	Date of last change

Note: This model implements the *OrderInterface* For more detailed information go to Sylius API Order.

OrderItem

OrderItem object represents items in order. OrderItems have the following properties:

Property	Description
id	Unique id of the orderItem
order	Reference to Order
quantity	Items quantity
unitPrice	The price of a single unit
adjustments	Collection of adjustments
adjustmentsTotal	Total of the adjustments in orderItem
total	Total of the orderItem (unitPrice * quantity + adjustmentsTotal)
immutable	Boolean flag of immutability

Note: This model implements the *OrderItemInterface* For more detailed information go to Sylius API OrderItem.

OrderItemUnit

OrderItemUnit object represents every single unit of order (for example `OrderItem` with quantity 5 should have 5 units). OrderItemUnits have the following properties:

Property	Description
id	Unique id of the orderItem
total	Total of the orderItemUnit (orderItem unitPrice + adjustmentsTotal)
orderItem	Reference to OrderItem
adjustments	Collection of adjustments
adjustmentsTotal	Total of the adjustments in orderItem

Note: This model implements the *OrderItemUnitInterface* For more detailed information go to Sylius API OrderItemUnit.

Adjustment

Adjustment object represents an adjustment to the order's or order item's total. Their amount can be positive (charges - taxes, shipping fees etc.) or negative (discounts etc.). Adjustments have the following properties:

Property	Description
id	Unique id of the adjustment
order	Reference to Order
orderItem	Reference to OrderItem
orderItemUnit	Reference to OrderItemUnit
type	Type of the adjustment (e.g. "tax")
label	e.g. "Clothing Tax 9%"
amount	Adjustment amount
neutral	Boolean flag of neutrality
locked	Adjustment lock (prevent from deletion)
originId	Origin id of the adjustment
originType	Origin type of the adjustment
createdAt	Date when adjustment was created
updatedAt	Date of last change

Note: This model implements the *AdjustmentInterface* For more detailed information go to Sylius API Adjustment.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

OrderInterface

This interface should be implemented by model representing a single Order.

Hint: It also contains the default *State Machine*.

Note: This interface extends *TimestampableInterface*, *TimestampableInterface*, *AdjustableInterface* and *CommentAwareInterface*

For more detailed information go to Sylius API OrderInterface.

OrderAwareInterface

This interface provides basic operations for order management. If you want to have orders in your model just implement this interface.

Note: For more detailed information go to Sylius API OrderAwareInterface.

OrderItemInterface

This interface should be implemented by model representing a single OrderItem.

Note: This interface extends the *OrderAwareInterface* and the *AdjustableInterface*,

For more detailed information go to Sylius API OrderItemInterface.

OrderItemUnitInterface

This interface should be implemented by model representing a single OrderItemUnit.

Note: This interface extends the *AdjustableInterface*,

For more detailed information go to Sylius API OrderItemUnitInterface.

AdjustmentInterface

This interface should be implemented by model representing a single Adjustment.

Note: This interface extends the *TimestampableInterface*.

For more detailed information go to Sylius API AdjustmentInterface.

AdjustableInterface

This interface provides basic operations for adjustment management. Use this interface if you want to make a model adjustable.

For example following models implement this interface:

- *Order*
- *OrderItem*

Note: For more detailed information go to Sylius API AdjustableInterface.

CommentInterface

This interface should be implemented by model representing a single Comment.

Note: This interface extends the *TimestampableInterface*

For more detailed information go to Sylius API CommentInterface.

CommentAwareInterface

This interface provides basic operations for comments management. If you want to have comments in your model just implement this interface.

Note: For more detailed information go to Sylius API CommentAwareInterface.

IdentityInterface

This interface should be implemented by model representing a single Identity. It can be used for storing external identifications.

Note: For more detailed information go to Sylius API IdentityInterface.

Services Interfaces

OrderRepositoryInterface

In order to decouple from storage that provides recently completed orders or check if given order's number is already used, you should create repository class which implements this interface.

Note: This interface extends the *RepositoryInterface*.

For more detailed information about the interface go to Sylius API OrderRepositoryInterface.

State Machine

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Order States

Sylius itself uses a complex state machine system to manage all states of the business domain. This component has some sensible default states defined in the **OrderInterface**.

All new **Order** instances have the state `cart` by default, which means they are unconfirmed.

The following states are defined:

Related constant	State	Description
STATE_CART	cart	Unconfirmed order, ready to add/remove items
STATE_NEW	new	Confirmed order
STATE_CANCELLED	cancelled	Cancelled by customer or manager
STATE_FULFILLED	fulfilled	Order has been fulfilled

> **Tip:** Please keep in mind that these states are just default, you can define and use your own. If you use this component with *SyliusOrderBundle* and Symfony, you will have full state machine configuration at your disposal.

Order Transitions

There are following order's transitions by default:

Related constant	Transition
SYLIUS_CREATE	create
SYLIUS_CANCEL	cancel
SYLIUS_FULFILL	fulfill

There is also the default graph name included:

Related constant	Name
GRAPH	sylius_order

> **Note:** All of above transitions and the graph are constant fields in the **OrderTransitions** class.

Processors

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Order processors are responsible for manipulating the orders to apply different predefined adjustments or other modifications based on order state.

OrderProcessorInterface

You can use it when you want to create your own custom processor.

The following code applies 10% discount adjustment to orders above 100€.

```php
<?php

use Sylius\Component\Order\Processor\OrderProcessorInterface;
use Sylius\Component\Order\Model\OrderInterface;
use Sylius\Component\Order\Model\Adjustment;

class DiscountByPriceOrderProcessor implements OrderProcessorInterface
{
    public function process(OrderInterface $order)
    {
        if($order->getTotal() > 10000) {
            $discount10Percent = new Adjustment();
            $discount10Percent->setAmount(-$order->getTotal() / 100 * 10);
            $order->addAdjustment($discount10Percent);
        }
    }
}
```

CompositeOrderProcessor

Composite order processor works as a registry of processors, allowing to run multiple processors in priority order.

Learn more

- *Carts & Orders in the Sylius platform* - concept documentation

Payment

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

PHP library which provides abstraction of payments management.

It ships with default **Payment** and **PaymentMethod** models.

Note: This component does not provide any payment gateway. Integrate it with Payum.

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/payment` on Packagist);

- Use the official Git repository (https://github.com/Sylius/Payment).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Payment

Every payment is represented by a **Payment** instance and has the following properties:

Property	Description
id	Unique id of the payment
method	Payment method associated with this payment
currency	Payment's currency
amount	Payment's amount
state	Payment's state
details	Payment's details
createdAt	Date of creation
updatedAt	Date of the last update

Note: This model implements the *PaymentInterface*.

For more detailed information go to Sylius API Payment.

Hint: All default payment states are available in *Payment States*.

PaymentMethod

Every method of payment is represented by a **PaymentMethod** instance and has the following properties:

Property	Description
id	Unique id of the payment method
code	Unique code of the payment method
name	Payment method's name
enabled	Indicate whether the payment method is enabled
description	Payment method's description
gatewayConfig	Payment method's gateway (and its configuration) to use
position	Payment method's position among other methods
environment	Required app environment
createdAt	Date of creation
updatedAt	Date of the last update

Note: This model implements the *PaymentMethodInterface*.

For more detailed information go to Sylius API PaymentMethod.

PaymentMethodTranslation

This model is used to ensure that different locales have the correct representation of the following payment properties:

Property	Description
id	Unique id of the payment method
name	Payment method's name
description	Payment method's description

Note: This model implements the *PaymentMethodTranslationInterface*.

For more detailed information go to Sylius API PaymentMethodTranslation.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

PaymentInterface

This interface should be implemented by any custom model representing a payment. Also it keeps all of the default *Payment States*.

Note: This interface extends the *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API PaymentInterface.

PaymentMethodInterface

In order to create a custom payment method class, which could be used by other models or services from this component, it needs to implement this interface.

Note: This interface extends the *TimestampableInterface* and the *PaymentMethodTranslationInterface*.

For more detailed information go to Sylius API PaymentMethodInterface.

PaymentMethodsAwareInterface

This interface should be implemented by any custom storage used to store representations of the payment method.

Note: For more detailed information go to Sylius API PaymentMethodsAwareInterface.

PaymentMethodTranslationInterface

This interface is needed in creating a custom payment method translation class, which then could be used by the payment method itself.

Note: For more detailed information go to Sylius API PaymentMethodTranslationInterface.

PaymentSourceInterface

This interface needs to be implemented by any custom payment source.

Note: For more detailed information go to Sylius API PaymentSourceInterface.

PaymentsSubjectInterface

Any container which manages multiple payments should implement this interface.

Note: For more detailed information go to Sylius API PaymentsSubjectInterface.

Service Interfaces

PaymentMethodRepositoryInterface

This interface should be implemented by your custom repository, used to handle payment method objects.

Note: For more detailed information go to Sylius API PaymentMethodRepositoryInterface.

State Machine

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Payment States

The following payment states are available by default:

Related constant	State	Description
STATE_CART	cart	Initial; Before the subject of payment is completed
STATE_NEW	new	After completion of the payment subject
STATE_PROCESSING	processing	Payment which is in process of verification
STATE_COMPLETED	completed	Completed payment
STATE_FAILED	failed	Payment has failed
STATE_CANCELLED	cancelled	Cancelled by a customer or manager
STATE_REFUNDED	refunded	A completed payment which has been refunded
STATE_UNKNOWN	unknown	Auxiliary state for handling external states

---

Note: All the above states are constant fields in the *PaymentInterface*.

Payment Transitions

The following payment transitions are available by default:

Related constant	Transition
SYLIUS_CREATE	create
SYLIUS_PROCESS	process
SYLIUS_COMPLETE	complete
SYLIUS_FAIL	fail
SYLIUS_CANCEL	cancel
SYLIUS_REFUND	refund

There's also the default graph name included:

Related constant	Name
GRAPH	sylius_payment

Note: All of above transitions and the graph are constant fields in the **PaymentTransitions** class.

