Formation PHP - Symfony

D09 - One Page Apps using Symfony

Summary: Following 42 formation course, you will learn how you can create a one page app using Symfony, how to make Ajax calls from a browser and how to use Websockets for client-server communication.

Contents

Chapter I

Foreword

Today we are going to learn how we can make a simple one page application using the Symfony framework.

If you do not know what a one page application is, it is basically a website that does not require a page to be reloaded by the browser, everything on the page is dynamically updated using Ajax calls from javascript.

We are also going to take a look at Websockets, which allows two way client and server communication over a TCP connection.

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 <u>must not</u> 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.

- Other third party bundles or libraries CAN be used.

- Other actions CAN be created in the provided controllers, but no other controllers should be created.

- It is recommended to use services to organize your code.

Chapter IV

Exercice 00

	Exercise
	Exercice 00: Base Bundle & Basic entity
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 **D09Bundle** and no other bundles should exist in your application. The bundle folder should not contain any unnecessary folders. Also, the **app/Resources/views** directory should currently be empty.

You will have to create a new entity in the bundle called **Post**. It should have at least the following fields. Their types should be self-explanatory.

- id

- title

- content

- created

The **created** field should take the value of the date the object was created at using Doctrine Events.

Hint: *The command **app/console doctrine:generate:entity** can be used to generate entities.*

Chapter V

Exercice 01

	Exercise
	Exercice 01: Post form & Security

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

Only users logged in to the application should be able to create new posts.For this you need to set up security and create some simple users.Create a controller **UserController** with a **loginAction** for the route **/login**. This action should return a template **login.html.twig** which should contain a login form.The form will need to be submitted via Ajax.

Also create a controller **PostController** with an action **defaultAction** for the default path which should return a template called **index.html.twig**. The template should either display the login form if the user is not logged in or a post form. The post form should contain the **title** and **content** fields and should be created using a post type class.

The login form needs to be submitted via Ajax and if the credentials are correct, instead of the login form the post form will need to be displayed. If the credentials are not correct, a browser alert should be shown.

The post form should also be submitted via Ajax and display a confirmation message if the post was successfully added.

Hint: *Symfony provides a base JsonResponse class, authentication handlers and ability to make subrequests from templates.Also make sure to secure your controller actions!*

Chapter VI

Exercice 02

	Exercise
	Exercice 02: Post List
Turn-in directory : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

Create a basic list below the form on the main page. The list should display all the posts created and their title and creation date.The list can be viewed even if a user is not authenticated.

Now modify the submitting of the form to also update this list and add newly created posts to the end of it after the Ajax request is finished.

Also implement validation for the form to make sure that the title of each post is unique. If you try to add a post with a title that already exists, you should get a browser alert with an error message.

Hint: *It is highly recommended to use translations for the form field labels and error messages. And remember, the browser page should never reload!*

Chapter VII

Exercice 03

	Exercise
	Exercice 03: Post Details & Deletion
Turn-in directory : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

Create an action in **PostController** called **viewAction** with the route **/view/{id}** which should be called using Ajax when clicking on the title of a post and return all of the post details.The details should then be displayed above the form.

Create another action in the same controller called **deleteAction** with the route **/delete/{id}** which when called from Ajax should delete the post. Only logged in users should be able to delete a post. The post details should contain a delete button which should show a confirmation dialog and then delete the post.

It should also update the posts list and remove the deleted post from it.

Chapter VIII

Exercice 04

	Exercise
	Exercice 04: Websockets!
Turn-in directory : *ex/*	
Files to turn in : `Files and folders from your application`	
Allowed functions : `All methods`	

It is now time to learn about Websockets.You probably have not heard much about them, but now is your chance to learn more!

You will first set up the Websocket server using Symfony on port 8080. The server should be started using a custom Symfony command:

app/console websocket:server

On the client side, you will have to create a new Websocket using Javascript. Now, each time a new post is added in any browser window, you will have to update the posts list of all the clients connected to the Websocket server to display the new post.

Make sure to also modify the delete form to remove the post from all the browser windows.

Hint: *This exercise can be solved in multiple ways. A variety of libraries can be found online that will help you with implementing mostly all of this functionality.*