SECURE WEB-BASED EXAM PAPER GENERATION SYSTEM

By

Ashley Holmes

Supervisor: Mr. Stephen Sheridan

SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE (HONOURS) IN COMPUTING
AT
INSTITUTE OF TECHNOLOGY BLANCHARDSTOWN
DUBLIN, IRELAND
23 OCTOBER 2016

Declaration

I herby certify that this material, which I now submit for assessment on the programme of study leading to the award of **Bachelor of Science (Honours) in Computing** in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Dated: 23 October 2016

Author: Ashley Holmes

Abstract

The main goal of this project will be to develop a secure web-based exam paper generation system. The system should support multiple users, subjects and courses. The end product should allow for the generation of pools of questions within a topic area and should also be able to generate exam papers based on a random selection of questions within a given pool or subject area.

The system should store historical data about which questions were used in what year and should avoid using the same question year on year. The system should also allow the user to have the ultimate say in what questions are chosen by allowing randomly selected questions to be replaced with other questions from with the same pool.

The system should also provide the ability to link marking schemes with questions and should ensure marking schemes are correct and add up to the correct totals. The system should be entirely secure from the login process to the storage of data and data in transit.

Keywords: Web-frameworks, Security, Encryption, Databases, HTML, CSS, Java

Motivation: With the growth of mature students entering third level educational institutions around Ireland. Mainly due to the economic downturn which has resulted in many working class people finding themselves unemployed.

It is for this reason why programmes were put into place by the Irish Government and European Union to establish a structure by investing in these peoples education and skills and thus offering them support when it comes to seeking employment once again. Either in fields which they are familiar with or something completely different.

It is initiatives like these which is supported and funded by authorities and governing bodies such as SUSI and the ESF Ireland which have lead to bringing the unemployment rate of persons aged 25 - 74 years down in the last two years by 2.7% CSO (2016) ESF (2016).

The difference between the total number of people now employed since September 2014 and September 2016 is 50, 200 persons. That is a great deal of people in terms of a two year period. These people attending the institutions could have not seen a classroom for more than twenty to thirty or up to forty years or more let alone used a computer and thus require a greater deal of help or attention. As some of them might be exploring an IT field.

It is this one on one attention that they desperately need and freeing up a lecturers time for this would be of great benefit. Since being exposed to college life and the reality of being a mature student have witnessed this firsthand. Most lecturers teach numerous subjects. And for each subject they need to put together numerous exam papers to account for semesters and repeats. This could total many hours of composition for their many subjects.

If a system was in place to automatically generate exam papers this will take away from the time that it takes to compose them. They would be able to give this time back to their students. In the form of a meeting for Q & A or for revision work. This is the impact that an exam paper generation system could have on the lives of mature students if successful.

Problem statement: Compiling an exam paper which will test the students knowledge in a particular subject is challenging in its own way. Firstly, there are the time constraints. Examiners a being faced with more and more work each year within the same space of time. If you add up the amount of time per semester over a given year which is set aside to put together an exam paper it will add up to many hours.

There needs to be a better approach to minimise these hours. It will take a great deal of time to produce a good quality paper. The questions need to be taken from the curriculum which was or is being delivered to the students over the semester. This brings upon the need to develop a paper from as many of the important areas of the module as possible. As this will be the process of determining the students

performance with regard to the questions which are asked and the complexity of them. The result of good exam question will determine the sort of student the college will produce.

Approach: The approach towards this project will be to do as much research as possible around the work of others. Reading the research papers which are available on the internet and in journals.

Which technologies and methodologies were used and incorporated into their work. If they had any shortcomings. See which improvements can be made. One can streamline a complicated version if one is available. And combining the methods of others to form one which is more successful. Furthermore, how long ago their work took place as perhaps a technology has caught up to what they were trying to achieve and would now be in a position to overcome any weaknesses.

For the purpose of this project the SDLC of choice will be the Agile Model. Which takes measures such as planning, analysis, designing building and testing. This will be done in small increments. Each time building on what is currently available and adding to it. Until all features are in place for a full system release.

Results: A conclusion has yet to be established. The results should represent a system which is better than the one in place offering a defined gain to the current method for examination compilation. This will be determined once the system is in place and can be tested in a scientific manner by comparing the two methods.

Conclusions: To follow.

Acknowledgements

I would like to thank my wife Jennifer Holmes, my son Hayden Holmes and Mr. Stephen Sheridan my supervisor and mentor...

Table of Contents

Abstract ii

Acknowledgements v

Table of Contents vi

List of Tables xii

List of Figures xiii

Abbreviations xviii

1 Introduction and Background 1

1.1 Title 1

1.2 Background 1

1.3 Main Research Question(s) 2

1.4 Justification / Benefits 2

1.5 Feasibility 3

1.6 Proposed Methodologies 3

1.7 Expected Results 3

1.8 Conclusion 5

1.9 Project plan 5

2 Literature Review **7**

2.1 Abstract 7

2.2 Literature Review 7

3 Methodology Chapter **12**

3.1 Introduction 12

3.1.1 What is Agile? 12

3.1.2 How does it work? 14

3.1.3 How is it different? 15

3.1.4 Agile vs Waterfall 16

3.2 Data Collection Methods 16

3.2.1 Internet Search 16

3.2.2 Supervisor Input 16

3.2.3 Journals in Library 16

3.3 Method of Analysis 16

3.3.1 Formulation of Where to Start 16

3.3.2 Early System Implementation 17

3.3.3 Review of Literature 17

3.4 Summary 17

4 Implementation - ”Building the solution” **18**

4.1 Introduction 18

4.2 Terminal 18
4.2.1 Command Line Instructions 18
4.3 Browser 23
4.3.1 Deploying the Application 23
4.4 IDE 28
4.4.1 PhpStorm IDE for PHP 28
4.5 Database 32
4.5.1 Creation of User Entity and CRUD 32
4.6 Twig 36
4.6.1 Templates are fine tuned 36
4.7 Forms 39
4.7.1 Understanding FormTypes 39
4.8 Validation 41
4.8.1 Validation of the Forms 41
4.9 Theming 44
4.9.1 Form Theming 44
4.10 Fixtures 45
4.10.1 Fixtures with Faker 45
4.11 Passwords 50
4.11.1 Password Fixtures 50
4.11.2 Password Services 54
4.11.3 Change Password Form 58
4.12 Authentication 61
4.12.1 Login Form 61

4.13 Controllers 63
4.13.1 Extending the controller 63
4.14 Roles 66
4.14.1 Putting roles in the database 66
4.14.2 Using the roles in the database 68
4.15 Authorisation 70
4.15.1 Access Control 70
4.16 CSRF Protection 71
4.16.1 Testing the roles 71
4.16.2 Controller and templates adjustments based on user roles 73
4.17 Registration 75
4.17.1 Registration Controller 75

5 Testing - ”Testing and evaluation” 79

5.1 Introduction 79
5.2 Testing the Homepage 79
5.2.1 Functional Testing 79
5.3 Testing Registration 82
5.3.1 Successful Registration 82
5.4 Testing Login 82
5.4.1 Successful Login 82
5.5 Testing Editing 83
5.5.1 Successful Editing Function 83
5.6 Testing Password 83

5.6.1 Successful Password Editing 83
5.7 Testing ROLES 87
5.7.1 ROLE Changes 87
5.8 Testing Exam Cover Page 87
5.8.1 Successful Form Validation 87
5.9 Testing Question Page 89
5.9.1 Successful Form Validation 89

6 Implementation of the System 96

6.1 Implementation Principles 96
6.1.1 Object-Oriented Approach 96
6.1.2 Design Patterns 96
6.1.3 Choice of Language 96
6.2 Stages of Admin Implementation 96
6.2.1 Login 96
6.2.2 Administration 97
6.2.3 Subsection header 3 97
6.3 Stages of User Implementation 97
6.3.1 Subsection header 1 97
6.3.2 Subsection header 2 97
6.3.3 Subsection header 3 97
6.4 Design 97
6.4.1 Subsection header 1 97
6.4.2 Subsection header 2 97
6.4.3 Subsection header 3 98

7 Testing and Evaluation **99**

7.1 Introduction 99

7.2 Tests Conducted 99

7.3 Algorithms 99

7.4 Summary 99

8 Conclusion and Future work **100**

8.1 Contributions 100

8.2 Limitations 100

8.3 Future Work 100

8.4 Data Collection 100

Bibliography **101**

Appendices **103**

A Web Application Login Screen 104

B Web Application Question Entry 105

C Generate a Question 106

D Show questions 107

E Show 108

List of Tables

1.1 Table to represent the Work Breakdown Structure 6

List of Figures

1.1 Graphical illustration of the Agile Model (tutorialspoint, 2016) 4

1.2 Gantt Chart 5

2.1 Schematic diagram of the system function module (tutorialspoint, 2016) . . 9

2.2 Technology road-map of the system (tutorialspoint, 2016) 9

2.3 Flow chart of the automatic paper generation method (tutorialspoint, 2016) 10

3.1 Graphical illustration of the Agile Model (tutorialspoint, 2016) 13

3.2 Increments of the Agile Model (agile in a nutshell, 2016) 14

3.3 Iterations of the Agile Model (agile in a nutshell, 2016) 14

3.4 Making a list (agile in a nutshell, 2016) 15

4.1 Symfony Documentation 19

4.2 Symfony Installation Setup 19

4.3 Symfony Application Setup 20

4.4 Terminal Window Top 21

4.5 Terminal Window Bottom 21

4.6 Php Server Run 22

4.7 Php Server Start 22

4.8 Symfony Browser 23

4.9 Configuration Checker 24
4.10 Web Debug Toolbar and Profiler Extended 25
4.11 Web Debug Toolbar and Profiler Extended 26
4.12 Web Debug Toolbar and Profiler Extended 27
4.13 Web Debug Toolbar and Profiler 28
4.14 DefaultController 29
4.15 Twig template in IDE 29
4.16 Adding Bootstrap 30
4.17 Adding Bootstrap 31
4.18 Home Page 31
4.19 Parameters Yaml 32
4.20 Mamp Ports 33
4.21 Entities 34
4.22 User Table 34
4.23 Twig templates 35
4.24 Form Type 35
4.25 index.html.twig 37
4.26 new.html.twig 38
4.27 show.html.twig 38
4.28 edit.html.twig 39
4.29 UserType.php 40
4.30 Form Label 41
4.31 Form Validation 42
4.32 Entity Validation 42
4.33 Constraints 43