Learn more

- *Payments in the Sylius platform* - concept documentation

Product

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Powerful products catalog for PHP applications.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/product` on Packagist).
- Use the official Git repository (https://github.com/Sylius/Product).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Creating a product

```php
<?php

use Sylius\Component\Product\Model\Product;

$product = new Product();

$product->getCreatedAt(); // Returns the \DateTime when it was created.
```

Product attributes management

```php
<?php

use Sylius\Component\Product\Model\Attribute;
use Sylius\Component\Product\Model\AttributeValue;
use Doctrine\Common\Collections\ArrayCollection;

$attribute = new Attribute();

$colorGreen = new AttributeValue();
$colorRed = new AttributeValue();

$attributes = new ArrayCollection();

$attribute->setName('Color');

$colorGreen->setValue('Green');
$colorRed->setValue('Red');

$colorGreen->setAttribute($attribute);
$colorRed->setAttribute($attribute);

$product->addAttribute($colorGreen);
$product->hasAttribute($colorGreen); // Returns true.
$product->removeAttribute($colorGreen);

$attributes->add($colorGreen);
$attributes->add($colorRed);
$product->setAttributes($attributes);

$product->hasAttributeByName('Color');
$product->getAttributeByName('Color'); // Returns $colorGreen.

$product->getAttributes(); // Returns $attributes.
```

Note: Only instances of **AttributeValue** from the *Product* component can be used with the *Product* model.

Hint: The `getAttributeByName` will only return the first occurrence of **AttributeValue** assigned to the **Attribute** with specified name, the rest will be omitted.

Product variants management

```php
<?php

use Sylius\Component\Product\Model\Variant;

$variant = new Variant();
$availableVariant = new Variant();

$variants = new ArrayCollection();

$availableVariant->setAvailableOn(new \DateTime());
```

(continues on next page)

```php
$product->hasVariants(); // return false

$product->addVariant($variant);
$product->hasVariant($variant); // returns true
$product->hasVariants(); // returns true
$product->removeVariant($variant);

$variants->add($variant);
$variants->add($availableVariant);

$product->setVariants($variants);

$product->getVariants(); // Returns an array containing $variant and
→$availableVariant.
```

Note: Only instances of **Variant** from the *Product* component can be used with the *Product* model.

Product options management

```php
<?php

use Sylius\Component\Product\Model\Option;

$firstOption = new Option();
$secondOption = new Option();

$options = new ArrayCollection();

$product->addOption($firstOption);
$product->hasOption($firstOption); // Returns true.
$product->removeOption($firstOption);

$options->add($firstOption);
$options->add($secondOption);

$product->setOptions($options);
$product->hasOptions(); // Returns true.
$product->getOptions(); // Returns an array containing all inserted options.
```

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Product

The **Product** model represents every unique product in the catalog. By default it contains the following properties:

Property	Description
id	Unique id of the product
name	Product's name taken from the `ProductTranslation`
slug	Product's urlized name taken from the `ProductTranslation`
description	Product's description taken from the `ProductTranslation`
metaKeywords	Product's meta keywords taken from the `ProductTranslation`
metaDescription	Product's meta description taken from the `ProductTranslation`
attributes	Attributes assigned to this product
variants	Variants assigned to this product
options	Options assigned to this product
createdAt	Product's date of creation
updatedAt	Product's date of update

Note: This model uses the *Using TranslatableTrait* and implements the *ProductInterface*.

For more detailed information go to Sylius API Product.

ProductTranslation

This model is responsible for keeping a translation of product's simple properties according to given locale. By default it has the following properties:

Property	Description
id	Unique id of the product translation

Note: This model extends the *Implementing AbstractTranslation* class and implements the *ProductTranslationInterface*.

For more detailed information go to Sylius API ProductTranslation.

AttributeValue

This **AttributeValue** extension ensures that it's **subject** is an instance of the *ProductInterface*.

Note: This model extends the *AttributeValue* and implements the *AttributeValueInterface*.

For more detailed information go to Sylius API AttributeValue.

Variant

This **Variant** extension ensures that it's **object** is an instance of the *ProductInterface* and provides an additional property:

Property	Description
availableOn	The date indicating when a product variant is available

---

Note: This model implements the *ProductVariantInterface*.

For more detailed information go to Sylius API Variant.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

ProductInterface

This interface should be implemented by models characterizing a product.

Note: This interface extends *SlugAwareInterface*, *TimestampableInterface* and *ProductTranslationInterface*.

For more information go to Sylius API ProductInterface.

ProductTranslationInterface

This interface should be implemented by models used for storing a single translation of product fields.

Note: This interface extends the *SlugAwareInterface*.

For more information go to Sylius API ProductTranslationInterface.

AttributeValueInterface

This interfaces should be implemented by models used to bind an attribute and a value to a specific product.

Note: This interface extends the *AttributeValueInterface*.

For more information go to Sylius API AttributeValueInterface.

ProductVariantInterface

This interface should be implemented by models binding a product with a specific combination of attributes.

Learn more

- *Products in the Sylius platform* - concept documentation

Promotion

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
> it. You can help us making documentation up to date via Sylius Github. Thank you!

Super-flexible promotions system with support of complex rules and actions. Coupon codes included!

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
> it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/promotion` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Promotion).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use
> it. You can help us making documentation up to date via Sylius Github. Thank you!

In order to benefit from the component's features at first you need to create a basic class that will implement the
PromotionSubjectInterface. Let's assume that you would like to have a system that applies promotions on Tickets.
Your **Ticket** class therefore will implement the *CountablePromotionSubjectInterface* to give you an ability to count
the subjects for promotion application purposes.

```php
<?php

namespace AppBundle\Entity;

use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use Sylius\Component\Promotion\Model\CountablePromotionSubjectInterface;
use Sylius\Component\Promotion\Model\PromotionSubjectInterface;
use Sylius\Component\Promotion\Model\PromotionInterface;

class Ticket implements CountablePromotionSubjectInterface
{
    /**
     * @var int
     */
    private $quantity;

    /**
```

(continues on next page)

```php
 * @var Collection
 */
private $promotions;

/**
 * @var int
 */
private $unitPrice;

public function __construct()
{
    $this->promotions = new ArrayCollection();
}
/**
 * @return int
 */
public function getQuantity()
{
    return $this->quantity;
}


/**
 * @param int $quantity
 */
public function setQuantity($quantity)
{
    $this->quantity = $quantity;
}


/**
 * {@inheritdoc}
 */
public function getPromotions()
{
    return $this->promotions;
}


/**
 * {@inheritdoc}
 */
public function hasPromotion(PromotionInterface $promotion)
{
    return $this->promotions->contains($promotion);
}


/**
 * {@inheritdoc}
 */
public function getPromotionSubjectTotal()
{
    //implementation
}


/**
 * {@inheritdoc}
 */
public function addPromotion(PromotionInterface $promotion)
```

```php
    {
        if (!$this->hasPromotion($promotion)) {
            $this->promotions->add($promotion);
        }
    }

    /**
     * {@inheritdoc}
     */
    public function removePromotion(PromotionInterface $promotion)
    {
        if($this->hasPromotion($promotion))
        {
            $this->promotions->removeElement($promotion);
        }
    }

    /**
     * {@inheritdoc}
     */
    public function getPromotionSubjectCount()
    {
        return $this->getQuantity();
    }

    /**
     * @return int
     */
    public function getUnitPrice()
    {
        return $this->unitPrice;
    }

    /**
     * @param int $price
     */
    public function setUnitPrice($price)
    {
        $this->unitPrice = $price;
    }

    /**
     * @return int
     */
    public function getTotal()
    {
        return $this->getUnitPrice() * $this->getQuantity();
    }
}
```

PromotionProcessor

The component provides us with a **PromotionProcessor** which checks all rules of a subject and applies configured actions if rules are eligible.

```
<?php

use Sylius\Component\Promotion\Processor\PromotionProcessor;
use AppBundle\Entity\Ticket;

/**
 * @param PromotionRepositoryInterface          $repository
 * @param PromotionEligibilityCheckerInterface  $checker
 * @param PromotionApplicatorInterface          $applicator
 */
$processor = new PromotionProcessor($repository, $checker, $applicator);

$subject = new Ticket();

$processor->process($subject);
```

Note: It implements the *PromotionProcessorInterface*.

CompositePromotionEligibilityChecker

The Promotion component provides us with a delegating service - the **CompositePromotionEligibilityChecker** that checks if the promotion rules are eligible for a given subject. Below you can see how it works:

> **Warning:** Remember! That before you start using rule checkers you need to have two Registries - rule checker registry and promotion action registry. In these you have to register your rule checkers and promotion actions. You will also need working services - 'item_count' rule checker service for our example:

```
<?php

use Sylius\Component\Promotion\Model\Promotion;
use Sylius\Component\Promotion\Model\PromotionAction;
use Sylius\Component\Promotion\Model\PromotionRule;
use Sylius\Component\Promotion\Checker\CompositePromotionEligibilityChecker;
use AppBundle\Entity\Ticket;

$checkerRegistry = new ServiceRegistry(
↪'Sylius\Component\Promotion\Checker\RuleCheckerInterface');
$actionRegistry = new ServiceRegistry(
↪'Sylius\Component\Promotion\Model\PromotionActionInterface');
$ruleRegistry = new ServiceRegistry(
↪'Sylius\Component\Promotion\Model\PromotionRuleInterface');

$dispatcher = new EventDispatcher();

/**
 * @param ServiceRegistryInterface $registry
 * @param EventDispatcherInterface $dispatcher
 */
$checker = new CompositePromotionEligibilityChecker($checkerRegistry, $dispatcher);

$itemCountChecker = new ItemCountRuleChecker();
```

```php
$checkerRegistry->register('item_count', $itemCountChecker);

// Let's create a new promotion
$promotion = new Promotion();
$promotion->setName('Test');

// And a new action for that promotion, that will give a fixed discount of 10
$action = new PromotionAction();
$action->setType('fixed_discount');
$action->setConfiguration(array('amount' => 10));
$action->setPromotion($promotion);

$actionRegistry->register('fixed_discount', $action);

// That promotion will also have a rule - works for item amounts over 2
$rule = new PromotionRule();
$rule->setType('item_count');

$configuration = array('count' => 2);
$rule->setConfiguration($configuration);

$ruleRegistry->register('item_count', $rule);

$promotion->addRule($rule);

// Now we need an object that implements the PromotionSubjectInterface
// so we will use our custom Ticket class.
$subject = new Ticket();

$subject->addPromotion($promotion);
$subject->setQuantity(3);
$subject->setUnitPrice(10);

$checker->isEligible($subject, $promotion); // Returns true
```

Note: It implements the *PromotionEligibilityCheckerInterface*.

