

Assignment2 Ricky Jenkins AWD

by Ricky Jenkins

FILE	12808_RICKY_JENKINS_ASSIGNMENT2_RICKY_JENKINS_AWD_178145_1788186316.DOCX (79.08K)		
TIME SUBMITTED	05-APR-2019 12:46PM (UTC+0100)	WORD COUNT	2245
SUBMISSION ID	104187800	CHARACTER COUNT	20864



Advanced Web Development: Assignment Two

By: Ricky Jenkins

submitted as part of the requirements for the award of BSc (Hons) Applied
Computing

Jan 2019

Table of Contents

1. Introduction.....	3
2. MVC framework.....	3
3. Symfony	4
4. ORM tools.....	5
5. Server-Side Templates.....	6
6. Bibliography.....	8
7. Appendix.....	10
FILENAME: BookController	10
FILENAME: DashboardController.....	15
FILENAME: UserController.....	16
FILENAME: Entity/Books	21
FILENAME: Entity/Users.....	24

1. Introduction

In this report, I will be discussing the uses of MVC frameworks in PHP. An MVC called Symfony and the advantages and disadvantages of using a MVC framework and compare other frameworks.

2. MVC framework

An MVC framework is made up of 3 main components, Modal, View and Controller.[1]

- Modal – A Modal will manage all the database connections and using the database.
- View – A view will manage all the HTML, CSS etc. What the user will see in the client.
- Controller – A controller manages the routes and the business logic that is implemented before going to the view.

[2][3] MVC is a design pattern used throughout multiple languages to split up the functionality of the program/site. There are many benefits with using MVC such as;

- Faster development
- Easier to expand and grow
- Being able to use multiple views
- Better file structure
- Better security, if implemented correctly
- Less code and repeating code
- Tools such as ORM, Debugging, testing

3. Symfony

With my practical side, I used Symfony as the framework of choice. Symfony can be installed with the full package or a skeleton package. Full package implementing all the resources that have been grouped up with Symfony or the skeleton package for the basic framework.

There are several reasons to use Symfony for its advantages such as:[4]

- Package levels – Symfony comes with several different installation packages, meaning the developer can assess what they would need in their application and install only the required parts, instead of installing the all framework.
- Flexibility – In most frameworks, being flexible depending on what the developer wants to use the framework is very important and Symfony does not disappoint. Symphony is very flexible with its multiple components and plugins.
- Well documented and maintained – As most frameworks do get well documented and maintained, Symfony shines due to the amount of support the team puts into their framework and tutorials that are produced.
- Reliability – Symfony is very known for the frameworks stability and reliability that it provides. There are several frameworks that have had issues in the past and still to this day, however Symfony is very stable and trusted with developers and companies. There are even components that other frameworks use from Symfony.
- Symfony has a built-in debugging/testing that developers can use to find bugs within their code and easily see the issues that are creating the bugs in the first place.

Symfony also has several disadvantages to its framework such as;

Learning Symfony - Symfony in some ways require a lot more in-depth research and practice on how to use the framework. It can be a lot more difficult than other frameworks such as Laravel, CodeIgniter etc.

Performance – in my experience and others, Symfony seems to have a performance problem due to it being a real-time application. Some performance issues with the framework can be sorted by changing some YAML and config files within the application. However, it still has a high load on the server that it operates on. Symfony is getting better with its performance but this is due to the PHP version being updated.

4. ORM tools

[5]ORM or Object relational mapping provides the ability to convert data into objects and objects into data. ORM is used throughout several languages and PHP has its own built in ORM called PDO. However, database handling and communication can become a problem sometimes and ORMs come to the rescue. ORMs virtually wraps themselves around communication between a programming language and the database. ORM's provide many methods to be used to communicate to the database and with that, deals with all the issues. when it comes to development within frameworks there is several ORM tools that you can use such as;

- Doctrine[6]
- Redbean[7]
- Propel[8]

There is also ORMs such as PDO and ADOdb however these are not the best for MVC frameworks.

[9] Within my practical part of the assignment, I used Doctrine due to the popularity and support it has within Symfony. Doctrine has several merits such as;

It has been around since 2006 and has high quality codebase

It is a stable ORM

Very flexible and rich with query features

Supports high-level and low-level database programming

Large size community meaning good documentation and tutorials

Is supported in several frameworks such as Symfony, Laravel, Zend and more.

5. Server-Side Templates

There are several template engines to use within an MVC framework, this does depend on the framework and what is compatible. A template engine gives easier control and reading ability of implementing back-end code into the view of an MVC framework. Frameworks normally come with a templating engine and they are as follows;

Symfony – Twig (separate component)

Laravel – Blade

CodeIgniter – comes with its own simple template parser.