4.34 Form Errors 44
4.35 Form Theming 45
4.36 Button Theming 46
4.37 Form Theming Result 46
4.38 Fixtures 47
4.39 Bundles 48
4.40 Faker 48
4.41 Object Manager Flush 49
4.42 Purge Database 50
4.43 Security.yml 51
4.44 Hash Password 52
4.45 getSalt Method 53
4.46 Password 53
4.47 UserInterface 54
4.48 UserManager 56
4.49 Yaml Services 57
4.50 UserController 57
4.51 Password Twig template 58
4.52 Change Password Form 59
4.53 passwordAction Controller 59
4.54 createPassword Form 61
4.55 Password Form Validation 62
4.56 SecurityController 63
4.57 Login Form 64
4.58 Symfony Login Profiler 64

4.59 Symfony Login Profiler 65
4.60 Login Form Error 66
4.61 User Roles in the the base template 67
4.62 UserRepository 67
4.63 JSON array 68
4.64 isAdmin 69
4.65 CheckboxType 70
4.66 CSRF Token 72
4.67 CSRF Token Browser 72
4.68 editAction 74
4.69 Show Page 74
4.70 User not in ROLES 75
4.71 RegisterController 76
4.72 Register Twig Template 77
4.73 Register with Errors 77
4.74 Registration Successful 78

5.1 Intentional Error Register 80
5.2 Intentional Error Login 80
5.3 Debug Error Register 81
5.4 Profiler Error Register 81
5.5 Register Test 82
5.6 Successful Registration 83
5.7 Database Check 84
5.8 Mr Oz Machado Login 84

5.9 Oz Machado Authenticated 85
5.10 Oz Machado Index Page 85
5.11 Oz Machado Show Page 86
5.12 Oz Name Changed 86
5.13 Oz Password Changed 87
5.14 Oz Password Changed DB 88
5.15 Oz ROLES Changed 88
5.16 Oz ROLES Changed in DB 89
5.17 Cover Form Validation 90
5.18 Cover Generation Step 1 90
5.19 Cover Generation Step 2 91
5.20 Cover Generation Step 3 91
5.21 Cover Generation Step 4 92
5.22 Cover Page Successful 92
5.23 Cover Page Successful DB 93
5.24 Question Page Successful 93
5.25 Question Page Successful DB 94
5.26 Question Page Metadata 94
5.27 Cover Page Successful DB 95
5.28 Cover Page Successful DB 95

1 Graphical illustration of the Login Screen 104
2 Graphical illustration of the Question Entry 105
3 Graphical illustration of the Generate a Question Paper 106
4 Graphical illustration of the Menu List to view the Questions 107
5 Graphical illustration of the Show list 108

Abbreviations

SUSI	Student Universal Support Ireland
CSO	Central Statistics Office
ESF	European Social Fund
IT	Information Technology
SDLC	Systems Development Life Cycle
REQ	Requirement Analysis
UML	Unified Modeling Language
ERD	Entity Relationship Diagram
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
IDE	Intergrated Development Environment
JSP	JavaServer Pages
BLOB	Binary Large Object
SQL	Structured Query Language
PHP	Hypertext Preprocessor
JAVA	Just Another Vulnerability Announcement
CDN	Content Delivery Network
ORM	Object Relational Mapping
YAML	YAML Ain't Markup Language
JSON	Java Script Object Notation
CSRF	Cross-Site Request Forgery
ACL	Access Control List

DB	Database
MAC	Mackintosh
OS	Operating System

Chapter 1

Introduction and Background

1.1 Title

SECURE WEB-BASED EXAM PAPER GENERATION SYSTEM

1.2 Background

With every new college year and semester, lecturers are faced with the prospect of composing examination papers for the next coming months. Since this can prove to be a very tedious and physically demanding task. More over, it can also be very challenging due to the time consumption and nature of the process for the examiner. The traditional method of composing papers can be automated. Therefore, there exists an opportunity to provide a service to simplify the process.

The use of a Web-based examination paper generation system which makes use of a relational database and database tables to cross-reference the newly created table of randomised questions with the tables from the previous years or semester. Resulting to a non-repeating question sheet.

1.3 Main Research Question(s)

1. **What will the end user experience be like, or will they prefer the old fashion method to what they are used to?**

   Not everyone can adapt to change as well as others. This is why getting used to a new system could take some time. Some folk may even get frustrated to the point where they find the new interface impossible to use. This is not the intension. The project is meant to make the process more streamlined and user friendly. This is why a great deal will be taken in the design of the user interface to make the experience a pleasurable one.

2. **Could the presence of an automated generation of questions system improve the accuracy of questions over a manual generation?**

   There are many factors which can affect a human beings output when given a task. These factors could range from fatigue. Being distracted by a colleague. Or not having the focus needed to complete the task at hand. This is where machines have the advantage over us. Humans suffer from what is called, ”Human error.” Whereas a machine can produce the same output with precision and repetition. This is why an automated system would work in college environment.

1.4 Justification / Benefits

When it comes to that time of year where lecturers need to set aside the time to create their examination papers for the modules which they deliver. This is where this project will come into its own with the aim of taking the stress out of the procedure and to provide examiners an easy to use means of examination paper compilation. This usability will come from a combination of a clean and simple user interface along with useful tools to create examination papers.

1.5 Feasibility

Since there are numerous examples of this implementation on the internet. This comes from reading research papers from other students in colleges and technical institutions all around the world. Furthermore the prerequisites obtained from this projects supervisor ensures that the project is technically feasible. However, some research needs to be undertaken regarding the security and encryption aspects. This will be the main technical difficulty and therefore there needs to be a sufficient technical understanding of the technologies involved in order to complete the project.

1.6 Proposed Methodologies

To articulate the methods and techniques used in this plan. Below is the outcome after reviewing various SDLC methodologies with reference to tutorialspoint (2016):

- Adoption of the Agile Model.
- Suits the requirements for this project.
- Widely accepted within companies within the IT industry.
- Valuable learning curve in gaining experience with this model.
- Model has the ability to adapt and tailor itself within each increment as the project moves forward.
- Advantageous to the project.

1.7 Expected Results

As noted in the Feasibility section the project should be feasible from a technical standpoint. It is therefore expected that the project will result in a fully-functioning web site that makes

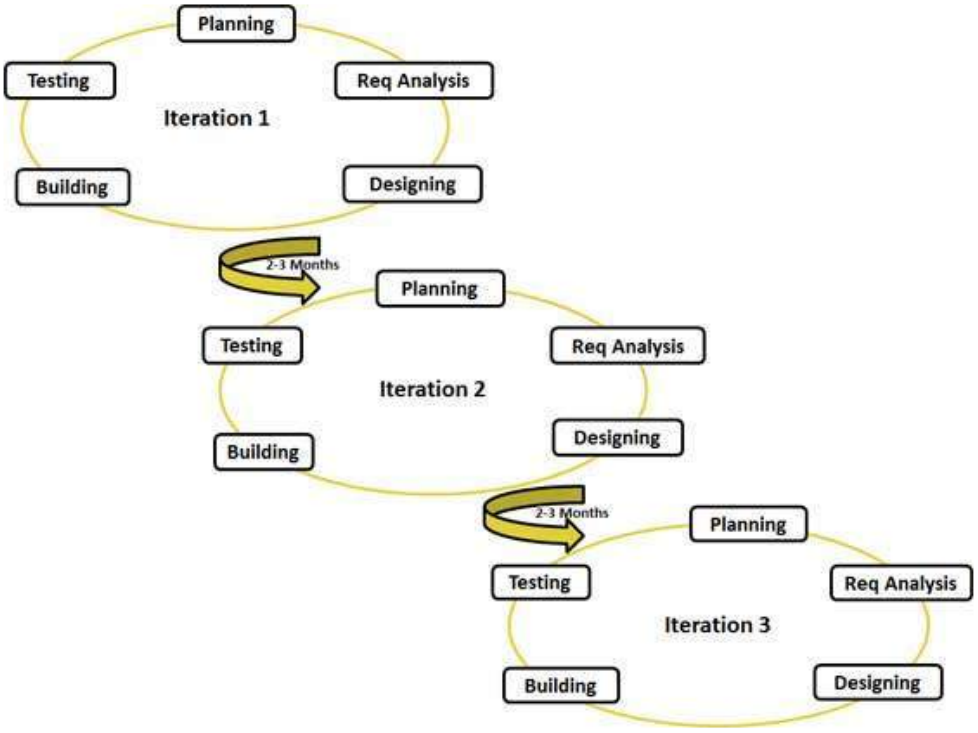

Figure 1.1: Graphical illustration of the Agile Model (tutorialspoint, 2016)

use of the technologies provided.

1.8 Conclusion

This project aims to provide a simple and easy to use service through the use of various Internet technologies combined with automatic generation of question papers and functions. It is hoped that such a service can reduce both the time and difficulties experienced by examiners during an busy time of the year.

1.9 Project plan

Table 1.1 Which shows the Work Breakdown Structure.