PromotionApplicator

In order to automate the process of promotion application the component provides us with a Promotion Applicator, which is able to apply and revert single promotions on a subject implementing the **PromotionSubjectInterface**.

```php
<?php

use Sylius\Component\Promotion\PromotionAction\PromotionApplicator;
use Sylius\Component\Promotion\Model\Promotion;
use Sylius\Component\Registry\ServiceRegistry;
use AppBundle\Entity\Ticket;

// In order for the applicator to work properly you need to have your actions created
// and registered before.
$registry = new ServiceRegistry(
    'Sylius\Component\Promotion\Model\PromotionActionInterface');
```

```php
$promotionApplicator = new PromotionApplicator($registry);

$promotion = new Promotion();

$subject = new Ticket();
$subject->addPromotion($promotion);

$promotionApplicator->apply($subject, $promotion);

$promotionApplicator->revert($subject, $promotion);
```

Note: It implements the *PromotionApplicatorInterface*.

PromotionCouponGenerator

In order to automate the process of coupon generation the component provides us with a Coupon Generator.

```php
<?php

use Sylius\Component\Promotion\Model\Promotion;
use Sylius\Component\Promotion\Generator\PromotionCouponGeneratorInstruction;
use Sylius\Component\Promotion\Generator\PromotionCouponGenerator;

$promotion = new Promotion();

$instruction = new PromotionCouponGeneratorInstruction(); // $amount = 5 by default

/**
 * @param RepositoryInterface    $repository
 * @param EntityManagerInterface $manager
 */
$generator = new PromotionCouponGenerator($repository, $manager);

//This will generate and persist 5 coupons into the database
//basing on the instruction provided for the given promotion object
$generator->generate($promotion, $instruction);

// We can also generate one unique code, and assign it to a new Coupon.
$code = $generator->generateUniqueCode();
$coupon = new Coupon();
$coupon->setCode($code);
```

Checkers

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ItemCountRuleChecker

You can use it when your subject implements the *CountablePromotionSubjectInterface*:

```php
<?php

$itemCountChecker = new ItemCountRuleChecker();
// a Subject that implements the CountablePromotionSubjectInterface
$subject->setQuantity(3);

$configuration = array('count' => 2);

$itemCountChecker->isEligible($subject, $configuration); // returns true
```

ItemTotalRuleChecker

If your subject implements the *PromotionSubjectInterface* you can use it with this checker.

```php
<?php

$itemTotalChecker = new ItemTotalRuleChecker();

// a Subject that implements the PromotionSubjectInterface
// Let's assume the subject->getSubjectItemTotal() returns 199

$configuration = array('amount' => 199);

$itemTotalChecker->isEligible($subject, $configuration); // returns true
```

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Promotion

The promotion is represented by a **Promotion** instance. It has the following properties as default:

Property	Description
id	Unique id of the promotion
code	Unique code of the promotion
name	Promotion's name
description	Promotion's description
priority	When exclusive, promotion with top priority will be applied
exclusive	Cannot be applied together with other promotions
usageLimit	Promotion's usage limit
used	Number of times this coupon has been used
startsAt	Start date
endsAt	End date
couponBased	Whether this promotion is triggered by a coupon
coupons	Associated coupons
rules	Associated rules
actions	Associated actions
createdAt	Date of creation
updatedAt	Date of update

Note: This model implements the *PromotionInterface* .

Coupon

The coupon is represented by a **Coupon** instance. It has the following properties as default:

Property	Description
id	Unique id of the coupon
code	Coupon's code
usageLimit	Coupon's usage limit
used	Number of times the coupon has been used
promotion	Associated promotion
expiresAt	Expiration date
createdAt	Date of creation
updatedAt	Date of update

Note: This model implements the *CouponInterface*.

PromotionRule

The promotion rule is represented by a **PromotionRule** instance. PromotionRule is a requirement that has to be satisfied by the promotion subject. It has the following properties as default:

Property	Description
id	Unique id of the coupon
type	Rule's type
configuration	Rule's configuration
promotion	Associated promotion

Note: This model implements the *PromotionRuleInterface*.

PromotionAction

The promotion action is represented by an **PromotionAction** instance. PromotionAction takes place if the rules of a promotion are satisfied. It has the following properties as default:

Property	Description
id	Unique id of the action
type	Rule's type
configuration	Rule's configuration
promotion	Associated promotion

Note: This model implements the *PromotionActionInterface*.

Interfaces

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

PromotionSubjectInterface

To characterize an object with attributes and options from a promotion, the object class needs to implement the **PromotionSubjectInterface**.

Note: You will find more information about this interface in Sylius API PromotionSubjectInterface.

PromotionInterface

This interface should be implemented by models representing a **Promotion**.

Note: This interface extends the *CodeAwareInterface* and *TimestampableInterface*.

You will find more information about this interface in Sylius API PromotionInterface.

PromotionActionInterface

This interface should be implemented by models representing an **PromotionAction**.

An **PromotionActionInterface** has two defined types by default:

Related constant	Type
TYPE_FIXED_DISCOUNT	fixed_discount
TYPE_PERCENTAGE_DISCOUNT	percentage_discount

Note: You will find more information about this interface in Sylius API PromotionActionInterface.

CouponInterface

This interface should be implemented by models representing a **Coupon**.

Note: This interface extends the *CodeAwareInterface* and the *TimestampableInterface*.

You will find more information about this interface in Sylius API CouponInterface.

PromotionRuleInterface

This interface should be implemented by models representing a **PromotionRule**.

A **PromotionRuleInterface** has two defined types by default:

Related constant	Type
TYPE_ITEM_TOTAL	item_total
TYPE_ITEM_COUNT	item_count

Note: You will find more information about this interface in Sylius API PromotionRuleInterface.

CountablePromotionSubjectInterface

To be able to count the object's promotion subjects, the object class needs to implement the `CountablePromotionSubjectInterface`.

Note: This interface extends the *PromotionSubjectInterface*.

You will find more information about this interface in Sylius API CountablePromotionSubjectInterface.

PromotionCouponAwarePromotionSubjectInterface

To make the object able to get its associated coupon, the object class needs to implement the `PromotionCouponAwarePromotionSubjectInterface`.

Note: This interface extends the *PromotionSubjectInterface*.

You will find more information about this interface in Sylius API PromotionCouponAwarePromotionSubjectInterface.

PromotionCouponsAwareSubjectInterface

To make the object able to get its associated coupons collection, the object class needs to implement the `PromotionCouponsAwareSubjectInterface`.

Note: This interface extends the *PromotionSubjectInterface*.

You will find more information about this interface in Sylius API PromotionCouponsAwareSubjectInterface.

Services Interfaces

PromotionEligibilityCheckerInterface

Services responsible for checking the promotions eligibility on the promotion subjects should implement this interface.

Note: You will find more information about this interface in Sylius API PromotionEligibilityCheckerInterface.

RuleCheckerInterface

Services responsible for checking the rules eligibility should implement this interface.

Note: You will find more information about this interface in Sylius API RuleCheckerInterface.

PromotionApplicatorInterface

Service responsible for applying promotions in your system should implement this interface.

Note: You will find more information about this interface in Sylius API PromotionApplicatorInterface.

PromotionProcessorInterface

Service responsible for checking all rules and applying configured actions if rules are eligible in your system should implement this interface.

Note: You will find more information about this interface in Sylius API PromotionProcessorInterface.

PromotionRepositoryInterface

In order to be able to find active promotions in your system you should create a repository class which implements this interface.

Note: This interface extends the *RepositoryInterface*.

For more detailed information about this interface go to Sylius API PromotionRepositoryInterface.

PromotionCouponGeneratorInterface

In order to automate the process of coupon generation your system needs to have a service that will implement this interface.

Note: For more detailed information about this interface go to Sylius API PromotionCouponGeneratorInterface.

PromotionActionCommandInterface

This interface should be implemented by services that execute actions on the promotion subjects.

Note: You will find more information about this interface in Sylius API PromotionActionCommandInterface.

Learn more

- *Promotions in the Sylius platform* - concept documentation

Registry

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Simple registry component useful for all types of applications.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/registry` on Packagist);

- Use the official Git repository (https://github.com/Sylius/Registry).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

A registry object acts as a collection of objects. The sylius **ServiceRegistry** allows you to store objects which implement a specific interface.

ServiceRegistry

To create a new **ServiceRegistry** you need to determine what kind of interface should be kept inside.

For the sake of examples lets use the *RuleCheckerInterface* from the *Promotion* component.

```php
<?php

use Sylius\Component\Registry\ServiceRegistry;

$registry = new ServiceRegistry(
↪'Sylius\Component\Promotion\Checker\RuleCheckerInterface');
```

Once you've done that you can manage any object with the corresponding interface.

So for starters, lets add some services:

```php
<?php

use Sylius\Component\Promotion\Checker\Rule\ItemTotalRuleChecker;
use Sylius\Component\Promotion\Checker\Rule\CartQuantityRuleChecker;

$registry->register('item_total', new ItemTotalRuleChecker());
$registry->register('cart_quantity', new CartQuantityRuleChecker());
```

> **Hint:** The first parameter of `register` is incredibly important, as we will use it for all further operations. Also it's the key at which our service is stored in the array returned by `all` method.

After specifying the interface and inserting services, we can manage them:

```php
<?php

$registry->has('item_total'); // returns true

$registry->get('item_total'); // returns the ItemTotalRuleChecker we inserted earlier
↪on

$registry->all(); // returns an array containing both rule checkers
```

Removing a service from the registry is as easy as adding:

```
<?php

$registry->unregister('item_total');

$registry->has('item_total'); // now returns false
```

Note: This service implements the *ServiceRegistryInterface*.

Caution: This service throws:

- \InvalidArgumentException when you try to `register` a service which doesn't implement the specified interface
- *ExistingServiceException*
- *NonExistingServiceException*

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ServiceRegistryInterface

This interface should be implemented by a service responsible for managing various services.

