Symfony Framework

Santosh Kalwar, 23.04.2024

Symfony Topics

General Discussions

Reflection Quiz 1

Routes, Controllers & Responses
- Creating the Controller
- Creating the Route
- Returning a Response

Hello World, Symfony App : Our first Symfony app
- Walk-through

ClassRoom Practice: **Temperature Conversion App**

Twig: Intro

Name Variation App: Our second Symfony app
- Walk-through
- Structure of the project
- Reflection

Structure of Symfony Project

The public/ Directory

The first is `public/`... and this is simple: it's the document root. In other words, if you need a file to be publicly accessible - like an image file or a CSS file - it needs to live inside `public/`.

Right now, this holds exactly one file: `index.php`, which is called the "front controller".

That's a fancy word that means that - no matter what URL the user goes to - *this* is the script that's always executed first. Its job is to boot up Symfony and run our app.

Structure of Symfony Project

config/ & src/

The `config/` directory holds... config files. And `src/` holds 100% of your PHP classes. We will spend 95% of our time inside the `src/` directory.

composer.json & vendor/

Okay... so where is "Symfony"? Our project started with a `composer.json` file. This lists all of the third party libraries that our app needs. The "symfony new" command that we ran secretly used "Composer" - that's PHP's package manager - to install these libraries... which is really just a way of saying that Composer downloaded these libraries into the `vendor/` directory.

`vendor/` is yet another directory that... we don't need to worry about!

Structure of Symfony Project

bin/ and var/

So what's left? Well, `bin/` holds exactly one file... and will always hold just this one file.

And the `var/` directory holds cache and log files. Those files are important... but *we* will never need to look at or think about that stuff.

We're going to live pretty much entirely inside of the `config/` and `src/` directories.

Routes, Controllers & Responses

Routes & Controllers

Ok: *every* web framework in *any* language has the same job: to help us create pages, whether those are HTML pages, JSON API responses or ASCII art. And pretty much every framework does this in the same way: via a route & controller system.

The route defines the URL for the page and points to a controller. The controller is a PHP function that builds that page.

So **route + controller** = page

Remember MVC? Here you can consider, View = Routes + Browser

Creating the Controller

Let's create the controller function

In Symfony, the controller function is always a *method* inside of a PHP class.

in the `src/Controller/` directory, create a new PHP class. Let's call it `HelloWorldController`, but the name could be anything.

Rules: the namespace of a class *must* match the directory structure... starting with `App`. You can imagine that the `App\` namespace points to the `src/` directory. Then, if you put a file in a `Controller/` sub-directory, it needs a `Controller` part in its namespace.

The *other* rule is that the ***name* of a file** must match the **class name** inside of it, plus `.php`.

Note: Controllers actually *do* need to live in `src/Controller/`, unless you change some config. Most PHP classes can live anywhere in `src/`.

Creating the Route

Let's create a route, which defines the *URL* to our new page and *points* to this controller.

There are a few ways to create routes in Symfony, but almost ***everyone uses annotation attributes***.

```php
use Symfony\Component\Routing\Annotation\Route;


#[Route('/')]
```

```yaml
index:
  path: /home
  controller: App\Controller\HelloWorldController::index
```

Hello Symfony, Our first Symfony app

Creating a new Symfony project with Composer

symfony new helloSymfony

cd helloSymfony

Create a controller inside **src/Controller** e.g. HelloController.php

```
class HelloController extends AbstractController
{
    #[Route("/hello", name: "hello_world")]
    public function index(): Response
    {
        return new Response("Hello World!");
    }
}
```

Visit the browser with url and see your first Hello Symfony app: https://127.0.0.1:8000/hello

Returning a Response

Let's create a route, which defines the *URL* to our new page and *points* to this controller.

The *only* thing that Symfony cares about is that your controller returns a `Response` object.

Check it out: type `return` and then start typing `Response`. HTTP foundation is one of those Symfony libraries... and it gives us nice classes for things like the Request, Response and Session.