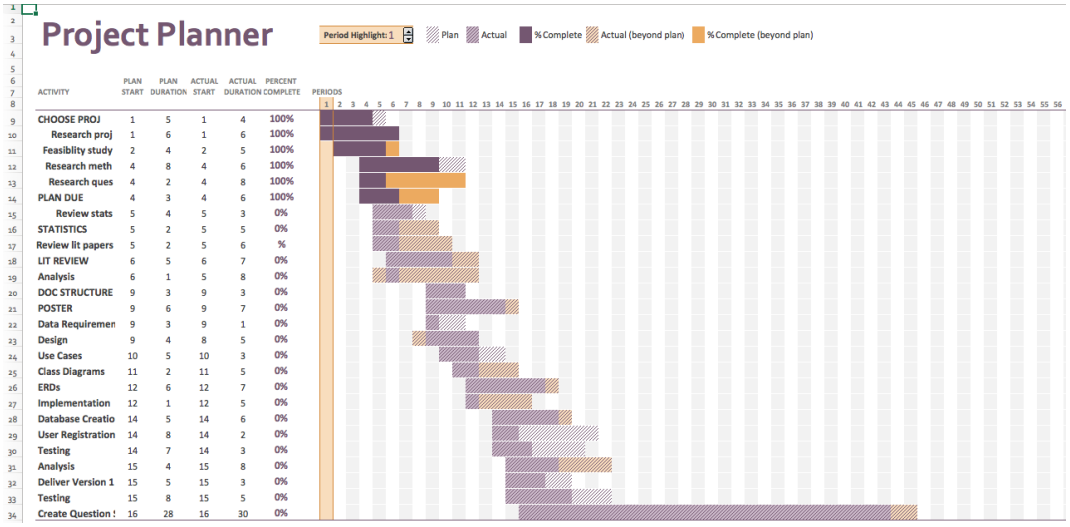

Figure 1.2: Gantt Chart

Task Name	Start	Finish	Duration
Planning	12/09/2016	17/10/2016	42d
Project Plan	12/09/2016	26/09/2016	14d
Research Project Ideas	12/09/2016	26/09/2016	5d
Project Proposal	27/09/2016	12/10/2016	5d
Feasibility Study	12/10/2016	14/10/2016	13d
Research Methodologies	14/09/2016	16/10/2016	7d
Create Project Proposal	16/10/2016	26/10/2016	5d
Submit Project Proposal	26/10/2016	26/10/2016	0d
Literature Review	26/10/2016	31/10/2016	2d
Submit Literature Review	31/10/2016	07/11/2016	0d
Development	07/11/2016	12/12/2016	108d
Version 1	12/12/2016	12/09/2016	28d
Analysis	14/09/2016	30/09/2016	7d
User Registration	14/09/2016	30/09/2016	7d
Question Entry	28/10/2016	12/09/2016	7d
Create Question Section	28/10/2016	12/09/2016	7d
Design	28/10/2016	31/10/2016	7d
Use Cases	28/10/2016	31/10/2016	2d
Class Diagrams	31/10/2016	07/11/2016	2d
ERDs	31/10/2016	07/11/2016	2d
Wireframes	31/10/2016	07/11/2016	1d
Implementation	31/10/2016	12/09/2016	14d
Database Creation	31/10/2016	07/11/2016	2d
Home Page	31/10/2016	07/11/2016	3d
User Registration	31/10/2016	07/11/2016	4d
Question Entry Page	31/10/2016	07/11/2016	4d
Deliver Version 1	07/11/2016	12/12/2016	0d
Testing	07/11/2016	12/12/2016	11d
Review Version 1	07/11/2016	12/12/2016	7d
Analysis	07/11/2016	12/12/2016	7d
View Listings	07/11/2016	12/12/2016	7d
Show Questions	12/12/2016	12/12/2016	7d
Generate Questions	12/12/2016	15/12/2016	7d
Design	12/12/2016	15/12/2016	7d
Use Cases	12/12/2016	15/12/2016	3d
Wireframes	12/12/2016	15/12/2016	2d
ERDs	12/12/2016	15/12/2016	3d
Implementation	09/01/2017	15/12/2016	14d
Database Changes	09/01/2017	15/12/2016	2d
View Questions Page	09/01/2017	09/01/2017	4d
Generate Question Page	09/01/2017	09/01/2017	4d
Testing	09/01/2017	09/01/2017	12
Deliver Version 2	09/01/2017	09/01/2017	7d

Table 1.1: Table to represent the Work Breakdown Structure

Chapter 2

Literature Review

2.1 Abstract

This chapter looks at existing research and development samples undertaken by other students from many countries around the world. These undertakings which have been published were sourced from publishings in academic papers, journal articles and books and gathered together from the major works to form part of the research of this narrow topic as they are in the same field of various implementations of random and automatic examination paper generation.

2.2 Literature Review

(Guang Cen, 2010) presented a method to eliminate (Mumbai, 2016) the tradition of the manual composition of examination papers which would usually rely on the writers own experience and style of question and knowledge. Although great care would be taken to achieve the best possible outcome of questions with traditional methods there was still the problem of a limited scope of topics and a time consideration. This would bring upon the separation between teaching and creating test papers by means of an automated computer system (Yang Yu, 2008). Comprised of JEE the test system includes modules such as user, subject, question, paper and classification management. Included in this is a question entry

and generation module. These modules can be seen in Figure 1 Schematic diagram of the system function module The question entry and generation makes use of browser and server architecture with a connection to a database of questions. Between this layer is a test server and a WWW server making up the middle layer (Chen, 2008). Figure 2 Technology road-map of the system shows a flow chart of the system architecture and the use of the MVC pattern with a JSP view, Java Beans, Servlet Controller and a MySQL database (Liu, 2008). The page layout uses divs and CSS technology. In addition to this is support from JavaScript (R. Johnson, 2004). It is the browser which allows the user to choose the subject which they intend on examining. A question type such as student input and a difficulty level. With all these combined parameters, a paper is generated using the generation algorithm (Wang, 2008). This will then be stored in the test database which can be recalled at any stage through system functions for query, or to update the database and for maintenance. It runs in separated modes for user and administrator use. In the end the final document is processed into a Microsoft Word .doc file for distribution in an exam environment. From Figure 3 Flow chart of the automatic paper generation method it shows how the document is generated.

There are 3 categories which this system falls under. They are, random algorithm based systems, backtracking systems and artificial intelligence and information processing. The first two do not satisfy the specifications (Guiying Deng, 1998). It is the latter which has been improved to avoid the disadvantages of the first and second algorithms. Giving it the ability of searching for questions based on experience and knowledge which guarantees a high standard and quality of examination papers (Hou, 2003). Through using a system with artificial intelligence and information processing the algorithm works quickly and effectively by not selecting a repeated question in a random manner. Questions and answers are separated. It also allowed the user a choice of topics, degree of difficulty, proportion or mark allocation and number of questions per section.

(Yajuan Zhang, 2011) proposed that although the traditional algorithms in a test paper generation system satisfy the requirements of shuffling the questions. Under certain constraints they do not perform as well as others which have been newly adopted. Here follows a

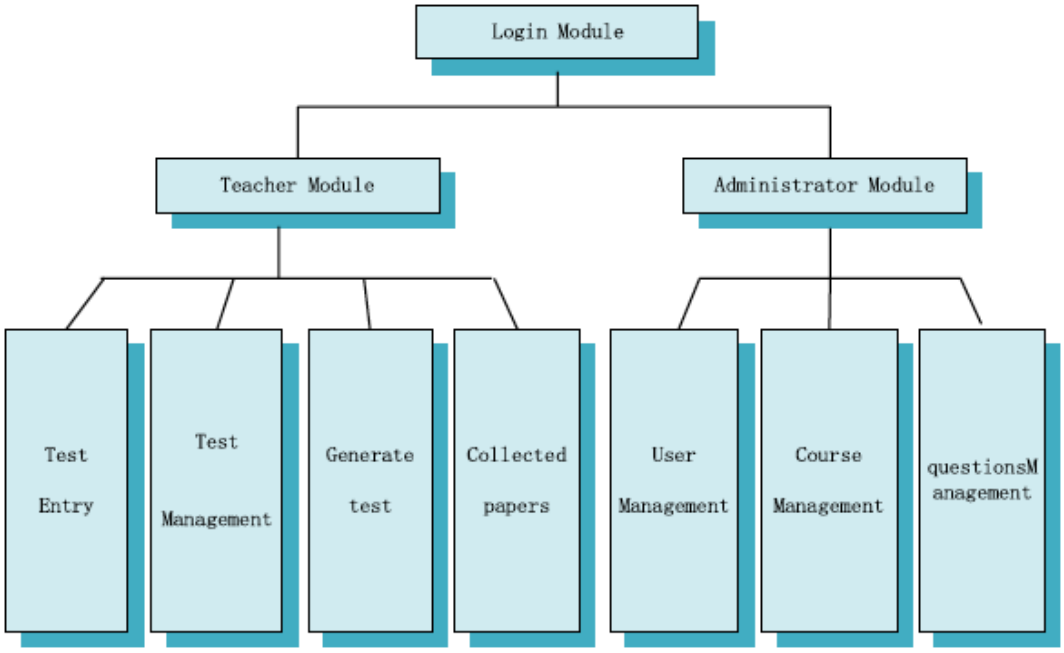

Figure 2.1: Schematic diagram of the system function module (tutorialspoint, 2016)

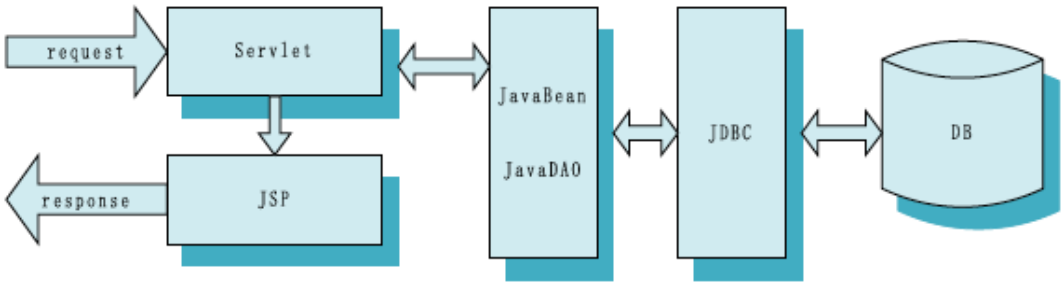

Figure 2.2: Technology road-map of the system (tutorialspoint, 2016)

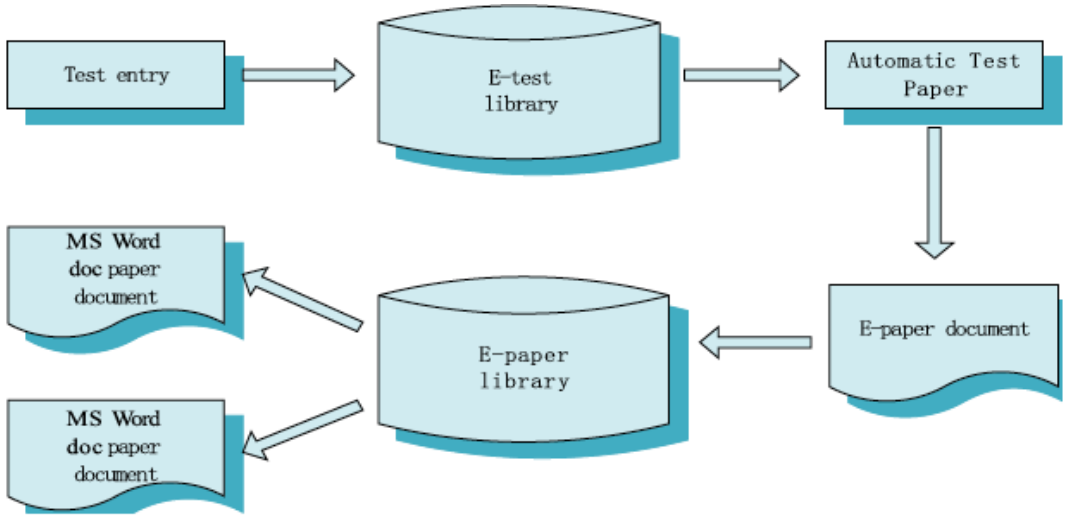

Figure 2.3: Flow chart of the automatic paper generation method (tutorialspoint, 2016)