Note: For more detailed information go to Sylius API ServiceRegistryInterface.

Exceptions

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ExistingServiceException

This exception is thrown when you try to **register** a service that is already in the registry.

Note: This exception extends the \InvalidArgumentException.

NonExistingServiceException

This exception is thrown when you try to **unregister** a service which is not in the registry.

Note: This exception extends the \InvalidArgumentException.

Resource

Domain management abstraction for PHP. It provides interface for most common operations on the application resources.

Installation

We assume you're familiar with Composer, a dependency manager for PHP. Use following command to add the component to your *composer.json* and download package.

If you have Composer installed globally.

```
$ composer require sylius/resource
```

Otherwise you have to download .phar file.

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require sylius/resource
```

Model interfaces

ResourceInterface

This primary interface marks the model as a resource and will ask you to implement the following methods to your model:

Method	Description	Returned value
getId()	Get identifier	mixed

TimestampableInterface

This interface will ask you to implement the following methods to your model, they will use by the timestampable Doctrine2 extension.

Method	Description	Returned value
getCreatedAt()	Get creation time	DateTime
setCreatedAt(DateTime $createdAt)	Set creation time	void
getUpdatedAt()	Get the time of last update	DateTime
setUpdatedAt(DateTime $updatedAt)	Set the time of last update	void

ToggleableInterface

This interface can be applied to every toggleable model and will ask you to implement the following methods to your model:

Method	Description	Returned value
isEnabled()	Return current status	bool
enable()	Set as enabled	void
disable()	Set as disabled	void
setEnabled(bool $enabled)	Set current status	void

CodeAwareInterface

This interface can be applied to every code aware model and will ask you to implement the following methods to your model:

Method	Description	Returned value
getCode()	Get code	string
setCode(string $code)	Set code	void

SlugAwareInterface

This interface is used by sluggable Doctrine2 extension and will ask you to implement the following methods to your model:

Method	Description	Returned value
getSlug()	Get slug	string
setSlug(string $slug = null)	Set slug	void

TranslatableInterface

This interface should be implemented by a model used in more than one language.

Hint: Although you can implement this interface in your class, it's easier to just use the *Using TranslatableTrait* class.

Note: For more detailed information go to Sylius API TranslatableInterface.

TranslationInterface

This interface should be implemented by a model responsible for keeping a single translation.

Hint: And as above, although you are completely free to create your own class implementing this interface, it's already implemented in the *Implementing AbstractTranslation* class.

Note: For more detailed information go to Sylius API TranslationInterface.

Factory Interface

FactoryInterface

Interface implemented by all Factory services for Resources. To learn more go to *Creating Resources*.

Method	Description
createNew()	Create a new instance of your resource

Repository Interfaces

RepositoryInterface

This interface should be implemented by every Resource Repository service.

Method	Description
createPaginator(array $criteria = null, array $orderBy = null)	Get paginated collection of your resources

Service interfaces

LocaleProviderInterface

This interface should be implemented by a service responsible for managing locales.

Note: For more detailed information go to Sylius API LocaleProviderInterface.

TranslatableRepositoryInterface

This interface should be implemented by a repository responsible for keeping the **LocaleProvider** and an array of fields

This interface expects you to implement a way of setting an instance of **LocaleProviderInterface**, and an array of translatable fields into your custom repository.

Note: This interface extends the *RepositoryInterface*.

For more detailed information go to Sylius API TranslatableResourceRepositoryInterface.

Translations

Implementing AbstractTranslation

First let's create a class which will keep our translatable properties:

```php
<?php

namespace Example\Model;

use Sylius\Component\Resource\Model\AbstractTranslation;

class BookTranslation extends AbstractTranslation
{
    /**
     * @var string
     */
    private $title;

    /**
     * @return string
     */
    public function getTitle()
    {
        return $this->title;
    }

    /**
     * @param string $title
     */
    public function setTitle($title)
    {
        $this->title = $title;
    }
}
```

Using TranslatableTrait

Now the following class will be actually capable of translating the **title**:

```php
<?php

namespace Example\Model;

use Sylius\Component\Resource\Model\TranslatableInterface;
use Sylius\Component\Resource\Model\TranslatableTrait;

class Book implements TranslatableInterface
{
    use TranslatableTrait;

    /**
     * @return string
     */
    public function getTitle()
    {
        return $this->getTranslation()->getTitle();
    }

    /**
     * @param string $title
     */
```

```php
    public function setTitle($title)
    {
        $this->getTranslation()->setTitle($title);
    }
}
```

Note: As you could notice, inside both methods we use the `getTranslation` method. More specified explanation on what it does is described further on.

Using Translations

Once we have both classes implemented we can start translating. So first we need to create a few instances of our translation class:

```php
<?php

use Example\Model\Book;
use Example\Model\BookTranslation;

$englishBook = new BookTranslation();
$englishBook->setLocale('en');
$englishBook->setTitle("Harry Potter and the Philosopher's Stone");
// now we have a title set for the english locale

$spanishBook = new BookTranslation();
$spanishBook->setLocale('es');
$spanishBook->setTitle('Harry Potter y la Piedra Filosofal');
// spanish

$germanBook = new BookTranslation();
$germanBook->setLocale('de');
$germanBook->setTitle('Harry Potter und der Stein der Weisen');
// and german
```

When we already have our translations, we can work with the **Book**:

```php
<?php

$harryPotter = new Book();

$harryPotter->addTranslation($englishBook);
$harryPotter->addTranslation($spanishBook);
$harryPotter->addTranslation($germanBook);

$harryPotter->setFallbackLocale('en'); // the locale which translation should be used
↪by default

$harryPotter->setCurrentLocale('es'); // the locale which translation we want to get

$harryPotter->getTitle(); // returns 'Harry Potter y la Piedra Filosofal'

$harryPotter->setCurrentLocale('ru');
```

```
$harryPotter->getTitle(); // now returns "Harry Potter and the Philosopher's Stone"
                          // as the translation for chosen locale is unavailable,
                          // instead the translation for fallback locale is used
```

You can always use the `getTranslation` method by itself, but the same principal is in play:

```php
<?php

$harryPotter->getTranslation('de');  // returns $germanBook
// but
$harryPotter->getTranslation();
// and
$harryPotter->getTranslation('hi');
// both return $englishBook
```

> **Caution:** The `getTranslation` method throws \RuntimeException in two cases:
>
> • No locale has been specified in the parameter and the current locale is undefined
>
> • No fallback locale has been set

LocaleProvider

This service provides you with an easy way of managing locales. The first parameter set in it's constructor is the current locale and the second, fallback.

In this example let's use the provider with our *Book* class which uses the *Using TranslatableTrait*:

```php
<?php

use Example\Model\Book;
use Sylius\Component\Resource\Provider\LocaleProvider;

$provider = new LocaleProvider('de', 'en');

$book = new Book();

$book->setCurrentLocale($provider->getCurrentLocale());
$book->setFallbackLocale($provider->getFallbackLocale());

$book->getCurrentLocale(); // returns 'de'
$book->getFallbackLocale(); // returns 'en'
```

. . . and with an *Implementing AbstractTranslation* class such as the exemplary *BookTranslation* it goes:

```php
<?php

use Example\Model\BookTranslation;
use Sylius\Component\Resource\Provider\LocaleProvider;

$provider = new LocaleProvider('de', 'en');

$bookTranslation = new BookTranslation();
```

```
$bookTranslation->setLocale($provider->getCurrentLocale());

$bookTranslation->getLocale(); // returns 'de'
```

Note: This service implements the *LocaleProviderInterface*.

Summary

phpspec examples

```
$ composer install
$ bin/phpspec run -fpretty --verbose
```

Bug tracking

This component uses GitHub issues. If you have found bug, please create an issue.

Learn more

- *Resource Layer in the Sylius platform* - concept documentation

Shipping

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Shipments and shipping methods management for PHP E-Commerce applications. It contains flexible calculators system for computing the shipping costs.

Installation

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/shipping` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Shipping).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In all examples is used an exemplary class implementing **ShippableInterface**, which looks like:

```php
<?php

declare(strict_types=1);

use Sylius\Component\Shipping\Model\ShippableInterface;
use Sylius\Component\Shipping\Model\ShippingCategoryInterface;

class Wardrobe implements ShippableInterface
{
    /**
     * @var ShippingCategoryInterface
     */
    private $category;

    /**
     * @var float
     */
    private $weight;

    /**
     * @var float
     */
    private $volume;

    /**
     * @var float
     */
    private $width;

    /**
     * @var float
     */
    private $height;

    /**
     * @var float
     */
    private $depth;

    /**
     * {@inheritdoc}
     */
    public function getShippingWeight(): float
    {
        return $this->weight;
    }

    /**
```

(continues on next page)

```php
     * {@inheritdoc}
     */
    public function setShippingWeight(float $weight): void
    {
        $this->weight = $weight;
    }

    /**
     * {@inheritdoc}
     */
    public function getShippingVolume(): float
    {
        return $this->volume;
    }

    /**
     * @param int $volume
     */
    public function setShippingVolume(float $volume)
    {
        $this->volume = $volume;
    }

    /**
     * {@inheritdoc}
     */
    public function getShippingWidth(): float
    {
        return $this->width;
    }


    /**
     * {@inheritdoc}
     */
    public function setShippingWidth(float $width)
    {
        $this->width = $width;
    }

    /**
     * {@inheritdoc}
     */
    public function getShippingHeight(): float
    {
        return $this->height;
    }

    /**
     * {@inheritdoc}
     */
    public function setShippingHeight(float $height)
    {
        $this->height = $height;
    }

    /**
```

```php
     * {@inheritdoc}
     */
    public function getShippingDepth(): float
    {
        return $this->depth;
    }

    /**
     * {@inheritdoc}
     */
    public function setShippingDepth(float $depth)
    {
        $this->depth = $depth;
    }

    /**
     * {@inheritdoc}
     */
    public function getShippingCategory(): ShippingCategoryInterface
    {
        return $this->category;
    }

    /**
     * {@inheritdoc}
     */
    public function setShippingCategory(ShippingCategoryInterface $category)
    {
        $this->category = $category;
    }
}
```

Shipping Category

Every shipping category has three identifiers, an ID, code and name. You can access those by calling `->getId()`, `->getCode()` and `->getName()` methods respectively. The name is mutable, so you can change them by calling and `->setName('Regular')` on the shipping category instance.

Shipping Method

Every shipping method has three identifiers, an ID code and name. You can access those by calling `->getId()`, `->getCode()` and `->getName()` methods respectively. The name is mutable, so you can change them by calling `->setName('FedEx')` on the shipping method instance.

Setting Shipping Category

Every shipping method can have shipping category. You can simply set or unset it by calling `->setCategory()`.