Hello Symfony, Our first Symfony app

Change a response to “Pink Floyd --- Another Brick In the wall”

```
return new Response("Pink Floyd --- Another Brick In the wall");
```

Show the above response in the main page e.g. https://127.0.0.1:8000/

Write following command in the terminal to debug/check routes

php bin/console debug:router

Classroom Practice

1. In previous, Hello Symfony app - let’s say you want to create and add an image

2. Create a images folder, where? You guessed it right under : pubic/images/

3. Run following command shown below from your project folder:

php -r "copy('https://images.pexels.com/photos/211122/pexels-photo-211122.jpeg', 'public/images/working-on-symfony.jpeg');"

5. Browse to https://127.0.0.1:8000/images/working-on-symfony.jpeg

Classroom Practice: Temperature Conversion App

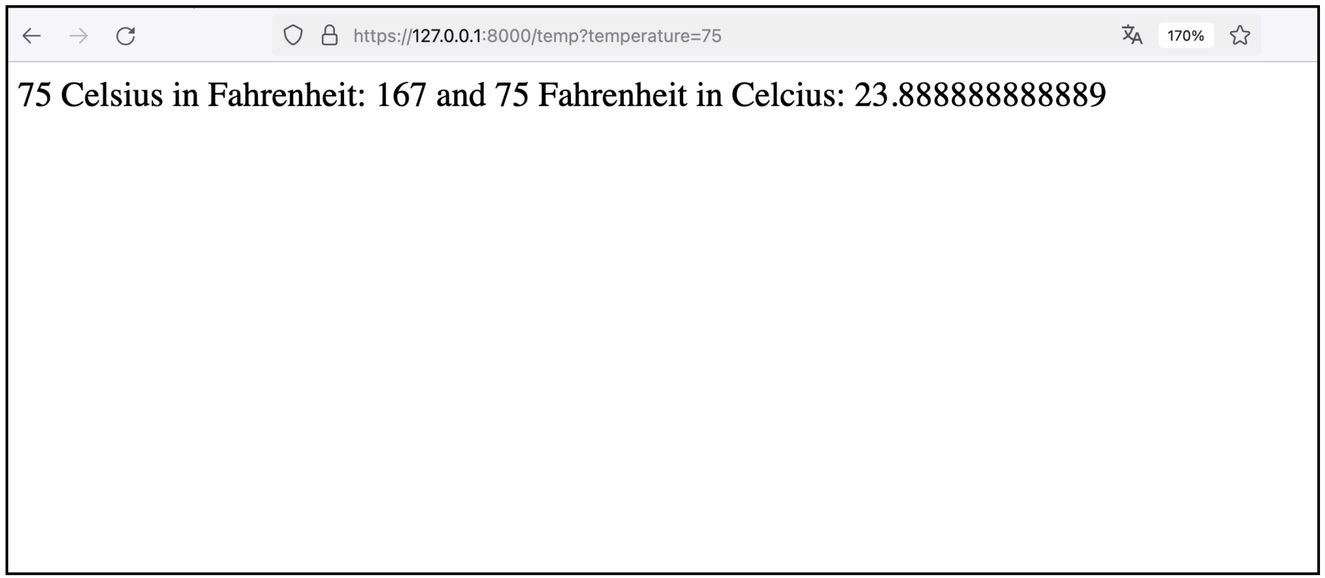
Create a new Symfony app to measure temperature in Fahrenheit/Celsius

How to Proceed? Steps:

- You can create a new Controller inside same helloSymfony project
- You can name your controller e.g., TemperatureController
- Get the temperature from the URL of browser request, you can use e.g.
  ```
  $temp = $request->query->get('temperature');
  ```
- Check if the temperature is valid
  ```
  if (!is_numeric($temp)) {
  // Return an error if the temperature is invalid
   return new Response("Error: Temperature must be a number", 400);
   }
  ```
- Calculate the converted temperature.
  ```
  $fahrenheit = ($temp * 9 / 5) + 32;
  // Return the converted temperature
  return new Response("The temperature in Fahrenheit : " . $fahrenheit)
  ```
- Test that your app works, and also test when your app throws an error if the temperature is invalid

Classroom Practice: Temperature Conversion App

1. Modify previous code a bit (to make it work in the same file)
2. Calculate the converted temperature also for Celsius
3. You can use formula e.g. $celcius = ($temp-32)/1.8;
4. Return the converted temperature
5. When you test your application, it should show result like this:



Twig

Symfony doesn't care *what* your controller looks like. But usually, you *will* extend a class called `AbstractController`.