[10] These are a few examples of templating engines that come with frameworks, however most frameworks are compatible with majority of the templating engines available. Templating engines are as follows;

- Twig
- FOIL
- Latter
- Haanga
- Dwoo
- RainTPL



- Smarty
- PH7
- Savant

[11]With the practical part of my assignment, I used twig and the template engine, I choice twig due to it being supported by Symfony already and the benefits it comes with such as;

- Unit Testing
- Documented and tutorials
- Easy to learn
- Template oriented syntax
- Secure
- Fast
- Debugging, outputting errors

6. Bibliography

- [1] Callum Hopkins, "PHP Master | The MVC Pattern and PHP, Part 1," <https://www.sitepoint.com/the-mvc-pattern-and-php-1/>, 2013. [Online]. Available: <https://www.sitepoint.com/the-mvc-pattern-and-php-1/>. [Accessed: 31-Mar-2019].
- [2] Marc PlotZ, "Principles Of MVC for PHP Developers," <https://www.htmlgoodies.com>. [Online]. Available: <https://www.htmlgoodies.com/beyond/php/article.php/3912211/Principles-Of-MVC-for-PHP-Developers.htm>. [Accessed: 31-Mar-2019].
- [3] Tony Marston, "The Model-View-Controller (MVC) Design Pattern for PHP," www.tonymarston.net, 2018. [Online]. Available: <https://www.tonymarston.net/php-mysql/model-view-controller.html>. [Accessed: 31-Mar-2019].
- [4] Steven Hansen, "7 Good Reasons to Use Symfony Framework for Your Project," <https://hackernoon.com/>, 2017. [Online]. Available: <https://hackernoon.com/7-good-reasons-to-use-symfony-framework-for-your-project-265f96dcf759>. [Accessed: 31-Mar-2019].
- [5] "Admin," "What are ORM Frameworks? – KillerPHP.com," <https://www.killerphp.com>, 2009. [Online]. Available: <https://www.killerphp.com/articles/what-are-orm-frameworks/>. [Accessed: 31-Mar-2019].
- [6] Doctrine, "Doctrine, The Open-Source PHP ORM and Persistence Tools Project," <https://www.doctrine-project.org/>, 2019. [Online]. Available: <https://www.doctrine-project.org/>. [Accessed: 31-Mar-2019].
- [7] RedBeanPhp, "RedBeanPHP :: Welcome," <https://redbeanphp.com>. [Online]. Available: <https://redbeanphp.com/index.php>. [Accessed: 31-Mar-2019].
- [8] Propel, "Propel, The Blazing Fast Open-Source PHP 5.5 ORM," <http://propelorm.org/>. [Online]. Available: <http://propelorm.org/>. [Accessed: 31-Mar-2019].
- [9] Doctrine, "Doctrine, The Open-Source PHP ORM and Persistence Tools

Project," <https://www.doctrine-project.org>, 2019. [Online]. Available: <https://www.doctrine-project.org/index.html>. [Accessed: 31-Mar-2019].

- [10] N/A, "13+ Best PHP Template Engines to Design your PHP Web Project! | Free & Premium Templates," www.template.net. [Online]. Available: <https://www.template.net/web-templates/php-templates/php-template-engine/>. [Accessed: 31-Mar-2019].
- [11] Symfony, "Home - Twig - The flexible, fast, and secure PHP template engine," <https://twig.symfony.com/>. [Online]. Available: <https://twig.symfony.com/>. [Accessed: 31-Mar-2019].

7. Appendix

FILENAME: BookController

#START

```
<?php
```

```
//Namespaces
```

```
namespace App\Controller;
```

```
//Classes from entity
```

```
use App\Entity\Users;
```

```
use App\Entity\Books;
```

```
//Classes from Model
```

```
//use businessLogic\BIUsers;
```

```
use App\Utils\BIUsers;
```

```
//Using Namespaces
```

```
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\HttpFoundation\Request;
```

```
//Using Annotations for routes
```

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

```
//Using to limit requests to be post/get/delete/update etc.
```

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
```

```
//Use for twig template engine.
```

```
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
```

```
use Symfony\Component\Filesystem\Filesystem;
```

```
use Doctrine\ORM\EntityManager;
```

```
class BookController extends Controller {
```

```
public function __construct() {
}

/**
 *@Route("books/viewall", name="viewallbooks")
 */
public function books() {
    $books = $this->getDoctrine()->getRepository(Books::class)->findAll();
    return $this->render('books/bookList.html.twig', array('books' => $books));
}