discussion of analysing five intelligent algorithms and how these existing global optimisation algorithms can be integrated into improved shared global optimisation algorithm and dynamic multi branches tree algorithm. These included, improved genetic algorithm, differential evolution algorithm and ant colony optimisation. The particle swarm optimisation algorithm and simulated annealing algorithm. These are divided into two categories which are population evolutionary and others are individual evolutionary and each uses different searching and selection mechanisms. There have been many different studies on trying to improve the algorithms used in terms of speed optimisation however, the improvements were only minor ones. And with the expansion of the system new classifiers need to be constructed for the added samples. The characteristics of the different global optimisation algorithms such as the improved genetic algorithm. Based on the genetic algorithm with modifications such as improvements to integer coding which displays higher search speeds performing well and is very practical. It also avoids prematurity which occurs in the genetic algorithm. The genetic algorithm performs a randomised search simulating natural selection and genetic variation to problem solve. With the disadvantage of having a low search efficiency with premature convergence. The differential evolution algorithm is simple and effective in that the population size remains unchanged throughout the operation process. These operations

include variation, crossover and selection with advances such as simple principle, control parameters, robustness a high convergence rate and straightforward realisation. The ant colony algorithm simulates an ant colony and their routing behaviours in nature. Finding a solution through information exchange and cooperation among the ant colony. However, the mechanism for feedback has a slow convergence speed. The particle swarm optimisation algorithm has good performance however, needs to be used indirectly in getting the optimal solution of multiple object optimisation problems. As during a search its own position needs to be updated through a follow up of individual extreme value and global extreme value. Simulated annealing algorithm finds the probability sense using a random search. Which is a global optimisation method. (Dan Liu, 2013) derived a method for test paper generation through using the ant colony algorithm. A comparison is also made between using other algorithms such as a random variable algorithm a backtracking algorithm and an artificial intelligence algorithm. Describing the random variable algorithm is that is extracts questions and if they meet certain conditions it then forms a test paper based on these conditions. However, it can fail to meet these requirements. Which in turn offers a poor success rate. The backtracking algorithm works well on small scale generation. Once the scale is largely increased so the time taken to process the generation increases. A new approach would be to compose test papers using the ant colony algorithm as it can search at a far greater speed with and intelligent search.

Chapter 3

Methodology Chapter

3.1 Introduction

Agile is not a methodology but more of an alternative to the existing SDLC Models. To articulate the methods and techniques used in this plan. In figure 3.4 is the outcome after reviewing various SDLC methodologies with reference to tutorialspoint (2016):

- Adoption of the Agile Model.
- Suits the requirements for this project.
- Widely accepted within companies within the IT industry.
- Valuable learning curve in gaining experience with this model.
- Model has the ability to adapt and tailor itself within each increment as the project moves forward.
- Advantageous to the project.

3.1.1 What is Agile?

Agile is an iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end. It works by

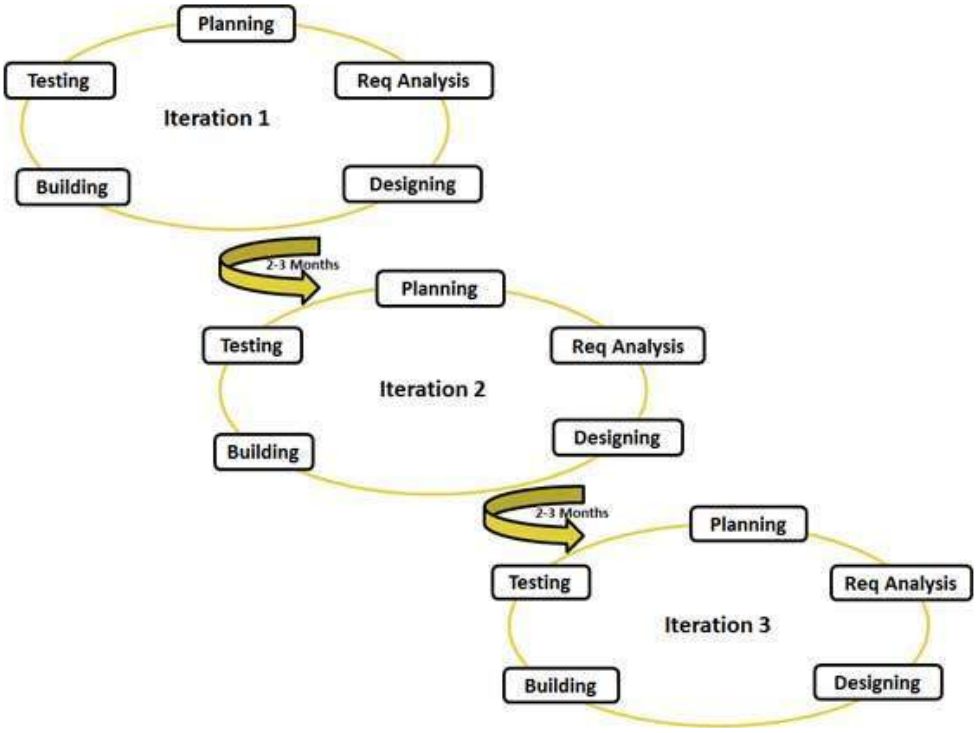

Figure 3.1: Graphical illustration of the Agile Model (tutorialspoint, 2016)

breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short two week cycles called iterations.

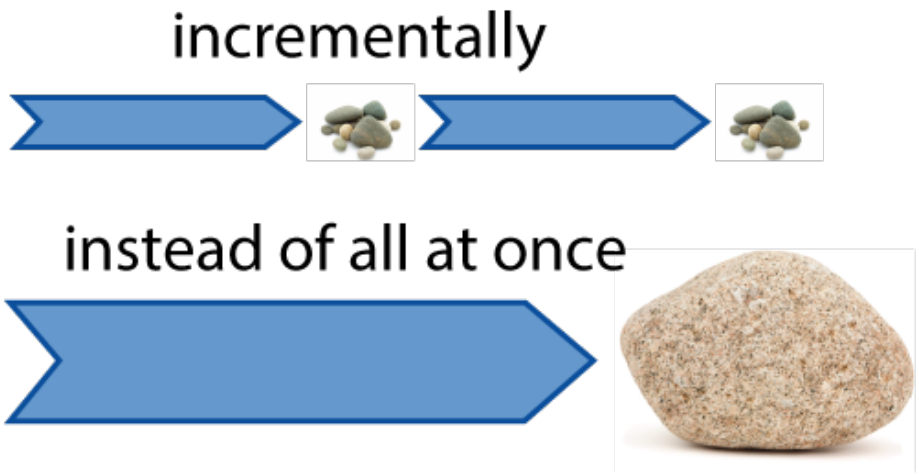

Figure 3.2: Increments of the Agile Model (agile in a nutshell, 2016)

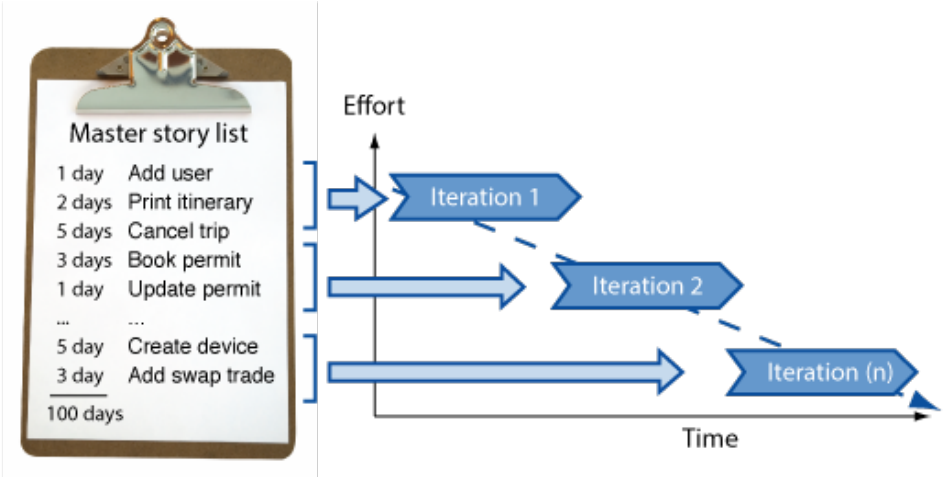

Figure 3.3: Iterations of the Agile Model (agile in a nutshell, 2016)

3.1.2 How does it work?

At its core, Agile does the same thing you and I do when faced with too much to do and not enough time. Then, using Agile estimation techniques, you size your stories relatively

to each other, coming up with a guess as to how long you think each user story will take. Like most lists, there always seems to be more to do than time allows. So you ask your customer to prioritize their list so you get the most important stuff done first, and save the least important for last. Then you start delivering some value. You start at the top. Work your way to the bottom. Building, iterating, and getting feedback from your customer as you go. Then, as you and your customer starting delivering, one of two things is going to happen. You'll discover:

You're going fast enough. All is good. Or, You have too much to do and not enough time.

At this point you have two choices. You can either a) do less and cut scope (recommended). Or you can b) push out the date and ask for more money.

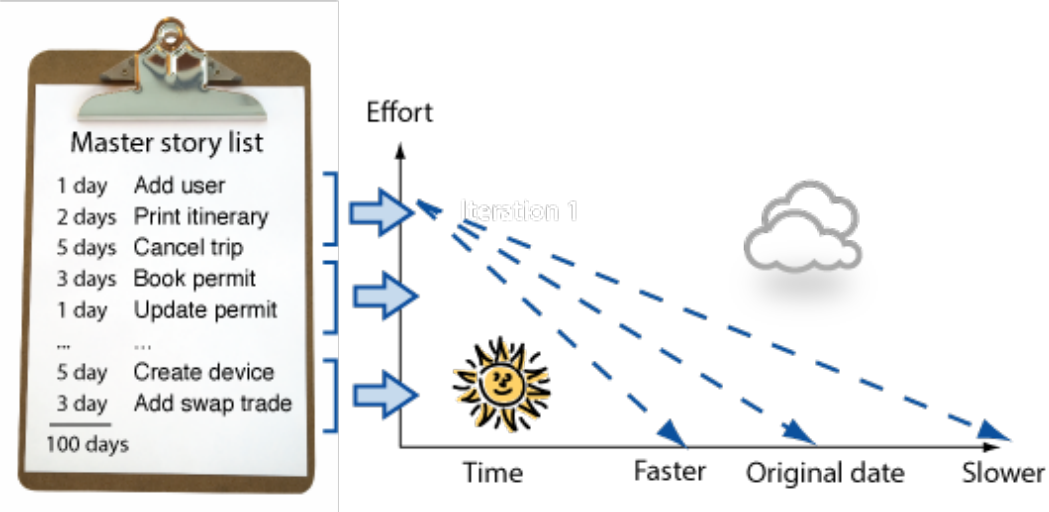

Figure 3.4: Making a list (agile in a nutshell, 2016)

3.1.3 How is it different?

Analysis, design, coding, and testing are continuous activities

You are never done analysis, design, coding and testing on an Agile project. So long as

there are features to build, and the means to deliver them, these activities continue for the duration of the project.