Why? Because it gives us shortcut methods.

And the *first* shortcut is `render()`: the method for rendering a template. So return `$this->render()` and pass it two things. The first is the name of the template. How about `foldername/index.html.twig`.

Twig looks in the **`templates/`** directory. So when you create a new folder eg. **`Foldername/`** sub-directory... and inside of that, a file called **`index.html.twig`**. Twig files will be taken care of.

https://twig.symfony.com/

Twig

Twig is one of the ***nicest*** parts of Symfony, and also one of the **easiest**. We're going to go through everything you need to know...

Twig has exactly ***three*** different syntaxes. If you need to print something, use `{{`. You call this the **"say something"** syntax.

If I say **`{{ saySomething }}`** that would print a variable called `saySomething`.

Once you're *inside* Twig, it looks a *lot* like JavaScript. For example, if I surround this in quotes, now I'm printing the *string* `saySomething`. Twig has functions... so that would call the function and print the result.

So syntax #1 - the "say something" syntax - is `{{`

The second syntax... doesn't really count. It's `{#` to create a comment... and that's it.

Twig

The *third* and final syntax you may call is **"do something"** syntax. This is when you're not *printing*, your *doing* something in the language.

Examples of "doing something" would be **if statements**, **for loops** or **setting variables**.

Twig Inheritance

Head to https://twig.symfony.com... and then click to check its documentation. There's lots of good stuff here. But what I want you to do is scroll down to the Twig reference. Yea!

Tags

This list represents *every* possible thing you can use with the do something syntax. Yup, it will always be `{%` and then *one* of these things, like `for` or `if`.

Filters

Filters are basically functions, but with a more hipster syntax. Twig *does* also have functions, but there are fewer: Twig really prefers filters: they're way cooler!

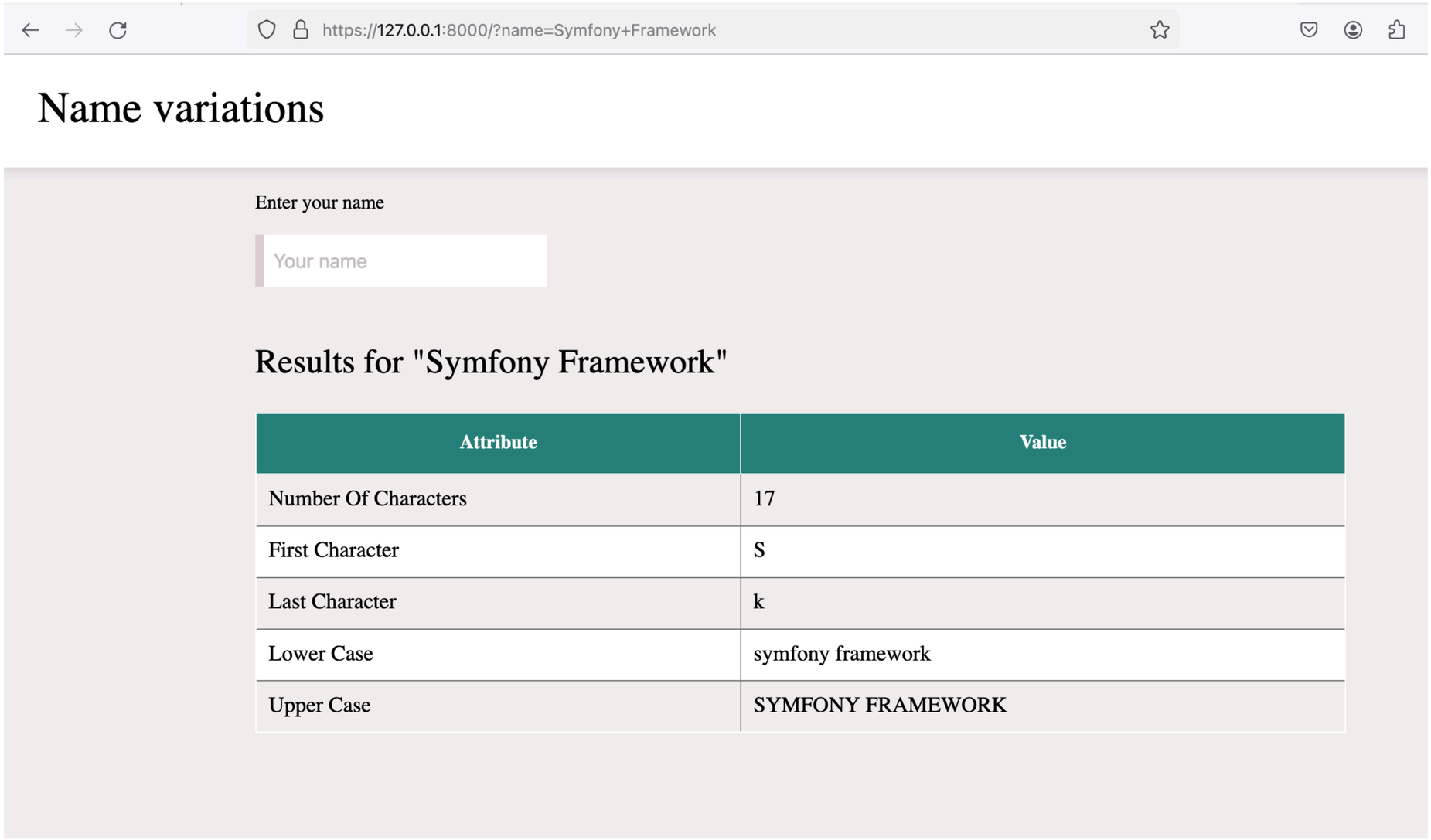
For example, there's a filter called `upper`. Using a filter is like using the `|` key on the command line. You have some value - then you "pipe it into" the filter you want, like `upper`.

