Formation PHP Symfony - 3

Advanced Symfony

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.

Version: 1

Contents

Chapter I

Foreword

So far you have been introduced to Symfony and some of it's 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

- Your project must be realized in a virtual machine.

- Your virtual machine must have all the necessary software to complete your project. These softwares must be configured and installed.

- You can choose the operating system to use for your virtual machine.

- You must be able to use your virtual machine from a cluster computer.

- You must use a shared folder between your virtual machine and your host machine.

- During your evaluations you will use this folder to share with your repository.

- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.

- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.

- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

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.

If no other explicit information is displayed, you must assume the following versions of languages :

- `PHP - Symfony LTS`

- `HTML 5`

- `CSS 3`

Chapter IV

Exercise 00

	Exercise
	Exercise 00: Base Custom Bundle
Turn-in directory : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

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 **src/D07Bundle/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 : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

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 **config/packages/bundle.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 : *ex/*
Files to turn in : `Files and folders from your application`
Allowed functions : `All methods`

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 the new files in your application 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 : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

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 : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

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 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!

Chapter IX

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

> The evaluation process will happen on the computer of the evaluated group.