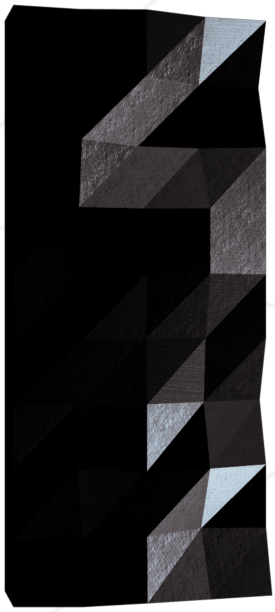

Formation PHP - Symfony

D04 - Getting started with Symfony

Summary: Following this day you will get to know basic concepts in Symfony framework.

Contents

Chapter I

Foreword

The void type, in several programming languages derived from C and Algol68, is the type for the result of a function that returns normally, but does not provide a result value to its caller. Usually such functions are called for their side effects, such as performing some task or writing to their output parameters. The usage of the void type in such context is comparable to procedures in Pascal and syntactic constructs which define subroutines in Visual Basic. It is also similar to the unit type used in functional programming languages and type theory.

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

- Every excercise should be resolved in a different bundle.
- Every controller has to go under Controller folder inside the bundle.
- File names containing controller classes should be suffixed with `Controller` and should contain a class with the same name.
- The content of every page should be in twig files and every twig file should have the file extension `.html.twig`
- The server used for this day is the one integrated in Symfony. It should be started and stoped using Symfony console commands.
- Only explicit request URLs should render a page without error. The not configured URLs should generate a 404 error.
- The requested URLs should work with and without trailing slash. E.g: both `/ex00` and `/ex00/` must work

Chapter IV

Exercise 00

	Exercise 00
Exercise 00: First page	
Turn-in directory : *ex*00/	
Files to turn in : `All the files of the application`	
Allowed functions : `All the functionalities of Symfony`	

In this excercise you have to create a simple page in a symfony application.

First, create a Symfony project using composer :

```
composer create-project symfony/framework-standard-edition d04 "^2.8"
```

Create a bundle `E00Bundle` and register it in the AppKernel file.

`You can create the bundle using the built-in commands from Symfony.`

For defining the routes you'll have to use annotations in the controller file and you should register the routes inside the `app/config/routing.yml` file.

For this excercise you have to create a page that is viewable on the following URL : `/e00/firstpage`. On this page you should display only the following string : `"Hello world!"`.

In this excercise you should not use any templates and you should have only the controller and you should use `Response` object from HttpFoundation component.

Chapter V

Exercise 01

	Exercise 01
Exercise 01: Multiple pages	
Turn-in directory : $ex01/$	
Files to turn in : `All the files of the application`	
Allowed functions : `All the functionalities of Symfony`	

Using the project already created in the first exercise create a new bundle `E01Bundle` and register it in the AppKernel file.

You'll have to create an application that will display cheatsheet files from the following repository **https://github.com/davidpv/symfony2cheatsheet**. You'll have to integrate only `.html.twig` files inside the application.

The application should contain a main page on `/e01` where you should list all the links to the subpages which should be accesible through the following URL structure : `/e01/category`. So for example : `/e01/cache` should display the contents of **https://github.com/dav**

The list of categories that should be accesible is : controller, routing, templating, doctrine, testing, validation, forms, security, cache, translations, services.

In case a url with a wrong category is accessed the main page should be displayed instead. So accessing `/e01/wrongcategory` should display the main page that contains a list of link to the category pages.

You can define the list of categories in an array inside the controller or you can use set parameters inside the `Resources/config/services.yml` file of the Bundle.

In order to make the pages render correct HTML you have to create a `base.html.twig` file that will contain the **<html>** and **<body>** tags of the page and all category templates should extend this template and overwrite the block `content` that should be also defined in the base template.

Chapter VI

Exercise 02

	Exercise 02
Exercise 02: First form	
Turn-in directory : $ex02/$	
Files to turn in : `All the files of the application`	
Allowed functions : `All the functionalities of Symfony`	

Using the project already created in the first exercise create a new bundle `E02Bundle` and register it in the AppKernel file.

In this exercise you'll have on the URL `/e02` a form which will have two inputs : a text field "Message" containing a message and a dropdown "Include timestamp" with two selectable values : Yes and No. These informations will have to be written in a file on the disk using the following business logic : if "Yes" is selected from the dropdown you have to write both the message and the timestamp of the message on a line in the file; if "No" is selected then you have to write only the message in the file.

You have to add a server-side validation on the Message field to not be blank. Be careful that the validation should be added on the server side, the HTML5 validation message is not enough.

The name of the file should be configurable inside `app/config/parameters.yml` and should be created in the project root folder, next to composer.lock and composer.json files. If the file does not exist, the application should not fail, but it should create the file in the specified location and with the specified name.

After if the form is submitted several times the content of the file "notes.txt" should be updated with the information from the form on a new line.

And also after the form is submitted the page should refresh and the form should remember the submitted information and the last line added in the file should be displayed under the form.

You should not hardcode the form in HTML, but you should use the Form component

from Symfony and you should use the built-in types from symfony.

Chapter VII

Exercise 03

	Exercise 03
Exercise 03: Fifthy shades of colors	
Turn-in directory : *ex*03/	
Files to turn in : `All the files of the application`	
Allowed functions : `All the functionalities of Symfony`	

Using the project already created in the first exercise create a new bundle `E03Bundle` and register it in the AppKernel file.

In this exercise you'll have to create a page that will display a number of shades for the following colours : *black*, *red*, *blue*, *green*. The number of the shades should be configurable from **app/config/parameters.yml** file in the parameter : `e03.number_of_colors`.

The page should contain also a header for the table with the colors for which the shades are displayed.

Table cells should have the following characteristics :

Height : 40 pixels.

Width : 80 pixels.

Background color : a shade of the color corresponding to the column.

The table should have the number of lines specified in the **app/config/parameters.yml** file in the parameter : **e03.number_of_colors**.

You are not allowed to hardcode the values in the HTML, you have to create them dynamically and send them to the Twig template as parameter from the controller.