3.1.4 Agile vs Waterfall

Traditional Waterfall treats analysis, design, coding, and testing as discrete phases in a software project. This worked OK when the cost of change was high. But now that it's low it hurts us in a couple of ways. First off, when the project starts to run out of time and money, testing is the only phase left. This means good projects are forced to cut testing short and quality suffers. Secondly, because working software isn't produced until the end of the project, you never really know where you are on a Waterfall project. That last 20% of the project always seems to take 80% of the time.

3.2 Data Collection Methods

3.2.1 Internet Search

Discusses where I obtained my information

3.2.2 Supervisor Input

Discussion on how to store the encrypted data in tables

3.2.3 Journals in Library

Journals which were read and relevant to this project

3.3 Method of Analysis

3.3.1 Formulation of Where to Start

Researching the title of the project

3.3.2 Early System Implementation

This project was linked to an assignment which helped in my progression

3.3.3 Review of Literature

Undertaking a literature review aided with making comparisons of other systems produced

3.4 Summary

Summary with all terms discussed within the chapter

Chapter 4

Implementation - ”Building the solution”

4.1 Introduction

This chapter is a walkthrough of the steps which describe the construction of the application.

4.2 Terminal

4.2.1 Command Line Instructions

As this application was centred towards security and being secure as the data it would hold would need to be kept safe. Therefore it made sense to focus on a good login with authentication and authorisation. To start working with Symfony, one needs to setup the Symfony environment through an installation process before Symfony applications can be created. Instructions for this can be found on the SensioLabs Symfony website. Navigating to the Documentation page. In there can be found Chapter 1 which has the Setup instructions under the Get Started dropdown menu. They explain the different ways for installing and setting up the Symfony framework for both Mac OS and Windows. Along with some troubleshooting ideas if there are any problems with the installation. This application was built on a Mac OS so the instructions would vary slightly due to the command line instructions used. Using the installer made it easy to create this application with Symfony and only needed to be done once as it was installed globally on the machine.

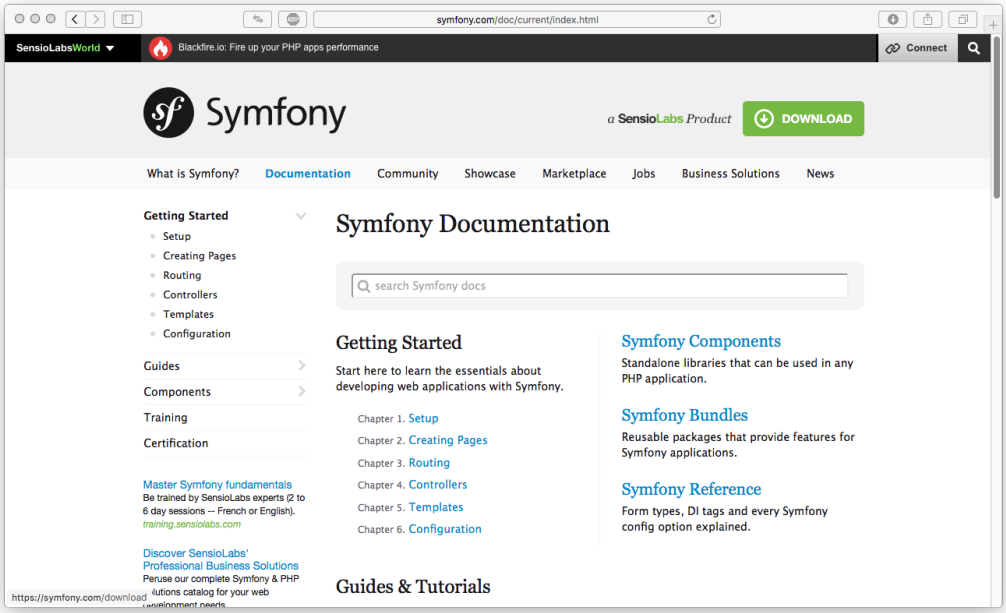

Figure 4.1: Symfony Documentation

Figure 4.2: Symfony Installation Setup

With this completed moving into the directory or environment that the application will live which was the desktop. A final command in the terminal window was issued. This time starting with symfony new and by giving a project a name of choice thereafter. In this case the name COMPH4021-Project was used. The project was based on the current version of Symfony which is version 3.2.8. However, other versions can be specified after the project name in the terminal window. Once this part was completed. All required components were downloaded into a project folder with the name of which was given. The components are a set of files and directories which form the web application which use the Symfony libraries. The installer also carries out a check to make sure all requirements are met. If requirements are not all met a list is generated which provides the changes that are needed. In this case no changes needed to be made.

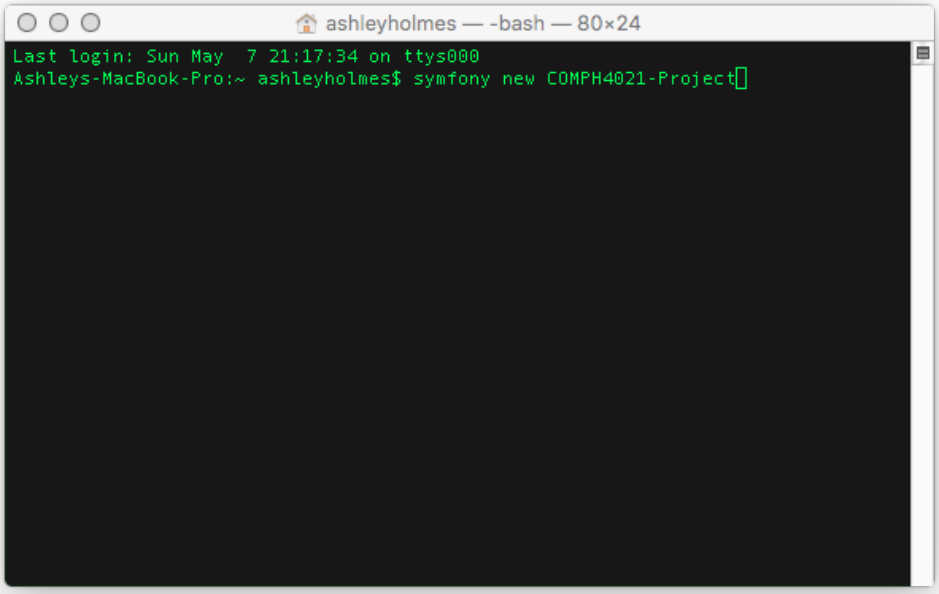

Figure 4.3: Symfony Application Setup

The below figure 4.4 displays the command issued to download the project folder and following that in figure 4.5 one can see that the project is being prepared and where it will be stored. In this case it was stored in /Users/ashleyholmes/Desktop/COMPH4021-Project.

Figure 4.4: Terminal Window Top

Figure 4.5: Terminal Window Bottom

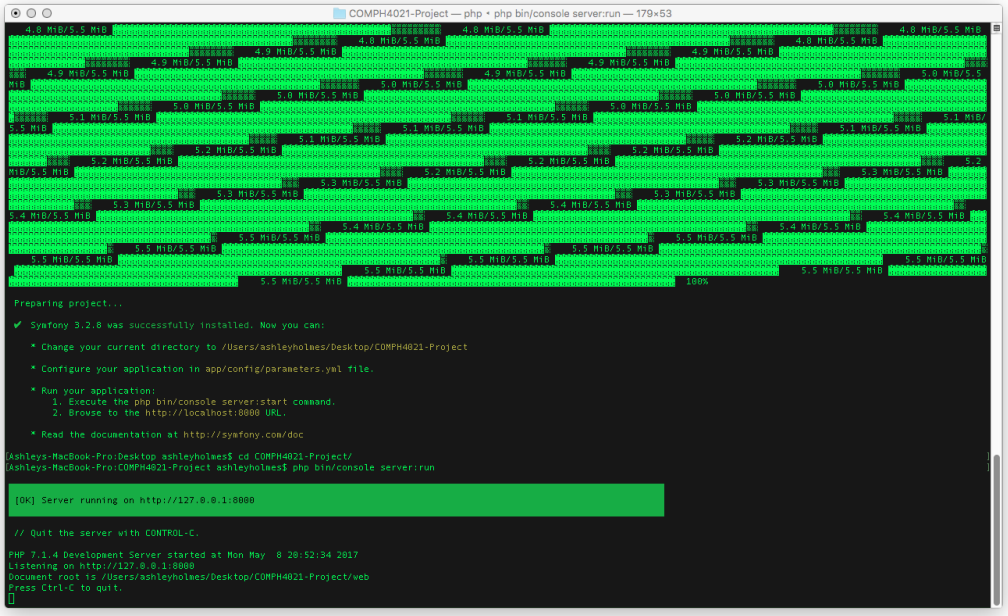

Figure 4.6: Php Server Run

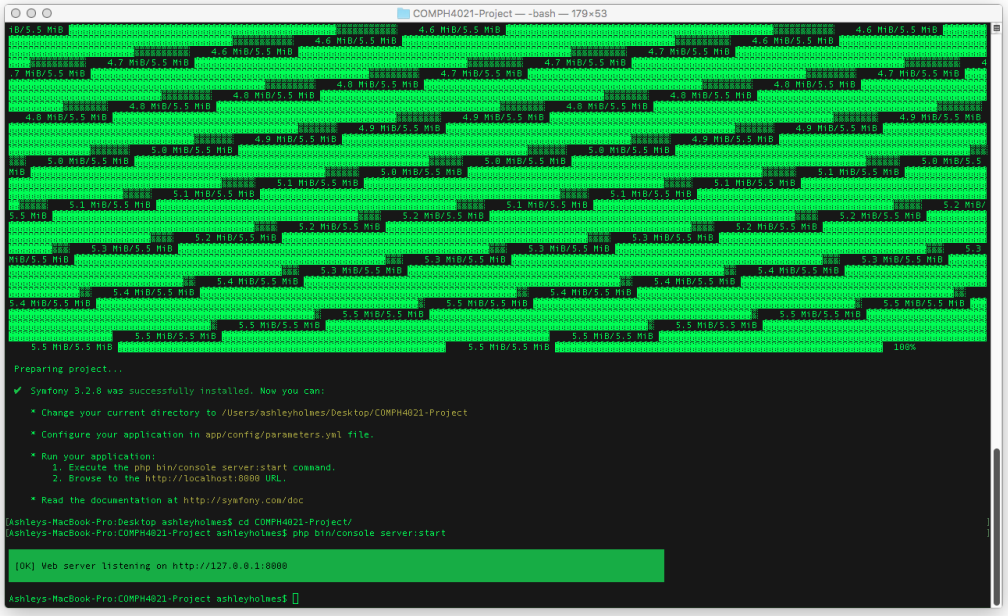

Figure 4.7: Php Server Start

The next step would be to change directory into the COMPH4021-Project directory as this is where the built in Php server needs to be run from. NGINX and Apache may be used as alternatives however, since the Php server is built in. It makes development much easier. Executing the following command php bin/console server:run starts the server. However, once issuing this command the open terminal window would now need to remain out of use while the server is running. One can use a separate terminal window or open a new tab to issue any addition commands which are needed or make use of the PhpStorm terminal window. To run other processes in the background, issuing a php bin/console server:start will make it possible to execute commands in the same window which was done here. The difference can be seen in figure 4.6 and figure 4.7.