```php
<?php

use Sylius\Component\Shipping\Model\ShippingMethod;
```

```php
use Sylius\Component\Shipping\Model\ShippingCategory;
use Sylius\Component\Shipping\Model\ShippingMethodInterface;

$shippingCategory = new ShippingCategory();
$shippingCategory->setName('Regular'); // Regular weight items

$shippingMethod = new ShippingMethod();
$shippingMethod->setCategory($shippingCategory); //default null, detach
$shippingMethod->getCategory(); // Output will be ShippingCategory object
$shippingMethod->setCategory(null);
```

Shipping Method Translation

ShippingMethodTranslation allows shipping method's name translation according to given locales. To see how to use translation please go to *Using Translations*.

Shipment Item

You can use a **ShippingItem** for connecting a shippable object with a proper **Shipment**. Note that a **ShippingItem** can exist without a **Shipment** assigned.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Model\ShipmentInterface;

$shipment = new Shipment();
$wardrobe = new Wardrobe();
$shipmentItem = new ShipmentItem();

$shipmentItem->setShipment($shipment);
$shipmentItem->getShipment(); // returns shipment object
$shipmentItem->setShipment(null);

$shipmentItem->setShippable($wardrobe);
$shipmentItem->getShippable(); // returns shippable object

$shipmentItem->getShippingState(); // returns const STATE_READY
$shipmentItem->setShippingState(ShipmentInterface::STATE_SOLD);
```

Shipment

Every **Shipment** can have the types of state defined in the **ShipmentInterface** and the **ShippingMethod**, which describe the way of delivery.

```php
<?php

use Sylius\Component\Shipping\Model\ShippingMethod;
use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentInterface;
```

```php
$shippingMethod = new ShippingMethod();

$shipment = new Shipment();
$shipment->getState(); // returns const checkout
$shipment->setState(ShipmentInterface::STATE_CANCELLED);

$shipment->setMethod($shippingMethod);
$shipment->getMethod();
```

Adding shipment item

You can add many shipment items to shipment, which connect shipment with shippable object.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;

$shipmentItem = new ShipmentItem();
$shipment = new Shipment();

$shipment->addItem($shipmentItem);
$shipment->hasItem($shipmentItem); // returns true
$shipment->getItems(); // returns collection of shipment items
$shipment->getShippingItemCount(); // returns 1
$shipment->removeItem($shipmentItem);
```

Tracking shipment

You can also define tracking code for your shipment:

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;

$shipment->isTracked();// returns false
$shipment->setTracking('5346172074');
$shipment->getTracking(); // returns 5346172074
$shipment->isTracked();// returns true
```

RuleCheckerInterface

This example shows how use an exemplary class implementing **RuleCheckerInterface**.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Model\Rule;
use Sylius\Component\Shipping\Checker\ItemCountRuleChecker;
```

```
$rule = new Rule();
$rule->setConfiguration(array('count' => 5, 'equal' => true));

$wardrobe = new Wardrobe();

$shipmentItem = new ShipmentItem();
$shipmentItem->setShippable($wardrobe);

$shipment = new Shipment();
$shipment->addItem($shipmentItem);

$ruleChecker = new ItemCountRuleChecker();
$ruleChecker->isEligible($shipment, $rule->getConfiguration()); // returns false,␣
↪because
// quantity of shipping item in shipment is smaller than count from rule's␣
↪configuration
```

Hint: You can read more about each of the available checkers in the *Checkers* chapter.

Delegating calculation to correct calculator instance

DelegatingCalculator class delegates the calculation of charge for particular shipping subject to a correct calculator instance, based on the name defined on the shipping method. It uses **ServiceRegistry** to keep all calculators registered inside container. The calculators are retrieved by name.

```php
<?php

use Sylius\Component\Shipping\Model\ShippingMethod;
use Sylius\Component\Shipping\Calculator\DefaultCalculators;
use Sylius\Component\Shipping\Calculator\PerItemRateCalculator;
use Sylius\Component\Shipping\Calculator\FlexibleRateCalculator;
use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Calculator\DelegatingCalculator;
use Sylius\Component\Registry\ServiceRegistry;

$configuration = array(
    'first_item_cost'      => 1000,
    'additional_item_cost' => 200,
    'additional_item_limit' => 2
);
$shippingMethod = new ShippingMethod();
$shippingMethod->setConfiguration($configuration);
$shippingMethod->setCalculator(DefaultCalculators::FLEXIBLE_RATE);

$shipmentItem = new ShipmentItem();

$shipment = new Shipment();
$shipment->setMethod($shippingMethod);
$shipment->addItem($shipmentItem);

$flexibleRateCalculator = new FlexibleRateCalculator();
```

```php
$perItemRateCalculator = new PerItemRateCalculator();

$calculatorRegistry = new ServiceRegistry(CalculatorInterface::class);
$calculatorRegistry->register(DefaultCalculators::FLEXIBLE_RATE,
↪$flexibleRateCalculator);
$calculatorRegistry->register(DefaultCalculators::PER_ITEM_RATE,
↪$perItemRateCalculator);

$delegatingCalculators = new DelegatingCalculator($calculatorRegistry);
$delegatingCalculators->calculate($shipment); // returns 1000

$configuration2 = array('amount' => 200);
$shippingMethod2 = new ShippingMethod();
$shippingMethod2->setConfiguration($configuration2);
$shippingMethod2->setCalculator(DefaultCalculators::PER_ITEM_RATE);

$shipment->setMethod($shippingMethod2);
$delegatingCalculators->calculate($shipment); // returns 200
```

> **Caution:** The method ->register() and ->get() used in ->calculate throw InvalidArgumentException. The method ->calculate throws UndefinedShippingMethodException when given shipment does not have a shipping method defined.

> **Hint:** You can read more about each of the available calculators in the *Calculators* chapter.

Resolvers

ShippingMethodsResolver

Sylius has flexible system for displaying the shipping methods available for given shippables (subjects which implement **ShippableInterface**), which is base on **ShippingCategory** objects and category requirements. The requirements are constant default defined in **ShippingMethodInterface**. To provide information about the number of allowed methods it use **ShippingMethodResolver**.

First you need to create a few instances of **ShippingCategory** class:

```php
<?php

use Sylius\Component\Shipping\Model\ShippingCategory;

$shippingCategory = new ShippingCategory();
$shippingCategory->setName('Regular');
$shippingCategory1 = new ShippingCategory();
$shippingCategory1->setName('Light');
```

Next you have to create a repository w which holds a few instances of **ShippingMethod**. An InMemoryRepository, which holds a collection of **ShippingMethod** objects, was used. The configuration is shown below:

```php
<?php
```

(continued from previous page)

```php
// ...
// notice:
// $categories = array($shippingCategory, $shippingCategory1);

$firstMethod = new ShippingMethod();
$firstMethod->setCategory($categories[0]);

$secondMethod = new ShippingMethod();
$secondMethod->setCategory($categories[1]);

$thirdMethod = new ShippingMethod();
$thirdMethod->setCategory($categories[1]);
// ...
```

Finally you can create a method resolver:

```php
<?php

use Sylius\Component\Shipping\Model\ShippingCategory;
use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Model\RuleInterface;
use Sylius\Component\Shipping\Checker\Registry\RuleCheckerRegistry;
use Sylius\Component\Shipping\Checker\ItemCountRuleChecker;
use Sylius\Component\Shipping\Resolver\ShippingMethodsResolver;
use Sylius\Component\Shipping\Checker\ShippingMethodEligibilityChecker;

$ruleCheckerRegistry = new RuleCheckerRegistry();
$methodEligibilityChecker = new shippingMethodEligibilityChecker(
→$ruleCheckerRegistry);

$shippingRepository = new InMemoryRepository(); //it has collection of shipping␣
→methods

$wardrobe = new Wardrobe();
$wardrobe->setShippingCategory($shippingCategory);
$wardrobe2 = new Wardrobe();
$wardrobe2->setShippingCategory($shippingCategory1);

$shipmentItem = new ShipmentItem();
$shipmentItem->setShippable($wardrobe);
$shipmentItem2 = new ShipmentItem();
$shipmentItem2->setShippable($wardrobe2);

$shipment = new Shipment();
$shipment->addItem($shipmentItem);
$shipment->addItem($shipmentItem2);

$methodResolver = new ShippingMethodsResolver($shippingRepository,
→$methodEligibilityChecker);
$methodResolver->getSupportedMethods($shipment);
```

The `->getSupportedMethods($shipment)` method return the number of methods allowed for shipment object. There are a few possibilities:

1. All shippable objects and all ShippingMethod have category *Regular*. The returned number will be 3.

2. All ShippingMethod and one shippable object have category *Regular*. Second shippable object has category

Light. The returned number will be 3.

3. Two ShippingMethod and one shippable object have category *Regular*. Second shippable object and one ShippingMethod have category *Light*. The returned number will be 3.

4. Two ShippingMethod and one shippable object have category *Regular*. Second shippable object and second ShippingMethod have category *Light*. The second Shipping category sets the category requirements as CATEGORY_REQUIREMENT_MATCH_NONE. The returned number will be 2.

5. Two ShippingMethod and all shippable objects have category *Regular*. Second ShippingMethod has category *Light*. The second Shipping category sets the category requirements as CATEGORY_REQUIREMENT_MATCH_NONE. The returned number will be 3.

6. Two ShippingMethod and one shippable object have category *Regular*. Second shippable object and second ShippingMethod have category *Light*. The second Shipping category sets the category requirements as CATEGORY_REQUIREMENT_MATCH_ALL. The returned number will be 2.

7. Two ShippingMethod have category *Regular*. All shippable object and second ShippingMethod have category *Light*. The second Shipping category sets the category requirements as CATEGORY_REQUIREMENT_MATCH_ALL. The returned number will be 1.

Note: The categoryRequirement property in **ShippingMethod** is set default to CATEGORY_REQUIREMENT_MATCH_ANY. For more detailed information about requirements please go to *Shipping method requirements*.

Calculators

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

FlatRateCalculator

FlatRateCalculator class charges a flat rate per shipment.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Calculator\FlatRateCalculator;
use Sylius\Component\Shipping\Model\ShipmentItem;

$shipmentItem = new ShipmentItem();
$shipment = new Shipment();
$shipment->addItem($shipmentItem);

$flatRateCalculator = new FlatRateCalculator();
// this configuration should be defined in shipping method allowed for shipment
$configuration = array('amount' => 1500);

$flatRateCalculator->calculate($shipment, $configuration); // returns 1500
$configuration = array('amount' => 500);
$flatRateCalculator->calculate($shipment, $configuration); // returns 500
```

