



Formation PHP -Symfony

D07 - Advanced Symfony Concepts

Rares Serban rares.serban@pitechplus.com
42 Staff pedago@staff.42.fr

Summary: Following [42](#) formation course, you will learn about more advanced Symfony concepts which are widely used by Symfony developers, like translations, custom bundle configuration and unit testing.

Contents

I	Foreword	2
II	General Rules	3
III	Day-specific rules	4
IV	Exercise 00	5
V	Exercise 01	6
VI	Exercise 02	7
VII	Exercise 03	9
VIII	Exercise 04	10

Chapter I

Foreword

So far you have been introduced to Symfony and some of its concepts, you learned about authentication and authorization and about SQL and ORM, but there are still a lot of concepts to cover in the Symfony framework. Today we are going to learn about more advanced topics that are used on a daily basis by the majority of Symfony developers.

Chapter II

General Rules

- This subject is the one and only trustable source. Don't trust any rumor.
- This subject can be updated up to one hour before the turn-in deadline.
- The assignments in a subject must be done in the given order. Later assignments won't be rated unless all the previous ones are perfectly executed.
- Be careful about the access rights of your files and folders.
- You must follow the **turn-in process** for each assignment. The url of your **GIT** repository for this day is available on your intranet.
- Your assignments will be evaluated by your Piscine peers.
- In addition to your peers evaluation, a program called the "Moulinette" should also evaluate your assignments. Fully automated, The Moulinette is tough and unforgiving in its evaluations. As a consequence, it is impossible to bargain your grade with it. Uphold the highest level of rigor to avoid unpleasant surprises.
- All shell assignments must run using `/bin/sh`.
- You must not leave in your turn-in repository any file other than the ones explicitly requested By the assignments.
- You have a question? Ask your left neighbor. Otherwise, try your luck with your right neighbor.
- Every technical answer you might need is available in the **mans** or on the Internet.
- Remember to use the Piscine forum of your intranet and also Slack!
- You must read the examples thoroughly. They can reveal requirements that are not obvious in the assignment's description.
- By Thor, by Odin! Use your brain!!!


Chapter III

Day-specific rules

- For this day, your repository must contain just one working Symfony application.
- Best practices of the Symfony framework should be respected.

Chapter IV


Exercise 00

	Exercise
Exercise 00: Base Custom Bundle	
Turn-in directory : <i>ex/</i>	
Files to turn in : Files and folders from your application	
Allowed functions : All methods	
Notes : n/a	

For this exercise you have to set up a new Symfony application and a custom bundle which will be extended by the following exercises. The bundle should be called **D07Bundle** and no other bundles should exist in your application. The bundle folder should not contain any other folders besides an empty controller folder for now. Also, the **app/Resources/views** directory should currently be empty. You should get a 404 error when trying to access your website on the default path.

Chapter V

Exercise 01

	Exercise
Exercise 01: Bundle Configuration	
Turn-in directory : <i>ex/</i>	
Files to turn in : Files and folders from your application	
Allowed functions : All methods	
Notes : n/a	

For this exercise you will start adding more advanced functionality to your bundle. You may be familiar with the Symfony configuration system and what bundles are. Each bundle can have it's own defined configuration. First, you need to create a configuration file for this bundle. Your bundle configuration should have the root key **d07** and the following 2 subkeys:

- **number** - mandatory, should be an integer
- **enable** - optional, should be a boolean, defaults to true


After you have set up the configuration file for the bundle, add the mandatory keys of your bundle configuration to your **app/config/config.yml** file.

To test that your configuration works properly, create a new controller class name **Ex01Controller** with an action called **ex01Action** with the route **/ex01** which should just return a plain response containing the value of the **number** from your bundle configuration.

Hint: *The controller has a nice helper function **getParameter** to get any parameter from the container, but make sure your bundle configuration is available in the container. An Extension Class might help?*

Chapter VI

Exercise 02

	Exercise
Exercise 02: Translations	
Turn-in directory : <i>ex/</i>	
Files to turn in : Files and folders from your application	
Allowed functions : All methods	
Notes : n/a	

It is now time to make your application display content in multiple languages, **en** and **fr**. Set the default locale for your application and enable translations. Create two new files in the **app/Resources/translations** directory. They should be called **messages.en.yml** and **messages.fr.yml**.

Then create a custom controller **Ex02Controller** with a custom action **translations-Action** which should take an optional parameter **count** which defaults to 0 and can only have the values between 0 and 9. The route for this action should be **/_{_locale}/ex02/{count}**. Depending on the **__locale** parameter, your site will be displayed in either english or french. Now create a template called **ex02.html.twig** which should be returned by your action and should have as a parameter the number found in your bundle configuration from the previous exercise and the **count** parameter.

The template should contain the following text depending on the language and the text should be retrieved from the appropriate translations file using the keys **ex02.number** and **ex02.count**.

English

- *The config number is %number%* - the correct parameter should be passed to this translation
- *none/one/number %count%* - depending on the parameter value, a different translation should be used

French


- *Le numéro de configuration est %number%* - the correct parameter should be passed to this translation

- *aucun/un/nombre %count%* - depending on the parameter value, a different translation should be used

Hint: *Use the available Twig filters for translations and the **@Route** annotation for parameter validation.*

Chapter VII

Exercise 03

	Exercise
Exercise 03: Twig Extension & Dependency Injection	
Turn-in directory : <i>ex/</i>	
Files to turn in : Files and folders from your application	
Allowed functions : All methods	
Notes : n/a	


I bet you were wandering how Twig works in more detail and how you can add your own custom functions and filters to use in your templates. For this you can create a Twig Extension. The class for this extension should be **src/D07Bundle/Twig/Ex03Extension.php**. Create a twig filter and a twig function. The filter should be called **uppercaseWords** and if applied on a string it should uppercase the first letter of each word from that string. The function should be called **countNumbers** and will return the number of digits in a string.

These functions should not be created in the extension class, but in a separate service class **src/D07Bundle/Service/Ex03Service**. This service class will then be injected in the twig extension. Then create a controller called **Ex03Controller** with an action **extensionAction** and route **/ex03** and a template **ex03.html.twig**. The template should use both the filter and the function on whatever strings you want that come from translations and display the results.

Hint: *Services can be defined in the **app/config/services.yml** file.*

Chapter VIII

Exercise 04

	Exercise
Exercise 04: Unit Testing	
Turn-in directory : <i>ex/</i>	
Files to turn in : Files and folders from your application	
Allowed functions : All methods	
Notes : n/a	

It is time to test the service created in the previous exercise using **PHPUnit** and unit testing. Create a test class for your service which should follow the general Symfony 2.8 tests naming convention and test the two functions in that service, **uppercaseWords** and **countNumbers**.

Write at least 3 different asserts for each function and make sure you respect the guidelines given in the previous exercise.

All the tests you write should pass!