/**
 *@Route("book/deletebook/{id}", name="deletebook")
 */
public function deletebook($id) {

    $em = $this->getDoctrine()->getEntityManager();
    $Book = $em->getRepository(Books::class)->find($id);

    $em->remove($Book);
    $em->flush();

    $url = $this->generateUrl('viewallbooks');
    return $this->Redirect($url .");
}

/**
 *@Route("book/updatebook/{id}", name="updatebook")
 */
public function updateBook($id) {
    //check if the user has permission to be within this route/view
    $session = $this->get('session')->get('User');
    if($session['accesslevel'] >= 3)
    {
        //check if its a post request
        if($_SERVER['REQUEST_METHOD'] == 'POST')
```

```
{
    //check if email exists
    if(!empty($_POST['bookname'] && $_POST['bookname']) != NULL)
    {
        $BookName = trim($_POST['bookname']);
        $BookPrice = trim($_POST['bookprice']);
        $BookAuthor = trim($_POST['bookauthor']);
        $BookDescription = trim($_POST['bookdescription']);
        $BookDate = trim($_POST['bookdate']);
        $BookUrl = trim($_POST['bookurl']);
        $BookContent = trim($_POST['bookcontent']);

        $em = $this->getDoctrine()->getManager();
        $registerBook = $em->getRepository(Books::class)->find($id);
        //Setters for the data
        $registerBook->setBookname($BookName);
        $registerBook->setBookdescription($BookDescription);
        $registerBook->setBookprice($BookPrice);
        $registerBook->setBookauthor($BookAuthor);
        $registerBook->setBookdate($BookDate);
        $registerBook->setBookimg($BookUrl);
        $registerBook->setBookcontent($BookContent);
        // tell Doctrine you want to (eventually) save the Product (no queries yet)
        //EntityManager->persist($registerBook);
        // actually executes the queries (i.e. the INSERT query)
        $em->flush();
        $url = $this->generateUrl('viewallbooks');
        return $this->Redirect($url . "");
    }
}

$book = $this->getDoctrine()->getRepository(Books::class)->find($id);
return $this->render('books/bookUpdate.html.twig', array('book' => $book));

}
```

```
}  
/**  
 *@Route("books/register")  
 */  
public function bookRegister() {  
    //check if the user has permission to be within this route/view  
    $session = $this->get('session')->get("User");  
    if($session['accesslevel'] >= 3)  
    {  
        //check if its a post request  
        if($_SERVER['REQUEST_METHOD'] == 'POST')  
        {  
            //check if email exists  
            if(!empty($_POST['bookname'] && $_POST['bookname']) != NULL)  
            {  
                $BookName = trim($_POST['bookname']);  
                $BookPrice = trim($_POST['bookprice']);  
                $BookAuthor = trim($_POST['bookauthor']);  
                $BookDescription = trim($_POST['bookdescription']);  
                $BookDate = trim($_POST['bookdate']);  
                $BookUrl = trim($_POST['bookurl']);  
                $BookContent = trim($_POST['bookcontent']);  
  
                $entityManager = $this->getDoctrine()->getManager();  
  
                $registerBook = new Books();  
                //Setters for the data  
                $registerBook->setBookname($BookName);  
                $registerBook->setBookdescription($BookDescription);  
                $registerBook->setBookprice($BookPrice);  
                $registerBook->setBookauthor($BookAuthor);  
                $registerBook->setBookdate($BookDate);  
                $registerBook->setBookimg($BookUrl);  
                $registerBook->setBookcontent($BookContent);
```

```
// tell Doctrine you want to (eventually) save the Product (no queries yet)
$entityManager->persist($registerBook);

// actually executes the queries (i.e. the INSERT query)
$entityManager->flush();

$url = $this->generateUrl('viewallbooks');
return $this->Redirect($url . "");

//return new Response('Registered a Book and id is: '.$registerBook->getId());
}
}

return $this->render('books/registerbook.html.twig');
} else {
    $url = $this->generateUrl('dashboard');
    return $this->Redirect($url . "");
}
}

/**
 * @Route("book/view/{id}", name="viewbook")
 */
public function bookView($id) {
    $book = $this->getDoctrine()->getRepository(Books::class)->find($id);
    return $this->render('books/viewbook.html.twig', array('book' => $book));
}

/**
 * @Route("book/viewbook/{id}", name="viewcontent")
 */
public function bookViewbook($id) {
    $book = $this->getDoctrine()->getRepository(Books::class)->find($id);
    return $this->render('books/bookcontent.html.twig', array('book' => $book));
}
}
```