FlexibleRateCalculator

FlexibleRateCalculator calculates a shipping charge, where first item has different cost that other items.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Calculator\FlexibleRateCalculator;

$shipment = new Shipment();
$shipmentItem = new ShipmentItem();
$shipmentItem2 = new ShipmentItem();
$shipmentItem3 = new ShipmentItem();
$shipmentItem4 = new ShipmentItem();

// this configuration should be defined in shipping method allowed for shipment
$configuration = array(
    'first_item_cost'      => 1000,
    'additional_item_cost' => 200,
    'additional_item_limit' => 2
);

$flexibleRateCalculator = new FlexibleRateCalculator();

$shipment->addItem($shipmentItem);
$flexibleRateCalculator->calculate($shipment, $configuration); // returns 1000

$shipment->addItem($shipmentItem2);
$shipment->addItem($shipmentItem3);
$flexibleRateCalculator->calculate($shipment, $configuration); // returns 1400

$shipment->addItem($shipmentItem4);
$flexibleRateCalculator->calculate($shipment, $configuration);
// returns 1400, because additional item limit is 3
```

PerItemRateCalculator

PerItemRateCalculator charges a flat rate per item.

```php
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Calculator\PerItemRateCalculator;

// this configuration should be defined in shipping method allowed for shipment
$configuration = array('amount' => 200);
$perItemRateCalculator = new PerItemRateCalculator();

$shipment = new Shipment();
$shipmentItem = new ShipmentItem();
$shipmentItem2 = new ShipmentItem();

$perItemRateCalculator->calculate($shipment, $configuration); // returns 0

$shipment->addItem($shipmentItem);
```

```
$perItemRateCalculator->calculate($shipment, $configuration); // returns 200

$shipment->addItem($shipmentItem2);
$perItemRateCalculator->calculate($shipment, $configuration); // returns 400
```

VolumeRateCalculator

VolumeRateCalculator charges amount rate per volume.

```
<?

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Calculator\VolumeRateCalculator;

$wardrobe = new Wardrobe();

$shipmentItem = new ShipmentItem();
$shipmentItem->setShippable($wardrobe);

$shipment = new Shipment();
$shipment->addItem($shipmentItem);

$configuration = array('amount' => 200, 'division' => 5);
// this configuration should be defined in shipping method allowed for shipment
$volumeRateCalculator = new VolumeRateCalculator();

$wardrobe->setShippingVolume(100);
$volumeRateCalculator->calculate($shipment, $configuration); // returns 4000

$wardrobe->setShippingVolume(20);
$volumeRateCalculator->calculate($shipment, $configuration); // returns  800
```

Hint: To see implementation of Wardrobe class please go to *Basic Usage*.

WeightRateCalculator

WeightRateCalculator charges amount rate per weight.

```
<?php

use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Calculator\WeightRateCalculator;

$configuration = array('fixed' => 200, 'variable' => 500, 'division' => 5);
// this configuration should be defined in shipping method allowed for shipment
$weightRateCalculator = new WeightRateCalculator();

$wardrobe = new Wardrobe();
```

```php
$shipmentItem = new ShipmentItem();
$shipmentItem->setShippable($wardrobe);

$shipment = new Shipment();
$shipment->addItem($shipmentItem);

$wardrobe->setShippingWeight(100);
$weightRateCalculator->calculate($shipment, $configuration); // returns 10200

$wardrobe->setShippingWeight(10);
$weightRateCalculator->calculate($shipment, $configuration); // returns 1200
```

Hint: To see implementation of Wardrobe class please go to *Basic Usage*.

Checkers

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

ItemCountRuleChecker

This class checks if item count exceeds (or at least is equal) the configured count. An example about how to use it is on *RuleCheckerInterface*.

Note: This checker implements the *RuleCheckerInterface*.

For more detailed information go to Sylius API ItemCountRuleChecker.

ShippingMethodEligibilityChecker

This class checks if shipping method rules are capable of shipping given subject.

```php
<?php

use Sylius\Component\Shipping\Model\Rule;
use Sylius\Component\Shipping\Model\ShippingMethod;
use Sylius\Component\Shipping\Model\ShippingCategory;
use Sylius\Component\Shipping\Model\Shipment;
use Sylius\Component\Shipping\Model\ShipmentItem;
use Sylius\Component\Shipping\Model\ShippingMethodTranslation;
use Sylius\Component\Shipping\Model\RuleInterface;
use Sylius\Component\Shipping\Checker\ItemCountRuleChecker;
use Sylius\Component\Shipping\Checker\ShippingMethodEligibilityChecker;
use Sylius\Component\Shipping\Checker\RuleCheckerInterface;
use Sylius\Component\Registry\ServiceRegistry;

$rule = new Rule();
```

```php
$rule->setConfiguration(array('count' => 0, 'equal' => true));
$rule->setType(RuleInterface::TYPE_ITEM_COUNT);

$shippingCategory = new ShippingCategory();
$shippingCategory->setName('Regular');

$hippingMethodTranslate = new ShippingMethodTranslation();
$hippingMethodTranslate->setLocale('en');
$hippingMethodTranslate->setName('First method');

$shippingMethod = new ShippingMethod();
$shippingMethod->setCategory($shippingCategory);
$shippingMethod->setCurrentLocale('en');
$shippingMethod->setFallbackLocale('en');
$shippingMethod->addTranslation($hippingMethodTranslate);

$shippingMethod->addRule($rule);

$shippable = new ShippableObject();
$shippable->setShippingCategory($shippingCategory);

$shipmentItem = new ShipmentItem();
$shipmentItem->setShippable($shippable);

$shipment = new Shipment();
$shipment->addItem($shipmentItem);

$ruleChecker = new ItemCountRuleChecker();

$ruleCheckerRegistry = new ServiceRegistry(RuleCheckerInterface::class);
$ruleCheckerRegistry->register(RuleInterface::TYPE_ITEM_COUNT, $ruleChecker);

$methodEligibilityChecker = new ShippingMethodEligibilityChecker(
→$ruleCheckerRegistry);

///returns true, because quantity of shipping item in shipment is equal as count in
→rule's configuration
$methodEligibilityChecker->isEligible($shipment, $shippingMethod);