e.g.

```
track.artist|upper.
```

https://twig.symfony.com/doc/

Name Variation App, Our second Symfony app



Name Variation App, Our second Symfony app

You know how to create a new Symfony project with Composer

symfony new namevariation

cd namevariation

Create a controller inside **src/Controller** e.g. NameVariationController.php

Copy everything what you had in HelloWorldController (but change the name to “Name Variation”

Your controller should be named, for example, `NameVariationController.php` in the `src/Controller` directory

Update your NameVariationController.php (shown in the next slide)

Name Variation App, Our second Symfony app

```php
class NameVariationController extends AbstractController {

    #[Route("/", name:"name_variation")]
    public function index(Request $request): Response
    {
        $name = $request->query->get('name', '');

        $nameDetails = [];
        if (!empty($name)) {
            $nameDetails = [
                'number_of_characters' => strlen($name),
                'first_character' => $name[0],
                'last_character' => $name[strlen($name) - 1],
                'lower_case' => strtolower($name),
                'upper_case' => strtoupper($name),
            ];
        }

        return $this->render('name_variation/index.html.twig', [
            'name' => $name,
            'nameDetails' => $nameDetails,
        ]);
    }
}
```

Name Variation App, Our second Symfony app

Next, create a Twig template for the view, for example, `index.html.twig` in the `templates/name_variation` directory

```
{% extends 'base.html.twig' %}

{% block title %}Name Variation{% endblock %}

{% block body %}
<main class="container">
<div>
    <label class="label" for="component-name">Enter your name</label>
    <form method="get" action="{{ path('name_variation') }}">
     <input type="text" name="name" class="input" placeholder="Your name" id="your-name" autocomplete="off">
    </form>
</div>

{% if name %}
    <h2>Results for "{{ name }}"</h2>
    <table class="table" border="1">
     <thead>
        <tr>
            <th>Attribute</th>
            <th>Value</th>
      </thead>
        </tr>
        {% for key, value in nameDetails %}
            <tr>
                <td>{{ key|replace({'_': ' '})|title }}</td>
                <td>{{ value }}</td>
            </tr>
        {% endfor %}
    </table>
{% endif %}
 </main>
{% endblock %}
```

Reflection - Symfony app and how it works

Let’s take a step back and in this reflection exercise, your task is to take 5-10 minutes and Reflect on how we did this app together:

Go back to 01- Symfony.pdf slides
Check slide number 24 - “Symfony - how it works”
Stare at the picture for 1-2 minutes
Run the “name variation app “
See and confirm does the boxes make any sense now?