```
?>
```

```
#END
```

```
FILENAME: DashboardController
```

```
#START
```

```
<?php
```

```
//Namespaces
```

```
namespace App\Controller;
```

```
//Classes from entity
```

```
use App\Entity\Users;
```

```
use App\Entity\Books;
```

```
//Classes from Model
```

```
//use businessLogic\BIUsers;
```

```
use App\Utils\BIUsers;
```

```
//Using Namespaces
```

```
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\HttpFoundation\Request;
```

```
//Using Annotations for routes
```

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

```
//Using to limit requests to be post/get/delete/update etc.
```

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
```

```
//Use for twig template engine.
```

```
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
```

```
use Symfony\Component\Filesystem\Filesystem;
```

```
use Doctrine\ORM\EntityManager;
```

```
class DashboardController extends Controller {

    public function __construct() {
    }

    /**
     *@Route("/", name="dashboard")
     */
    public function index() {
        //return new Response('<h1>Hi</h1>');
        return $this->render('dashboard/index.html.twig');
    }

}

?>

#END
```

FILENAME: UserController

#START

```
<?php
//Namespaces
namespace App\Controller;

//Classes from entity
use App\Entity\Users;
```

```
use App\Entity\Books;

//Classes from Model
//use businessLogic\BIUsers;
use App\Utils\BIUsers;
//Using Namespaces
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
//Using Annotations for routes
use Symfony\Component\Routing\Annotation\Route;
//Using to limit requests to be post/get/delete/update etc.
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
//Use for twig template engine.
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Filesystem\Filesystem;

use Doctrine\ORM\EntityManager;

class UserController extends Controller {

    public function __construct() {
    }

    /**
     * @Route("users/login", name="login")
     */
    public function login() {
        //check if its a post request
        if($_SERVER['REQUEST_METHOD'] == 'POST')
        {
            //check if email exists
            if(!empty($_POST['username'] && $_POST['password']) != NULL)
```

```
{
    $username = trim($_POST['username']);
    $password = trim($_POST['password']);

    //$password = password_hash($password, PASSWORD_DEFAULT);
    $User = $this->getDoctrine()->getRepository(Users::class)-
>findOneBy(array('username' => $username));
}
if(!is_null($User))
{
    $Uname = $User->getUsername();
    $Upassword = $User->getPassword();
    $Uid = $User->getId();
    $Uaccesslevel = $User->getAccesslevel();

    if(password_verify($password,$Upassword))
    {
        $session = $this->get('session');
        $session->set('User', array('username' => $Uname, 'userid'=>$Uid,
'accesslevel' => $Uaccesslevel));

        $data = ['message' => 'sucessfully Loggedin'];
        $response = $this->forward('App\Controller\DashboardController::index', []);
        return $response;
    }
    else {
        $data = ['message' => 'Username/Password is wrong.'];
        return $this->render('users/login.html.twig', $data);
    }
} else {
    $data = ['message' => 'Username/Password is wrong.'];
    return $this->render('users/login.html.twig', $data);
}
}
```

```
$data = ['message' => ''];  
return $this->render('users/login.html.twig', $data);  
}  
  
/**  
 *@Route("users/register", name="register")  
 */  
public function register() {  
    //check if its a post request  
    if($_SERVER['REQUEST_METHOD'] == 'POST')  
    {  
        //check if data is suitable  
        if(!empty($_POST['username'] && $_POST['password']) != NULL)  
        {  
            $username = trim($_POST['username']);  
            $password = trim($_POST['password']);  
            $password = password_hash($password, PASSWORD_DEFAULT);  
            $User = $this->getDoctrine()->getRepository(Users::class)-  
>findOneBy(array('username' => $username));  
            if(is_null($User)) {  
                $entityManager = $this->getDoctrine()->getManager();  
                $register = new Users();  
                //Setters for the data  
                $register->setUsername($username);  
                $register->setPassword($password);  
                $register->setAccesslevel(1);  
                // tell Doctrine you want to (eventually) save the Product (no queries yet)  
                $entityManager->persist($register);  
                // actually executes the queries (i.e. the INSERT query)  
                $entityManager->flush();  
            }  
        }  
    }  
}
```

```
$data = ['message' => 'Account successfully created, please login.'];

$url = $this->generateUrl('login');
return $this->Redirect($url . '');
// $this->render('users/login.html.twig', $data);
} else {
    $data = ['message' => 'Username has already been taken'];
    return $this->render('users/register.html.twig', $data);
}

}
}
return $this->render('users/register.html.twig');
}

/**
 * @Route("/Logout", name="LogoutUser")
 */
public function logout() {
    $session = $this->get('session');
    $session = $this->get('session')->clear();
    // $session->invalidate(); // Didn't work for me!??
    $url = $this->generateUrl('dashboard');
    return $this->Redirect($url . '');
}
}

?>
#END
```

