AN INTRODUCTION TO SYMFONY 3

(for people that already know OO-PHP and some MVC stuff)

by

Dr. Matt Smith

2017

Acknowledgements

Thanks to ...

Table of Contents

1 Introduction

1.1 What is Symfony 3?

It's a PHP 'framework' that does loads for you, if you're writing a secure, database-drive web application.

1.2 How to I need on my computer to get started?

I recommend you install the following:

- PHP 7 (on windows Laragon works pretty well)
- a MySQL database server (on windows Laragon works pretty well)
- a good text editor (I like PHPStorm, but then it's free for educational users…)
- Composer (PHP package manager - on windows Laragon works pretty well)

or … you could use something like Cloud9, web-based IDE. You can get started on the free version and work from there …

1.3 How to I get started?

Either:

- install the Symfony command line installed, then create a project like this (to create a new project in a directory named `project01`):

  ```
  $ symfony new project01
  ```

or

- use Composer to create a new blank project for you, like this (to create a new project in a directory named `project01`):

  ```
  $  composer create-project symfony/framework-standard-edition project01
  ```

Learn about both these methods at the Symfony download-installer page and the Symfony setup page

or

- download one of the projects accompanying this book

1.4 Where are the projects accompanying this book?

There are on Github:

- https://github.com/dr-matt-smith/php-symfony3-book-codes

Download a project (e.g. `git clone URL`), then type `composer update` to download 3rd-party packages into a `/vendor` folder.

1.5 How to I run a Symfony webapp?

If you're not using a database engeine like MySQL, then you can use the Symfony console command to 'serve up' your Symfony project from the command line

At the CLI (comamnd line terminal) ensure you are at the base level of your oproject (i.e. the same directory that has your `composer.json` file), and type the following:

```
$ php bin/console server:run
```

2
First steps

2.1 It isn't working

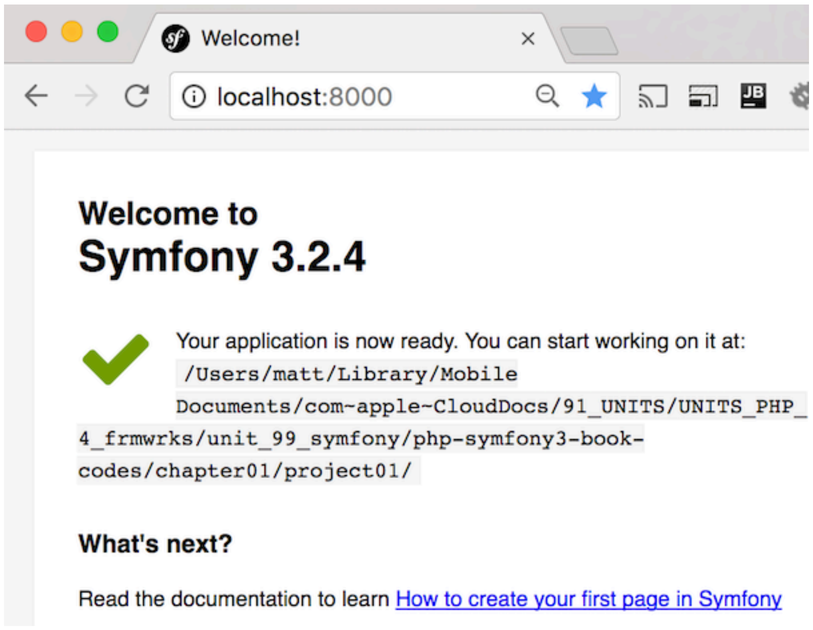
If you don't get the default Sfymfony home page, try this:

- copy the contents of `/web/app_dev.php` into `/web/app.php`

WARNING - this is just for now (we'll learn property Symfony configuration later). But this should get you going for now. You should NEVER do this for a project that might actually end up as a public production site!

2.2 All I get is the symfony home page (`project01`)

Figure 2.1 is your basic, default Symfony home page if everything is up and running for a new Symfony project.



Welcome to
Symfony 3.2.4

Your application is now ready. You can start working on it at:

```
/Users/matt/Library/Mobile
Documents/com~apple~CloudDocs/91_UNITS/UNITS_PHP_
4_frmwrks/unit_99_symfony/php-symfony3-book-
codes/chapter01/project01/
```

What's next?

Read the documentation to learn How to create your first page in Symfony

Figure 2.1: New Symfony project home page.

2.3 What we'll make (`project02`)

See Figure 2.2 for a screenshot of the new homepage we'll create this chapter.