4.3 Browser

4.3.1 Deploying the Application

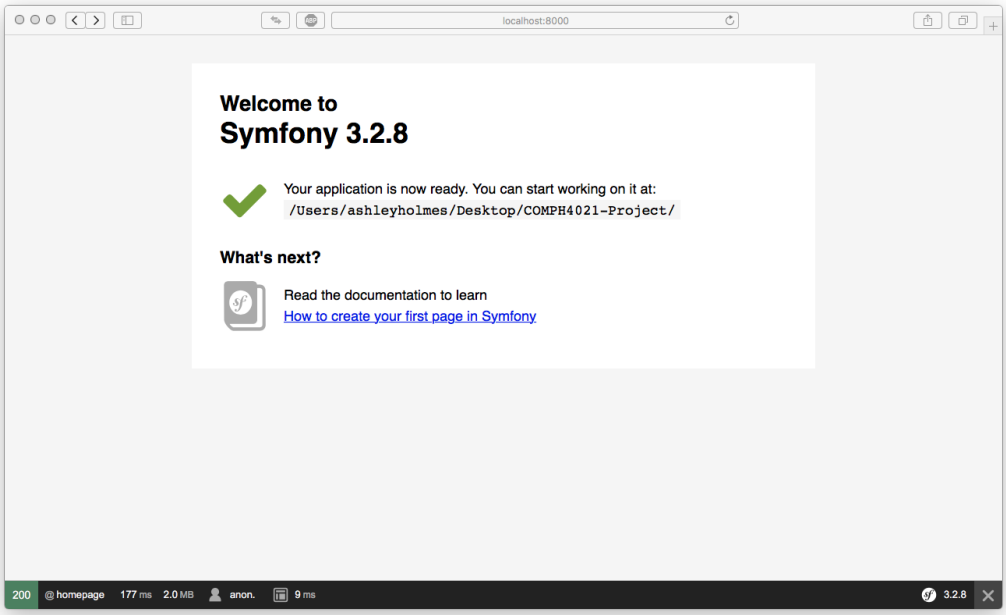

Figure 4.8: Symfony Browser

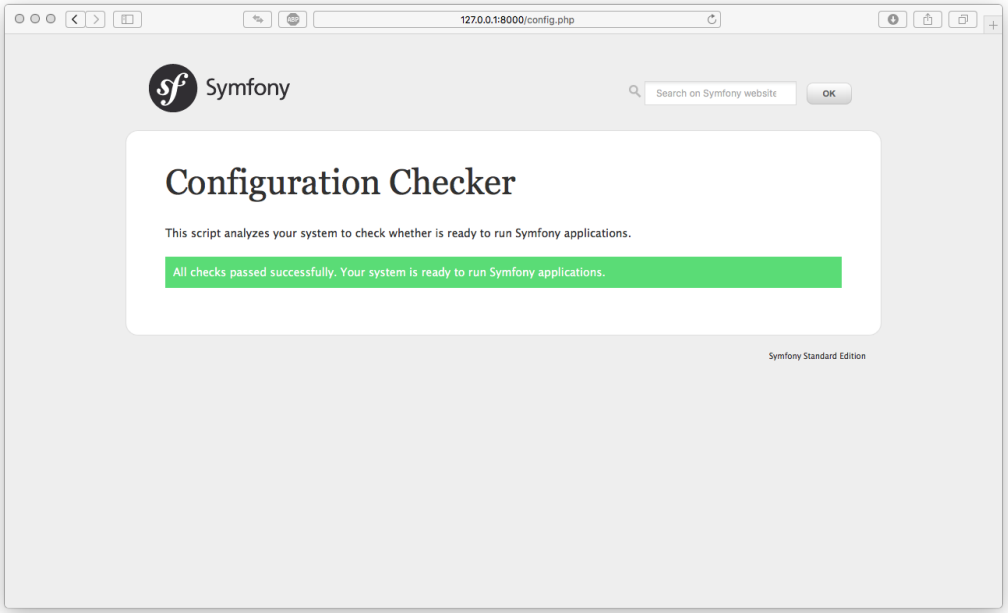

Figure 4.9: Configuration Checker

Now that the configuration phase has been completed one now navigates to the browser and using the URL http://localhost:8000 as shown in the terminal window in figure 4.5 under the Run your application heading. This brings up the following page in figure 4.8. It is being executed by the Symfony framework from the files inside the project folder. The code is depicted in figure 4.13. In the bottom of the window is the web debug toolbar which is in a maximised position and can be minimised by clicking on the X in the right hand corner. This often offers better visibility when developing. If the mouse is used to hover over the toolbar it will display information such as routing, controllers which were executed, time it took to load the page, which way the user is authenticated on the page and more debugging information and a link to the resources and documentation as shown in figure 4.12. Clicking on the icons will give much more information. The URL http://127.0.0.1:8000/config.php would show the user the instructions needed for further configuration. This is reference to figure 4.9.