FILENAME: Entity/Books

#START

```
<?php
```

```
namespace App\Entity;
```

```
use Doctrine\ORM\Mapping as ORM;
```

```
/**
```

```
 * @ORM\Entity(repositoryClass="App\Repository\BooksRepository")
```

```
 */
```

```
class Books
```

```
{
```

```
    /**
```

```
     * @ORM\Id()
```

```
     * @ORM\GeneratedValue()
```

```
     * @ORM\Column(type="integer")
```

```
     */
```

```
    private $id;
```

```
    public function getId(): ?int
```

```
    {
```

```
        return $this->id;
```

```
    }
```

```
    /**
```

```
     * @ORM\Column(type="text")
```

```
     */
```

```
    private $bookname;
```

```
    public function getBookname() {
```

```
        return $this->bookname;
```

```
    }
```

```
    public function setBookname($bookname) {
```

```
$this->bookname = $bookname;
}
/**
 * @ORM\Column(type="text")
 */
private $bookdescription;

public function getBookdescription() {
    return $this->bookdescription;
}
public function setBookdescription($bookdescription) {
    $this->bookdescription = $bookdescription;
}
/**
 * @ORM\Column(type="text")
 */
private $bookprice;

public function getBookprice() {
    return $this->bookprice;
}
public function setBookprice($bookprice) {
    $this->bookprice = $bookprice;
}

/**
 * @ORM\Column(type="text")
 */
private $bookauthor;

public function getBookauthor() {
    return $this->bookauthor;
}
public function setBookauthor($bookauthor) {
```

```
$this->bookauthor = $bookauthor;
}

/**
 * @ORM\Column(type="text")
 */
private $bookdate;

public function getBookdate() {
    return $this->bookdate;
}
public function setBookdate($bookdate) {
    $this->bookdate = $bookdate;
}

/**
 * @ORM\Column(type="text")
 */
private $booking;

public function getBooking() {
    return $this->booking;
}
public function setBooking($booking) {
    $this->booking = $booking;
}

/**
 * @ORM\Column(type="text")
 */
private $bookcontent;

public function getBookcontent() {
    return $this->bookcontent;
```

```
}  
  
public function setBookcontent($bookcontent) {  
    $this->bookcontent = $bookcontent;  
}  
}  
#END
```

FILENAME: Entity/Users

#START

<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

```
/**  
 * @ORM\Entity(repositoryClass="App\Repository\UsersRepository")  
 */  
class Users  
{  
    /**  
     * @ORM\Id()  
     * @ORM\GeneratedValue()  
     * @ORM\Column(type="integer", length=12)  
     */  
    private $id;  
  
    public function getId(): ?int  
    {  
        return $this->id;  
    }  
}
```

```
/**
 * @ORM\Column(type="text", length=100)
 */
private $username;

public function setUsername($username) {
    $this->username = $username;
}

public function getUsername() {
    return $this->username;
}

/**
 * @ORM\Column(type="string")
 */
private $password;

public function setPassword($password) {
    $this->password = $password;
}

public function getPassword() {
    return $this->password;
}

/**
 * @ORM\Column(type="integer")
 */
private $accesslevel;

public function setAccesslevel($accesslevel) {
    $this->accesslevel = $accesslevel;
}

public function getAccesslevel() {
    return $this->accesslevel;
}
}
```

#END

Twig.Html. didn't copy here. View using
Root/Assignment2AWD/Symfony/Core/Templates/...

Assignment2 Ricky Jenkins AWD

GRADEMARK REPORT

FINAL GRADE

67 / 100

GENERAL COMMENTS

Instructor

Investigation of Fowler (30) 11

Investigation of Symfony (20) 16 Quality (20) 16

View handling (10) 9

Database handling (15) 14

Software testing (5) 1

You have discussed MVC in general, but there is little consideration of Fowler patterns.

The evaluation of Symfony lacks technical detail, and could have done with a couple of simple examples to illustrate its usage.

I would like to have seen more database search functionality.

I didn't find much evidence of testing.

Code is quite nice but report is a bit brief and lacks evaluation. I've adjusted the marking a little because the code provided is good.

PAGE 3

PAGE 4

PAGE 5

PAGE 6



Comment 1

Example would have been nice

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25
