Faculty of Science, Technology and Medicine

Web Programming

Volker Müller
University of Luxembourg

UNIVERSITÉ DU LUXEMBOURG

Symfony Framework (symfony.com)

PHP framework for web applications, aims at speeding up creation and maintenance of web sites

OO design: provides many decoupled, reusable components (also usable in other products)

Tries to realize design principles Separation of Concerns (separate computer program into distinct features that overlap in functionality as little as possible) and Convention over Configuration

Implements MVC

Tutorials on Symfony

On symfonycasts.com, many video tutorials on different aspects of Symfony are available, symfony.com contains a detailed tutorial

These might be quite helpful if you have to dig deeper into the system

I will provide now a few basic examples, see the official website for more details

Project Setup

Symfony is completely based on composer

Initial project setup: composer create-project symfony/website-skeleton MYPROJECT

Other components can also be installed at any time with composer (symfony.com/doc/current/components)

For a page, you have to define a controller, a view and a route to the page

Example Controller

```
<?php   // src/Controller/LuckyNumberController.php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Routing\Annotation\Route;

class LuckyNumberController extends AbstractController   {

    /**

     * @Route("/lucky/number", name="lucky_number")

     */

    public function index()

    {  …. return $this->render('lucky_number/index.html.twig',

            [ 'number' => 1 ]);  }  }
```

Creating Controllers

Controllers can be generated quickly with auto-generation:

```
php bin/console make:controller
```

→ Skeleton of a controller will be created

Note that every controller inherits from AbstractController

bin/console provides options for various auto-generations and other admin tasks

Routes

Routes can be defined with an annotation (@Route) or in an external configuration file config/routes.yaml

YAML format is exactly the same format as used in Docker configuration files

YAML is a human friendly data serialization standard for all prog. languages → yaml.org

Dynamic Routes

Routes are not necessarily static, but they can contain dynamic parts of "defined format" whose value can be accessed in annotated method:

```
/**  @Route("/blog/{page}", name="blog_list",
 *                   requirements = {"page" = "\d+"})
*/
 public function list($page) { ….}
```

Parameters of Controller Functions

Controller functions can have parameters:

```
public function index(Request $request,
SessionInterface $session)
```

Request object provides access to HTTP request data

SessionInterface is an interface for the session data

Similar for many other "services" (Symfony name for useful added functionalities) like logging

Twig (twig.symfony.com)

Modern templating engine for PHP: "fast, secure, flexible"

Very concise syntax:

`{{var}}` will output value of variable "var"

`{{var | escape}}` will output "var" with escaping

Twig is realizing the view in Symfony

In controller: `$this->render(filename_to_twig_file, array_of_variables_with_values);`

UNIVERSITÉ DU LUXEMBOURG

Why Templating?

Templates ensure that all pages will have the same overall structure

Templates can also be used recursively to define parts of a page with a common layout

Templates ensure that same CSS and JS code is included in every page

One web application can make use of many different templates

Twig Syntax Examples

{{ ... }} → output something

{% ... %} → do something (run code in body)

{# ... #} → Twig comment

Filters: {{ title | upper }}

Long list of available filters: abs, batch, date, escape, format, json_encode, lower, upper, path, reverse, url_encode …. (see documentation for full list)

A Note on Links in Templates

In Twig templates, links to other pages should never be hard-coded as string

In the definition of routes, we have seen a route name

Use the route name in every template:

```
<a href="{{path("routename")}}"> … </a>
```

This allows changing the route address without affecting other code (as long as name unchanged)

Twig Syntax Examples (cont.)

```
{% for i in 1..10 %}
  <div class="{{cycle([‘even’, ‘odd’], i)}}">
      ….
  </div>
{% endfor %}
```

Twig Syntax Examples (cont.)

```
<ul>  {% for user in users if user.active %}
              <li>{{ user.username }}</li>
        {% else %}
              <li>Inactive user</li>
       {% endfor %}
</ul>
```

Twig Templating – Base Template

```
<title>{% block title %}Test{% endblock %}</title> ….
  <div id="sidebar">
        {% block sidebar %}
           <ul> <li><a href="/">Home</a></li>
                 <li><a href="/blog">Blog</a></li>
           </ul>
        {% endblock %}
  </div> ...
```

Blocks can have default values

Twig Templating – Using Base Template

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

```
{% block title %}My cool blog posts{% endblock %}
```

If no value provided for block, then the default value from base template used

Template are usually stored in the /templates directory or sub-directories

Twig Documentation

Twig offers many more features

For a complete overview with many detailed examples, see the official website twig.symfony.com/doc/2.x/

Forms in Controller - Views

Forms for user input need specific code, not only HTML tags in view:

- Forms are represented as objects → "Entity class" must be defined with all input data (input elements = object attributes)
- Object handed over to view (as var "f"), rendered with {{form_start (f)}} …. {{form_end}}
- Controller calls function "handleRequest" for input submission

Simple Example for Form Object

```
$form = $formFactory->createBuilder()
    ->add('task', TextType::class)
    ->add('dueDate', DateType::class)
    ->getForm();  …..


return $this->render('new.html.twig',
[  'form' => $form->createView()   ]));
```

Doctrine

Doctrine is another very useful (third party) library tightly integrated into Symfony

Doctrine provides ORM ("Object-Relational Mapping")

Tables in DB $\leftrightarrow$ Classes in PHP

Users work with PHP classes, Doctrine "automatically" synchronizes these with used DB to make changes persistent $\rightarrow$ these classes represent a persisted model

Auto-Created Entities

Enter connection details in .env (in debug mode)

php bin/console doctrine:database:create

php bin/console make:entity will create the PHP class

php bin/console make:migration and php bin/console make:migrations:migrate will create the resp. DB table

Using Doctrine – Writing Data

Persist writes changes to a cache, but not yet to the DB → this is done by flush

```
use App\Entity\Student;
  ….
  $entityManager = $this->getDoctrine()->getManager();
  $student = new Student();
  $student->setName('VM'); ….
  $entityManager->persist($student);
  $entityManager->flush();
```

Using Doctrine – Reading Data

```
use App\Entity\Student;

  ….

  $this->getDoctrine()->getRepository(Student::class)

      -> find(1);
```

Repository for each entity class also automatically generated, provides list of functions provided to find entities satisfying some conditions

Other Features in Doctrine

Insert, update, delete are naturally possible

Search can also provide an array with conditions:

```
...-> findBy(array('age' => array(20,30,40)))
```

Relationships can be represented

Explicit query in a language similar to SQL also possible

See the tutorial for more details

<u>Remark:</u> ORM in JavaEE (JPA) very similar

Final Remarks

Only a small glimpse of possibilities shown (debugging and testing support still missing)

Learning Symfony requires some time, but then development can be done quite fast

I encourage you to go through the tutorial and to invest this time $\rightarrow$ 3rd (optional) exercise with Symfony gives up to 10% bonus if done

Next Meeting

The Basics of JavaScript