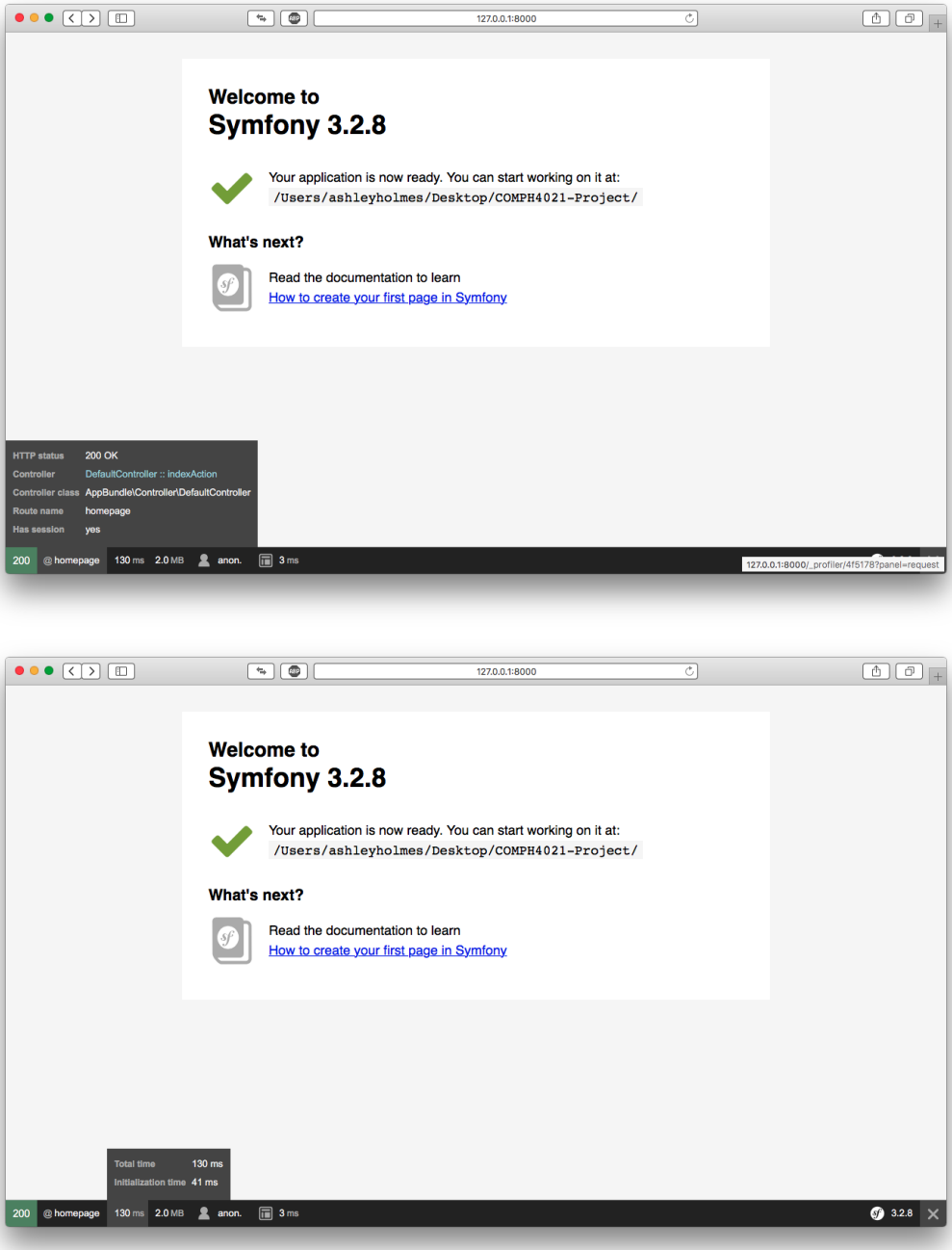

Figure 4.10: Web Debug Toolbar and Profiler Extended

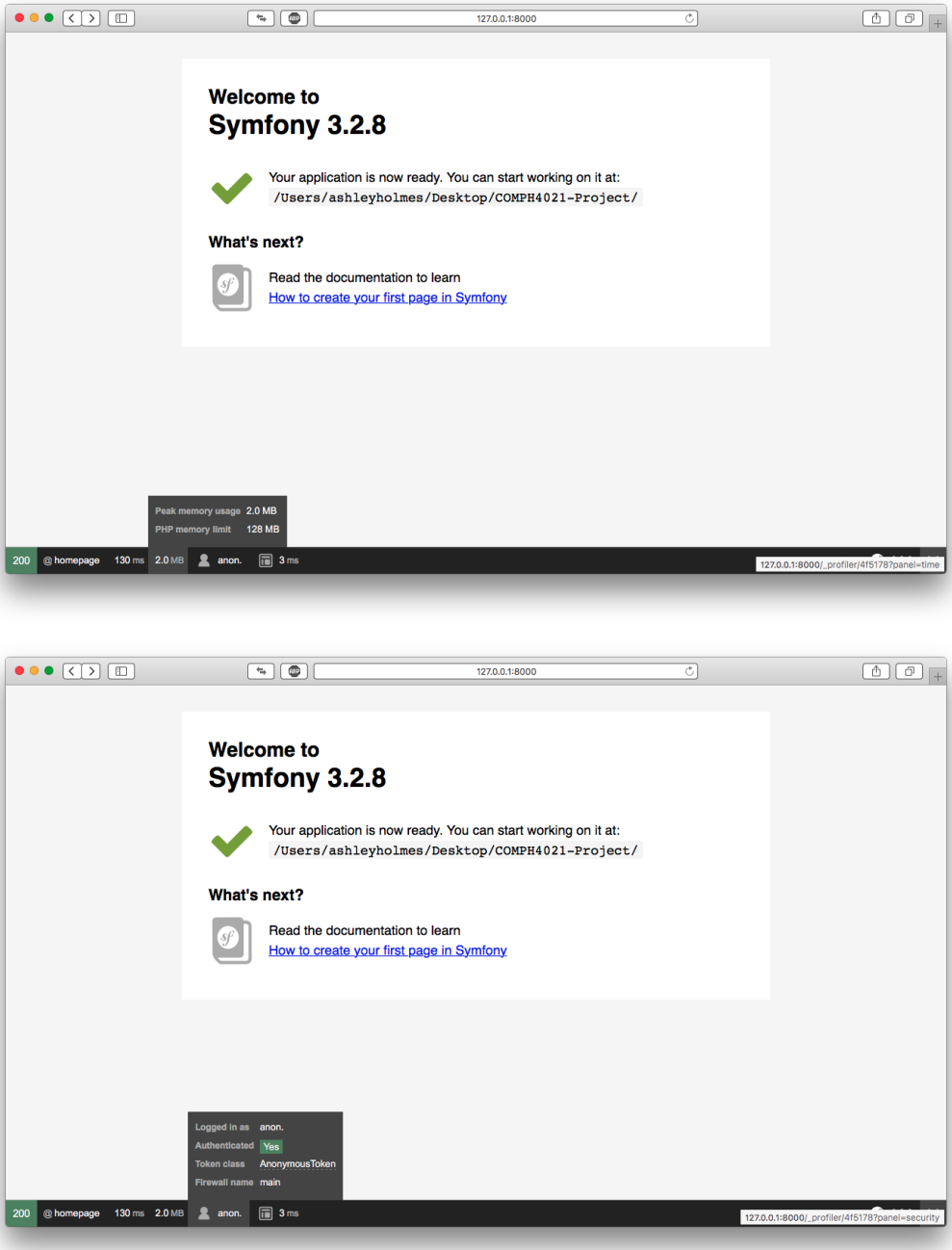

Figure 4.11: Web Debug Toolbar and Profiler Extended

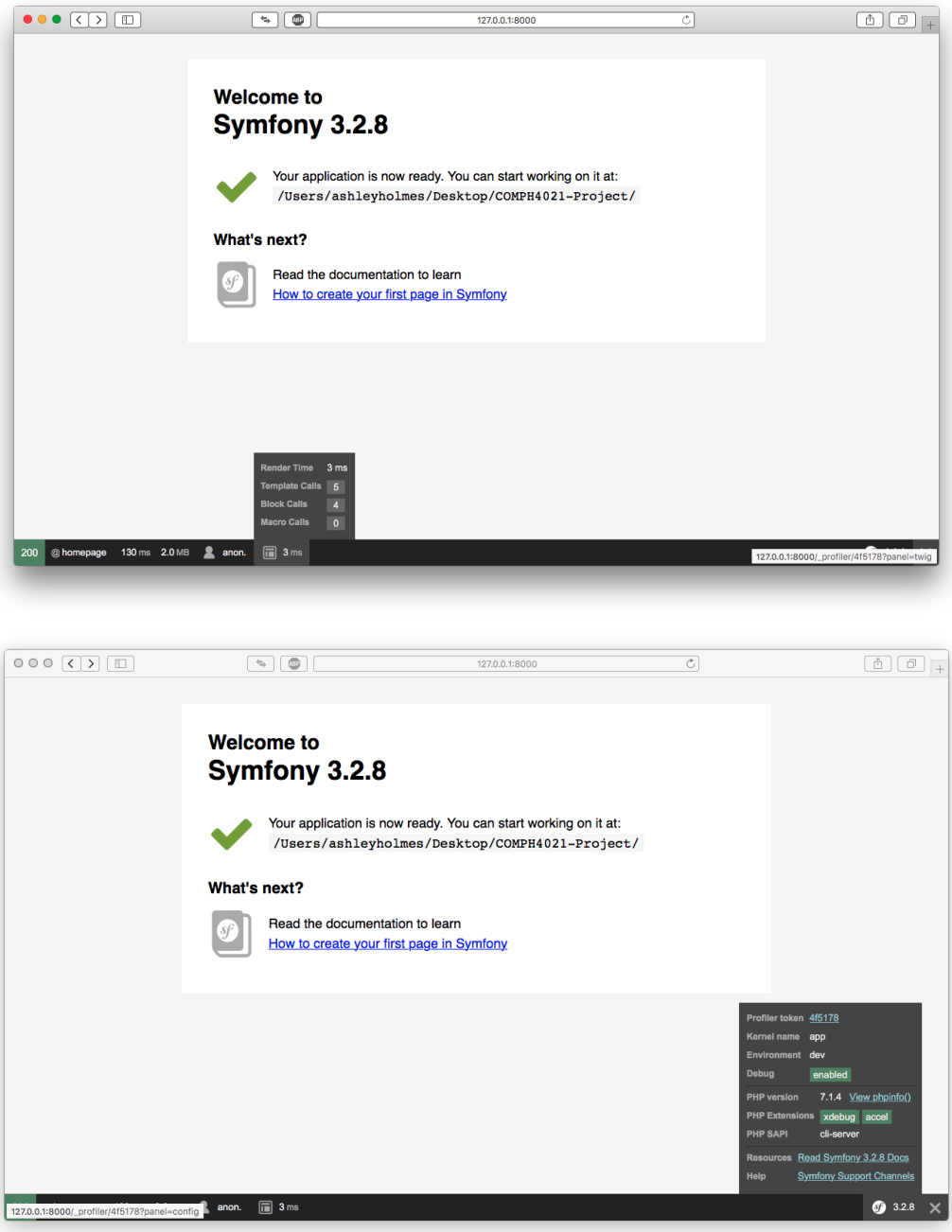

Figure 4.12: Web Debug Toolbar and Profiler Extended

Figure 4.13: Web Debug Toolbar and Profiler

4.4 IDE

4.4.1 PhpStorm IDE for PHP

Most of the files live in src/AppBundle. Looking at the controller called DefaultController in the below figure 4.14. This controller class defines what is seen in figure 4.8. It renders the Symfony Welcome Page. Take note of the @Route(”/”, name=”homepage”) annotation on line 12. It matches the route in figure 4.12. The next process was to remove the extraneous code which was not needed in the Twig template and the DefaultController. In the template itself, it was using some variables which was passed in line 18 of figure 4.14 in the DefaultController.php class. With this removed, Bootstrap was enabled as a CDN and added to the templates to provide consistency across all pages. In order to use Bootstrap, it needed to be downloaded from the Bootstrap website. From the Getting started section there are examples which can be chosen as a starting template to use as a foundation for all templates. This was added to the base.html.twig and then modified to make it more specific to the project. Twig uses inheritance and all templates extend from the base.html.twig. In addition to this a small amount of custom CSS was added to achieve the result in figure 4.18