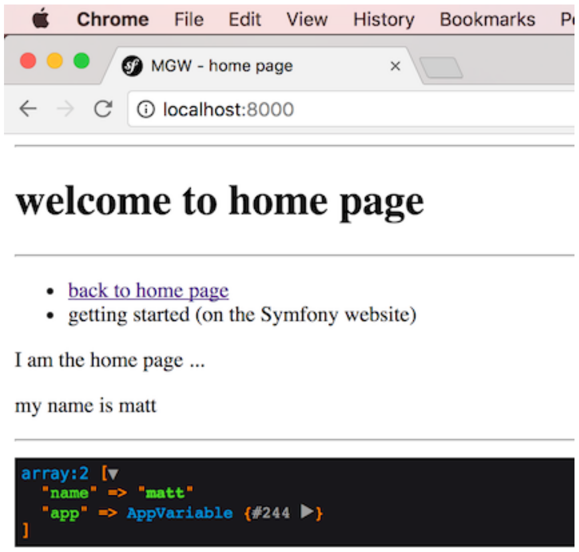
There are 3 things Symfony needs to serve up a page (with the Twig templating system):

1. a route
2. a controller class and method
3. a Twig template

The first 2 can be combined, through the use of 'Anotation' comments, which declare the route in a comment immediately before the controller method defining the 'action' for that route, e.g.:

```
/**
 * @Route("/", name="homepage")
 */
public function indexAction(Request $request)
{
    $argsArray =  [
        'name' => 'matt'
    ];

    $templateName = 'index';
    return $this->render($templateName . '.html.twig', $argsArray);
}
```



welcome to home page

- back to home page
- getting started (on the Symfony website)

I am the home page ...

my name is matt

```
array:2 [▼
  "name" => "matt"
  "app" => AppVariable {#244 ▶}
]
```

Figure 2.2: New home page.

```
}
```

The last (Twig template) can be a single file, and a simpler template that 'extends' a base template (which has all the standard doctype, css, js and core HTML structure in it).

If don't know much about Twig then go off an learn it (you can learn it stand alone, with a simple micro-framework like Silex, and as part of learning Symfony).

Here is our `_base.html.twig` template:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>MGW - {% block pageTitle %}{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>

    <hr>
        {% block body %}
        {% endblock %}

        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Here is the template for our index page, `index.html.twig`:

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

{% block pageTitle %}home page{% endblock %}

{% block body %}

    <h1>welcome to home page</h1>
    <ul>
        <li>
            <a href="{{ path('homepage') }}">back to home page</a>
        </li>
        <li>
        <a hrer="http://symfony.com/doc/current/page_creation.html">getting started (on the Symfon
        </li>
    </ul>

    <p>
        I am the home page ...
    <br>
        my name is {{ name }}
    </p>

    {{ dump() }}
{% endblock %}
```

2.4 First - get rid of all that default page stuff (`project02`)

A new Symfony project places its `DefaultController` at this location:

`/src/AppBundle/Controller/DefaultController.php`

Let's clear out the content of the controller, so there is no code in the body of the `indexAction()` method:

```
class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(Request $request)
    {
    }
}
```

(NOTE - leave all the 'uses' statements and the namespace, since they mean any classes we refer to, or annontations we use, all work correctly).

We want to use the template `index.html.twig`, since they all end in `.html.twig` let's concatenate that on later

```
    $templateName = 'index';
```

Twig templates expect to be given an associative array of any special data for the template, so let's illustrate this by passing a parameter `name` with your name (I'm Matt, so that will be my name parameter's value!):

```
$argsArray =  [
    'name' => 'matt'
];
```

There is nothing magic about the array identifier `$argsArray` - it's just a habit I've go in to when teaching Twig to my students - so change this (and anything - it's **your** project) to become more confident with working with the different bits of Symfony.

Symfomy's `Controller` class offers a hand method `render()` with accesses the Twig service in the Symfony application, so we can just invoke this method passing the template name (and appending the `.html.twig` string), and the array of arguments:

```
/**
 * @Route("/", name="homepage")
 */
public function indexAction(Request $request)
{
    $argsArray =  [
        'name' => 'matt'
    ];

    $templateName = 'index';
    return $this->render($templateName . '.html.twig', $argsArray);
}
```

Note that this final statement is a `return` statement. Basically any web application received (and interprets the contents of) an HTTP 'request', and builds and sends back an HTTP 'response'. They way Symfony (and most MCV webapps) work is that the controller method invoked for a given route has the responsibility of building and returning a 'response' (or sometimes just the text 'content' of a response, and the MVC application will build an HTTP response around that text content.

A
Appendix topics

Here are some appendices

List of References