// returns true, because the shippable object has the same category as shippingMethod
// and shipping method has default category requirement
$methodEligibilityChecker->isCategoryEligible($shipment, $shippingMethod);
```

Caution: The method ->register() throws InvalidArgumentException.

Note: This model implements the *ShippingMethodEligibilityCheckerInterface*.

For more detailed information go to Sylius API ShippingMethodEligibilityChecker.

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Shipment

Shipment object has methods to represent the events that take place during the process of shipment. Shipment has the following properties:

Property	Description
id	Unique id of the shipment
state	Reference to constant from ShipmentInterface
method	Reference to ShippingMethod
items	Reference to Collection of shipping items
tracking	Tracking code for shipment
createdAt	Creation time
updatedAt	Last update time

Note: This model implements the *ShipmentInterface*.

For more detailed information go to Sylius API Shipment.

ShipmentItem

ShipmentItem object is used for connecting a shippable object with a proper shipment. ShipmentItems have the following properties:

Property	Description
id	Unique id of the ShipmentItem
shipment	Reference to Shipment
shippable	Reference to shippable object
shippingState	Reference to constant from ShipmentInterface
createdAt	Creation time
updatedAt	Last update time

Note: This model implements the *ShipmentItemInterface*.

For more detailed information go to Sylius API ShipmentItem.

ShippingCategory

ShippingCategory object represents category which can be common for **ShippingMethod** and object which implements **ShippableInterface**. ShippingCategory has the following properties:

Property	Description
id	Unique id of the ShippingCategory
code	Unique code of the ShippingCategory
name	e.g. "Regular"
description	e.g. "Regular weight items"
createdAt	Creation time
updatedAt	Last update time

Hint: To understand relationship between **ShippingMethod** and shippable object base on **ShippingCategory** go to *Shipping method requirements*.

Note: This model implements the *ShippingCategoryInterface*.

For more detailed information go to Sylius API ShippingCategory.

ShippingMethod

ShippingMethod object represents method of shipping allowed for given shipment. It has the following properties:

Property	Description
id	Unique id of the ShippingMethod
code	Unique code of the ShippingMethod
category	e.g. "Regular"
categoryRequirement	Reference to constant from ShippingMethodInterface
enabled	Boolean flag of enablement
calculator	Reference to constant from DefaultCalculators
configuration	Extra configuration for calculator
rules	Collection of Rules
createdAt	Creation time
updatedAt	Last update time
currentTranslation	Translation chosen from translations list accordingly to current locale
currentLocale	Currently set locale
translations	Collection of translations
fallbackLocale	Locale used in case no translation is available

Note: This model implements the *ShippingMethodInterface* and uses the *Using TranslatableTrait*.

For more detailed information go to Sylius API ShippingMethod.

ShippingMethodTranslation

ShippingMethodTranslation object allows to translate the shipping method's name accordingly to the provided locales. It has the following properties:

Property	Description
id	Unique id of the ShippingMethodTranslation
name	e.g. "FedEx"
locale	Translation locale
translatable	The translatable model assigned to this translation

Note: This model implements the *ShippingMethodTranslationInterface* and extends *Implementing AbstractTranslation* class.

Form more information go to Sylius API ShippingMethodTranslation.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

RuleInterface

This interface should be implemented by class which will provide additional restriction for **ShippingMethod**.

Note: For more detailed information go to Sylius API RuleInterface.

ShipmentInterface

This interface should be implemented by class which will provide information about shipment like: state, shipping method and so on. It also has a method for shipment tracking.

Note: This interface extends the *ShippingSubjectInterface*.

For more detailed information go to Sylius API ShipmentInterface.

ShipmentItemInterface

This interface is implemented by class responsible for connecting shippable object with proper shipment. It also provides information about shipment state.

Note: This interface extends the *ShippingSubjectInterface*.

For more detailed information go to Sylius API ShipmentItemInterface.

ShippableInterface

This interface should be implemented by model representing physical object which can by stored in a shop.

Note: For more detailed information go to Sylius API ShippableInterface.

ShippingCategoryInterface

This interface should be implemented by model representing a shipping category and it is required if you want to classify shipments and connect it with right shipment method.

Note: This interface extends the *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API ShippingCategoryInterface.

ShippingMethodInterface

This interface provides default requirements for system of matching shipping methods with shipments based on **ShippingCategory** and allows to add a new restriction to a basic shipping method.

Note: This interface extends the *CodeAwareInterface*, *TimestampableInterface* and *ShippingMethodTranslationInterface*.

For more detailed information go to Sylius API ShippingMethodInterface.

ShippingMethodTranslationInterface

This interface should be implemented by model responsible for keeping translation for **ShippingMethod** name.

Note: For more detailed information go to Sylius API ShippingMethodTranslationInterface.

ShippingSubjectInterface

This interface should be implemented by any object, which needs to be evaluated by default shipping calculators and rule checkers.

Note: For more detailed information go to Sylius API ShippingSubjectInterface.

Calculator interfaces

CalculatorInterface

This interface provides basic methods for calculators. Every custom calculator should implement **CalculatorInterface** or extends class **Calculator**, which has a basic implementation of methods from this interface.

Note: For more detailed information go to Sylius API CalculatorInterface.

DelegatingCalculatorInterface

This interface should be implemented by any object, which will be responsible for delegating the calculation to a correct calculator instance.

Note: For more detailed information go to Sylius API DelegatingCalculatorInterface.

CalculatorRegistryInterface

This interface should be implemented by an object, which will keep all calculators registered inside container.

Note: For more detailed information go to Sylius API CalculatorRegistryInterface.

Checker Interfaces

RuleCheckerRegistryInterface

This interface should be implemented by an service responsible for providing an information about available rule checkers.

Note: For more detailed information go to Sylius API RuleCheckerRegistryInterface.

RuleCheckerInterface

This interface should be implemented by an object, which checks if a shipping subject meets the configured requirements.

Note: For more detailed information go to Sylius API RuleCheckerInterface.

ShippingMethodEligibilityCheckerInterface

This interface should be implemented by an object, which checks if the given shipping subject is eligible for the shipping method rules.

Note: For more detailed information go to Sylius API ShippingMethodEligibilityCheckerInterface.

Processor Interfaces

ShipmentProcessorInterface

This interface should be implemented by an object, which updates shipments and shipment items states.

Note: For more detailed information go to Sylius API ShipmentProcessorInterface.

Resolver Interfaces

ShippingMethodsResolverInterface

This interface should be used to create object, which provides information about all allowed shipping methods for given shipping subject.

Note: For more detailed information go to Sylius API ShippingMethodsResolverInterface.

State Machine

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Shipment States

Sylius itself uses a state machine system to manage all states of the business domain. This component has some sensible default states defined in **ShipmentInterface**.

All new **Shipment** instances have the state `ready` by default, which means they are prepared to be sent.

The following states are defined:

Related constant	State	Description
STATE_READY	ready	Payment received, shipment has been ready to be sent
STATE_CHECKOUT	checkout	Shipment has been created
STATE_ONHOLD	onhold	Shipment has been locked and it has been waiting to payment
STATE_PENDING	pending	Shipment has been waiting for confirmation of receiving payment
STATE_SHIPPED	shipped	Shipment has been sent to the customer
STATE_CANCELLED	cancelled	Shipment has been cancelled
STATE_RETURNED	returned	Shipment has been returned

Learn more

- *Shipments in the Sylius platform* - concept documentation

Taxation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Tax rates and tax classification for PHP applications. You can define different tax categories and match them to objects.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/taxation` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Taxation).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Tax Rate

Every tax rate has three identifiers, an ID, code and name. You can access those by calling `->getId()`, `->getCode()` and `getName()` respectively. The name and code are mutable, so you can change them by calling `->setCode('X12XW')` and `->setName('EU VAT')` on the tax rate instance.

Setting Tax Amount

A tax rate has two basic amounts - the *amount* and the *amount as percentage* (by default equal 0).

```php
<?php

use Sylius\Component\Taxation\Model\TaxRate;
use Sylius\Component\Taxation\Model\TaxCategory;
```

(continues on next page)

```php
$taxRate = new TaxRate();
$taxCategory = new TaxCategory();

$taxRate->setAmount(0.5);
$taxRate->getAmount(); // Output will be 0.5
$taxRate->getAmountAsPercentage(); // Output will be 50
```

Setting Tax Category

Every tax rate can have a tax category. You can simply set or unset it by calling ->setCategory().

```php
<?php

$taxRate->setCategory($taxCategory);
$taxRate->getCategory(); // Output will be  $taxCategory object
$taxRate->setCategory();
$taxRate->getCategory(); // Output will be null
```

Including tax rate in price

You can mark a tax rate as included in price by calling setIncludedInPrice(true) (false by default). To check if tax rate is included in price call isIncludedInPrice().

Hint: You can read how this property influences on the tax calculation in chapter *Default Calculator*.

Setting calculator

To set type of calculator for your tax rate object call setCalculator('nameOfCalculator'). Notice that nameOfCalculator should be the same as name of your calculator object.

Hint: To understand meaning of this property go to *Delegating Calculator*.

Tax Category

Every tax category has three identifiers, an ID, code and name. You can access those by calling ->getId(), ->getCode() and getName() respectively. The code and name are mutable, so you can change them by calling ->setCode('X12X') and ->setName('Clothing') on the tax category instance.

Tax Rate Management

The collection of tax rates (Implementing the Doctrine\Common\Collections\Collection interface) can be obtained using the getRates() method. To add or remove tax rates, you can use the addRate() and removeRate() methods.

```php
<?php

use Sylius\Component\Taxation\Model\TaxRate;
use Sylius\Component\Taxation\Model\TaxCategory;

$taxCategory = new TaxCategory();

$taxRate1 = new TaxRate();
$taxRate1->setName('taxRate1');

$taxRate2 = new TaxRate();
$taxRate2->setName('taxRate2');

$taxCategory->addRate($taxRate1);
$taxCategory->addRate($taxRate2);
$taxCategory->getRates();
//returns a collection of objects that implement the TaxRateInterface
$taxCategory->removeRate($taxRate1);
$taxCategory->hasRate($taxRate2); // returns true
$taxCategory->getRates(); // returns collection with one element
```

Calculators

Default Calculator

Default Calculator gives you the ability to calculate the tax amount for given base amount and tax rate.

```php
<?php

use Sylius\Component\Taxation\Model\TaxRate;
use Sylius\Component\Taxation\Calculator\DefaultCalculator;

$taxRate = new TaxRate();
$taxRate->setAmount(0.2);
$basicPrice = 100;
$defaultCalculator = new DefaultCalculator();
$defaultCalculator->calculate($basicPrice, $taxRate); //return 20
$taxRate->setIncludedInPrice(true);
$defaultCalculator->calculate($basicPrice, $taxRate);
// return 17, because the tax is now included in price
```

Delegating Calculator

Delegating Calculator gives you the ability to delegate the calculation of amount of tax to a correct calculator instance based on a type defined in an instance of **TaxRate** class.

```php
<?php

use Sylius\Component\Taxation\Model\TaxRate;
use Sylius\Component\Taxation\Calculator\DefaultCalculator;
use Sylius\Component\Registry\ServiceRegistry;
use Sylius\Component\Taxation\Calculator\DelegatingCalculator;
use Sylius\Component\Taxation\Calculator\CalculatorInterface;
```

(continues on next page)

```php
$taxRate = new TaxRate();
$taxRate->setAmount(0.2);
$base = 100; //set base price to 100
$defaultCalculator = new DefaultCalculator();

$serviceRegistry =
new ServiceRegistry(CalculatorInterface::class);
$serviceRegistry->register('default', $defaultCalculator);

$delegatingCalculator = new DelegatingCalculator($serviceRegistry);
$taxRate->setCalculator('default');
$delegatingCalculator->calculate($base, $taxRate); // returns 20
```

Tax Rate Resolver

TaxRateResolver gives you ability to get information about tax rate for given taxable object and specific criteria. The criteria describes tax rate object.

```php
<?php

use Sylius\Component\Taxation\Resolver\TaxRateResolver;
use Sylius\Component\Taxation\Model\TaxCategory;

$taxRepository = new InMemoryTaxRepository(); // class which implements
↪RepositoryInterface
$taxRateResolver= new TaxRateResolver($taxRepository);

$taxCategory = new TaxCategory();
$taxCategory->setName('TaxableGoods');

$taxableObject = new TaxableObject(); // class which implements TaxableInterface
$taxableObject->setTaxCategory($taxCategory);

$criteria = array('name' => 'EU VAT');
$taxRateResolver->resolve($taxableObject, $criteria);
// returns instance of class TaxRate, which has name 'EU VAT' and category
↪'TaxableGoods'
```

Models

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

TaxRate

Tax rate model holds the configuration for particular tax rate.

Property	Description
id	Unique id of the tax rate
code	Unique code of the tax rate
category	Tax rate category
name	Name of the rate
amount	Amount as float (for example 0,23)
includedInPrice	Is the tax included in price?
calculator	Type of calculator
createdAt	Date when the rate was created
updatedAt	Date of the last tax rate update

Note: This model implements `TaxRateInterface`.

TaxCategory

Tax category model holds the configuration for particular tax category.

Property	Description
id	Unique id of the tax category
code	Unique code of the tax category
name	Name of the category
description	Description of tax category
rates	Collection of tax rates belonging to this tax category
createdAt	Date when the category was created
updatedAt	Date of the last tax category update

Note: This model implements `TaxCategoryInterface`.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Model Interfaces

Taxable Interface

To create taxable object which has specific type of tax category, the object class needs to implement **TaxableInterface**.

Note: For more detailed information go to Sylius API Taxable Interface.

Tax Category Interface

To create object which provides information about tax category, the object class needs to implement **TaxCategoryInterface**.

Note: This interface extends *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API Tax Category Interface.

Tax Rate Interface

To create object which provides information about tax rate, the object class needs to implement **TaxCategoryInterface**.

Note: This interface extends *CodeAwareInterface* and *TimestampableInterface*.

For more detailed information go to Sylius API Tax Rate Interface.

Calculator Interfaces

CalculatorInterface

To make the calculator able to calculate the tax amount for given base amount and tax rate, the calculator class needs implement the **CalculatorInterface**.

Note: For more detailed information about the interfaces go to Sylius API Calculator Interface.

Resolver Interfaces

TaxRateResolverInterface

To create class which provides information about tax rate for given taxable object and specific criteria, the class needs to implement **TaxRateResolverInterface**. The criteria describes tax rate object.

Note: For more detailed information about the interfaces go to Sylius API Tax Rate Resolver Interface.

Learn more

- *Taxation in the Sylius platform* - concept documentation

Taxonomy

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Basic taxonomies library for any PHP application. Taxonomies work similarly to the distinction of species in the fauna and flora and their aim is to help the store owner manage products.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/taxonomy` on Packagist);
- Use the official Git repository (https://github.com/Sylius/Taxonomy).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Basic Usage

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

```php
<?php

use Sylius\Component\Taxonomy\Model\Taxon;
use Sylius\Component\Taxonomy\Model\Taxonomy;

// Let's assume we want to begin creating new taxonomy in our system
// therefore we think of a new taxon that will be a root for us.
$taxon = new Taxon();

// And later on we create a taxonomy with our taxon as a root.
$taxonomy = new Taxonomy($taxon);

// Before we can start using the newly created taxonomy, we have to define its
↪locales.
$taxonomy->setFallbackLocale('en');
$taxonomy->setCurrentLocale('en');
$taxonomy->setName('Root');

$taxon->getName(); //will return 'Root'
```

Models

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Taxonomy is a list constructed from individual Taxons. Taxonomy is a special case of Taxon itself (it has no parent). All taxons can have many child taxons, you can define as many of them as you need.

Good examples of taxonomies are "Categories" and "Brands". Below you can see exemplary trees.

```
| Categories
|\__T-Shirts
|   |\__Men
|    \__Women
|\__Stickers
|\__Mugs
 \__Books

| Brands
|\__SuperTees
|\__Stickypicky
|\__Mugland
 \__Bookmania
```

Taxon

Property	Description
id	Unique id of the taxon
code	Unique code of the taxon
name	Name of the taxon taken form the `TaxonTranslation`
slug	Urlized name taken from the `TaxonTranslation`
description	Description of taxon taken from the `TaxonTranslation`
parent	Parent taxon
children	Sub taxons
left	Location within taxonomy
right	Location within taxonomy
level	How deep it is in the tree
position	Position of the taxon on its taxonomy

> **Note:** This model implements the *TaxonInterface*. You will find more information about this model in Sylius API Taxon.

TaxonTranslation

This model stores translations for the **Taxon** instances.

Property	Description
id	Unique id of the taxon translation
name	Name of the taxon
slug	Urlized name
description	Description of taxon

Note: This model implements the *TaxonTranslationInterface*. You will find more information about this model in Sylius API TaxonTranslation.

Interfaces

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Models Interfaces

TaxonInterface

The **TaxonInterface** gives an object an ability to have Taxons assigned as children.

Note: This interface extends the *CodeAwareInterface*, *TranslatableInterface* and the *TaxonTranslationInterface*.

You will find more information about that interface in Sylius API TaxonInterface.

TaxonsAwareInterface

The **TaxonsAwareInterface** should be implemented by models that can be classified with taxons.

Note: You will find more information about that interface in Sylius API TaxonsAwareInterface.

TaxonTranslationInterface

This interface should be implemented by models that will store the **Taxon** translation data.

Note: You will find more information about that interface in Sylius API TaxonTranslationInterface.

Services Interfaces

TaxonRepositoryInterface

In order to have a possibility to get Taxons as a list you should create a repository class, that implements this interface.

Note: You will find more information about that interface in Sylius API TaxonRepositoryInterface.

Learn more

- *Taxons in the Sylius platform* - concept documentation

User

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Users management implementation in PHP.

Installation

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

You can install the component in 2 different ways:

- *Install it via Composer* (`sylius/user` on Packagist);
- Use the official Git repository (https://github.com/Sylius/User).

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Sylius component.

Models

Customer

The customer is represented as a **Customer** instance. It should have everything concerning personal data and as default has the following properties:

Property	Description	Type
id	Unique id of the customer	integer
email	Customer's email	string
emailCanonical	Normalized representation of an email (lowercase)	string
firstName	Customer's first name	string
lastName	Customer's last name	string
birthday	Customer's birthday	DateTime
gender	Customer's gender	string
user	Corresponding user object	UserInterface
group	Customer's groups	Collection
createdAt	Date of creation	DateTime
updatedAt	Date of update	DateTime

Note: This model implements `CustomerInterface`

User

The registered user is represented as an **User** instance. It should have everything concerning application user preferences and a corresponding **Customer** instance. As default has the following properties:

Property	Description	Type
id	Unique id of the user	integer
customer	Customer which is associated to this user (required)	CustomerInterface
username	User's username	string
usernameCanonical	Normalized representation of a username (lowercase)	string
enabled	Indicates whether user is enabled	bool
salt	Additional input to a function that hashes a password	string
password	Encrypted password, must be persisted	string
plainPassword	Password before encryption, must not be persisted	string
lastLogin	Last login date	DateTime
confirmationToken	Random string used to verify user	string
passwordRequestedAt	Date of password request	DateTime
locked	Indicates whether user is locked	bool
expiresAt	Date when user account will expire	DateTime
credentialExpiresAt	Date when user account credentials will expire	DateTime
roles	Security roles of a user	array
oauthAccounts	Associated OAuth accounts	Collection
createdAt	Date of creation	DateTime
updatedAt	Date of update	DateTime

Note: This model implements `UserInterface`

CustomerGroup

The customer group is represented as a **CustomerGroup** instance. It can be used to classify customers. As default has the following properties:

Property	Description	Type
id	Unique id of the group	integer
name	Group name	string

Note: This model implements `CustomerGroupInterface`

UserOAuth

The user OAuth account is represented as an **UserOAuth** instance. It has all data concerning OAuth account and as default has the following properties:

Property	Description	Type
id	Unique id of the customer	integer
provider	OAuth provider name	string
identifier	OAuth identifier	string
accessToken	OAuth access token	string
user	Corresponding user account	UserInterface

Note: This model implements `UserOAuthInterface`

Basic Usage

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

Canonicalization

Danger: We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In order to be able to query or sort by some string, we should normalize it. The most common use case for that is canonical email or username. We can then allow for case insensitive users identification by email or username.

Canonicalizer

User component offers simple canonicalizer which converts given string to lowercase letters. Example usage:

```php
// File example: src/script.php
<?php

// update this to the path to the "vendor/"
```

```php
// directory, relative to this file
require_once __DIR__.'/../vendor/autoload.php';

use Sylius\Component\User\Model\User;
use Sylius\Component\Canonicalizer\Canonicalizer;

$canonicalizer = new Canonicalizer();

$user = new User();
$user->setEmail('MyEmail@eXample.Com');

$canonicalEmail = $canonicalizer->canonicalize($user->getEmail());
$user->setEmailCanonical($canonicalEmail);

$user->getEmail() // returns 'MyEmail@eXample.Com'
$user->getEmailCanonical() // returns 'myemail@example.com'
```

Updating password

> **Danger:** We're sorry but **this documentation section is outdated**. Please have that in mind when trying to use it. You can help us making documentation up to date via Sylius Github. Thank you!

In order to store user's password safely you need to encode it and get rid of the plain password.

PasswordUpdater

User component offers simple password updater and encoder. All you need to do is set the plain password on *User* entity and use *updatePassword* method on *PasswordUpdater*. The plain password will be removed and the encoded password will be set on *User* entity. Now you can safely store the encoded password. Example usage:

```php
// File example: src/script.php
<?php

// update this to the path to the "vendor/"
// directory, relative to this file
require_once __DIR__.'/../vendor/autoload.php';

use Sylius\Component\User\Model\User;
use Sylius\Component\User\Security\PasswordUpdater;
use Sylius\Component\User\Security\UserPbkdf2PasswordEncoder;

$user = new User();
$user->setPlainPassword('secretPassword');

$user->getPlainPassword(); // returns 'secretPassword'
$user->getPassword(); // returns null

// after you set user's password you need to encode it and get rid of unsafe plain␣
↪text
$passwordUpdater = new PasswordUpdater(new UserPbkdf2PasswordEncoder());
$passwordUpdater->updatePassword($user);
```

```
// the plain password no longer exist
$user->getPlainPassword(); // returns null
// encoded password can be safely stored
$user->getPassword(); //returns 'notPredictableBecauseOfSaltHashedPassword'
```

Note: The password encoder takes user's salt (random, autogenerated string in the *User* constructor) as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Learn more

- *Customers & Users in the Sylius platform* - concept documentation
- *Components General Guide*
- *Addressing*
- *Attribute*
- *Channel*
- *Currency*
- *Grid*
- *Inventory*
- *Locale*
- *Mailer*
- *Order*
- *Payment*
- *Product*
- *Promotion*
- *Registry*
- *Resource*
- *Shipping*
- *Taxation*
- *Taxonomy*
- *User*
- *Sylius Components Documentation*
- *Sylius Bundles Documentation*
- *Sylius Components Documentation*
- *Sylius Bundles Documentation*

Index