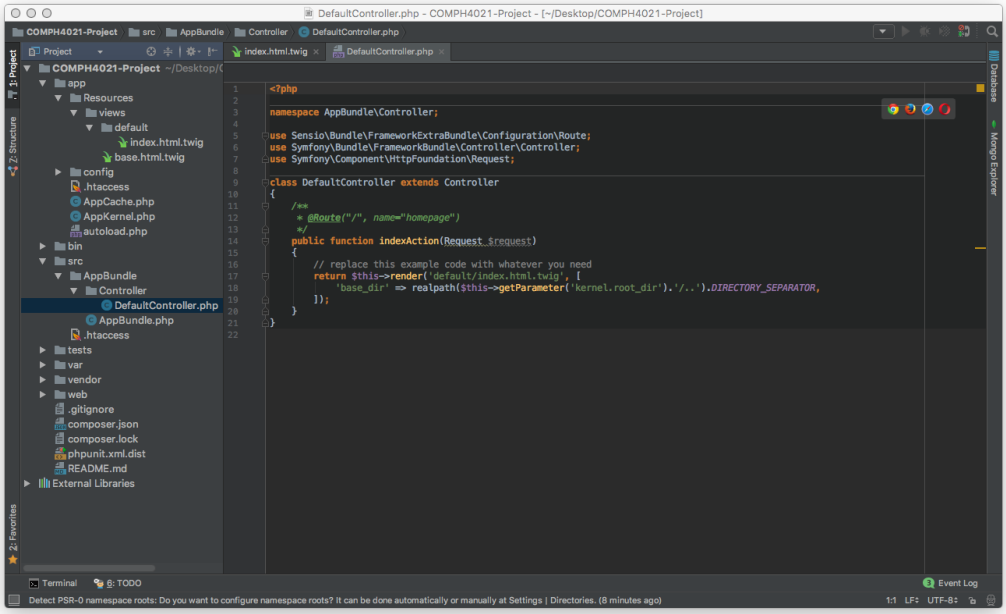

Figure 4.14: DefaultController

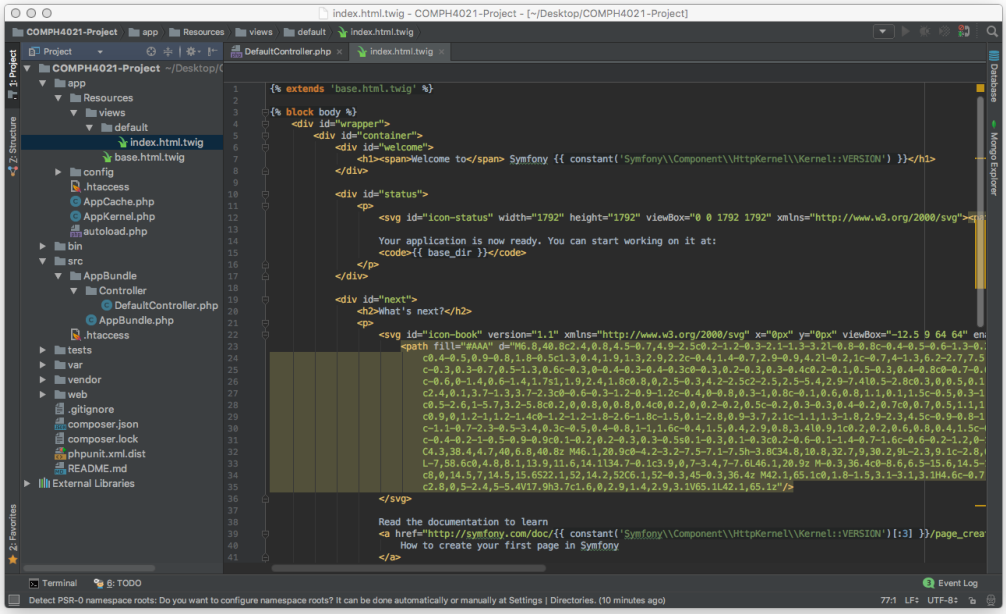

Figure 4.15: Twig template in IDE

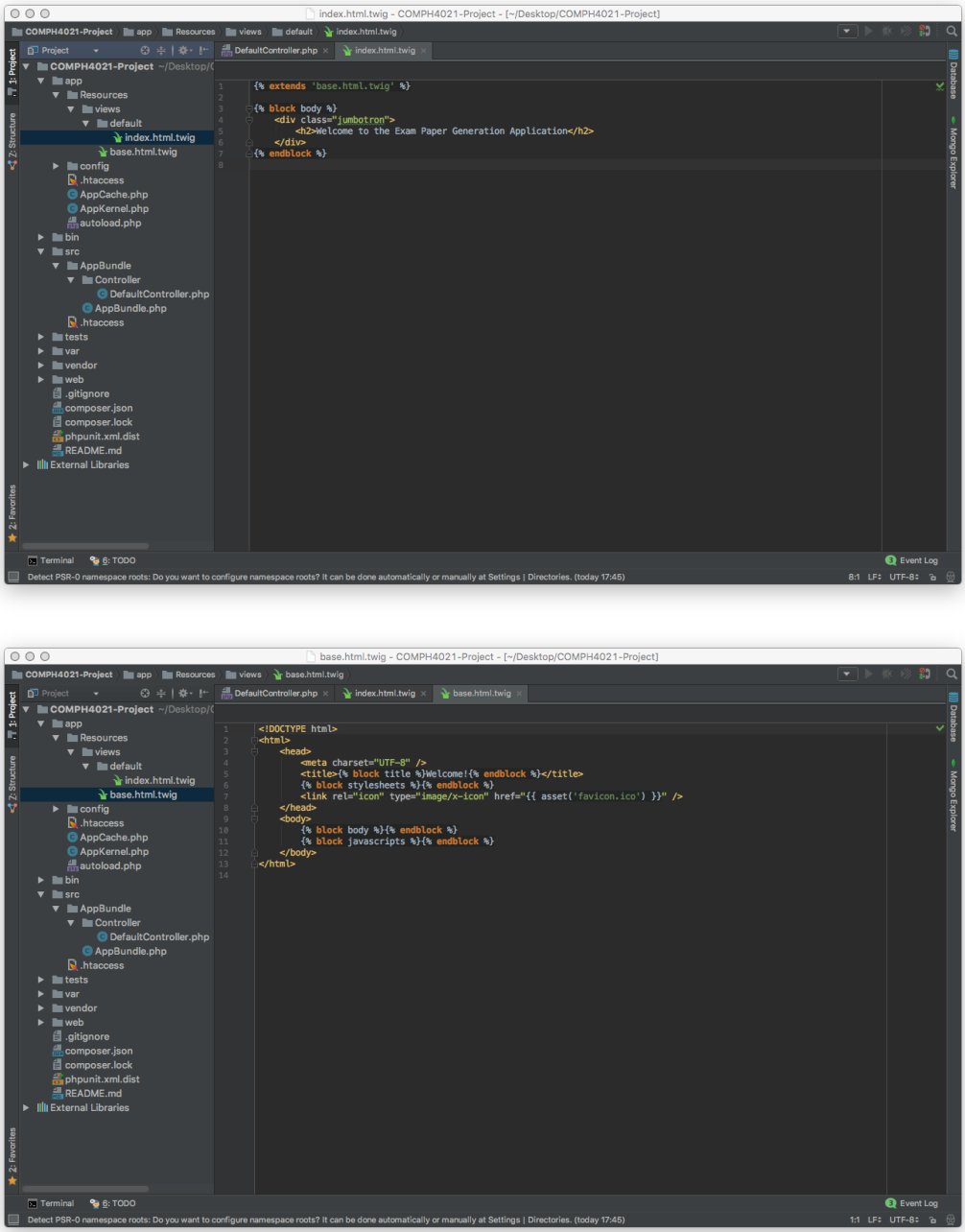

Figure 4.16: Adding Bootstrap

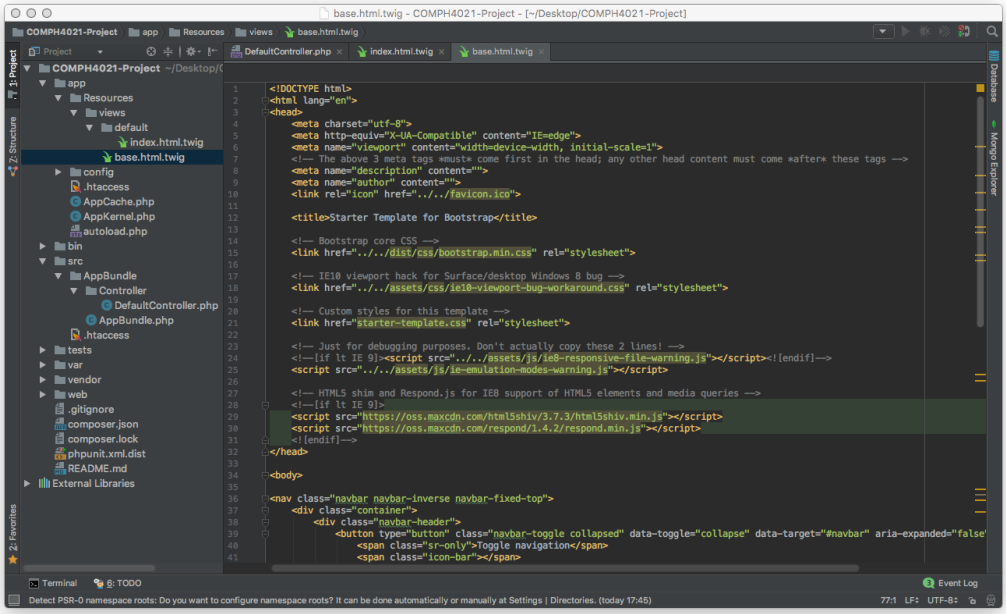

Figure 4.17: Adding Bootstrap

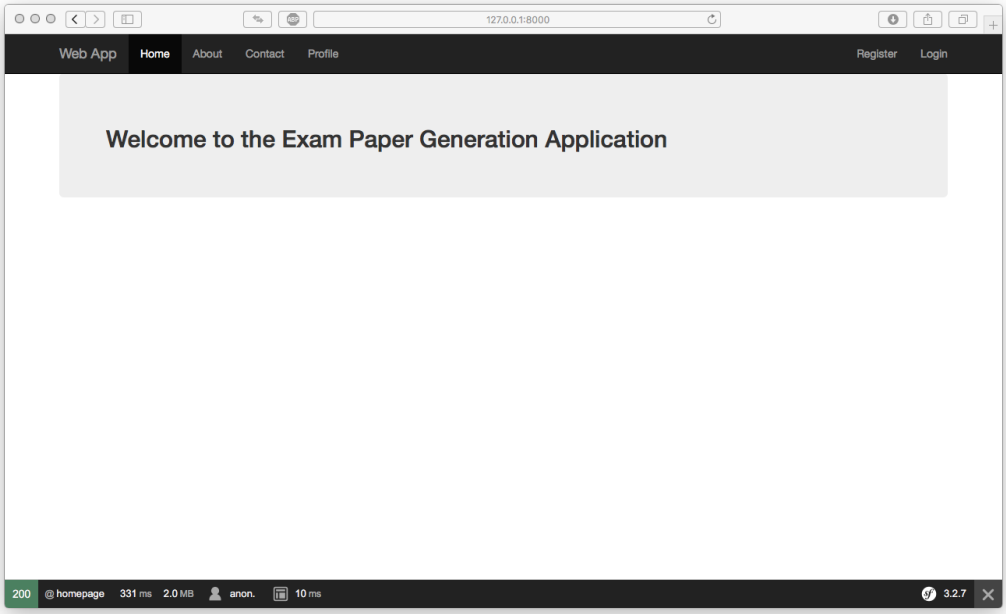

Figure 4.18: Home Page

4.5 Database

4.5.1 Creation of User Entity and CRUD

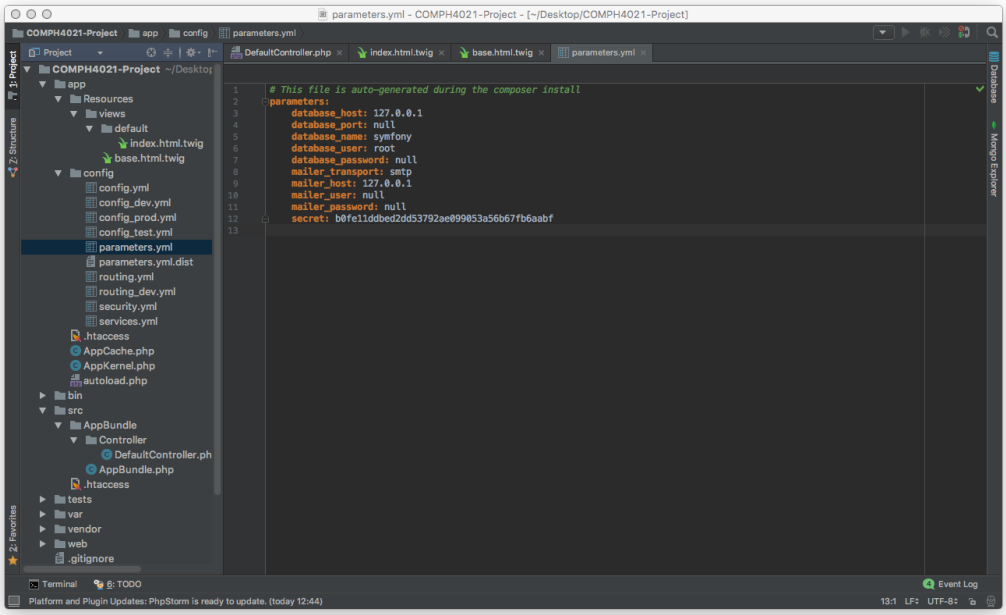

Figure 4.19: Parameters Yaml

This section covers Object Relational Mapping or ORM. With using an ORM, the concept of working with a database becomes easier as it is an easy switch from one to another such as Mongo to SQL or Postgres. Not having to worry about the actual database. Symfony takes care of the abstraction. In addition the ORM generates the CRUD off the entity which is made and makes everything work. The first thing that needs to be done is make a change to the parameters.yml which can be seen in figure 4.19. The change was to the database name from symfony in line 5 to a name of choice. In this case the name project was used. The database port changed to 8889 and the database password changed to root to correspond with the database login details. With having done that an external MySQL server was needed to make a communication to the database itself. The port number is the port on the local MySQL server. In this case MAMP was used in figure 4.20.

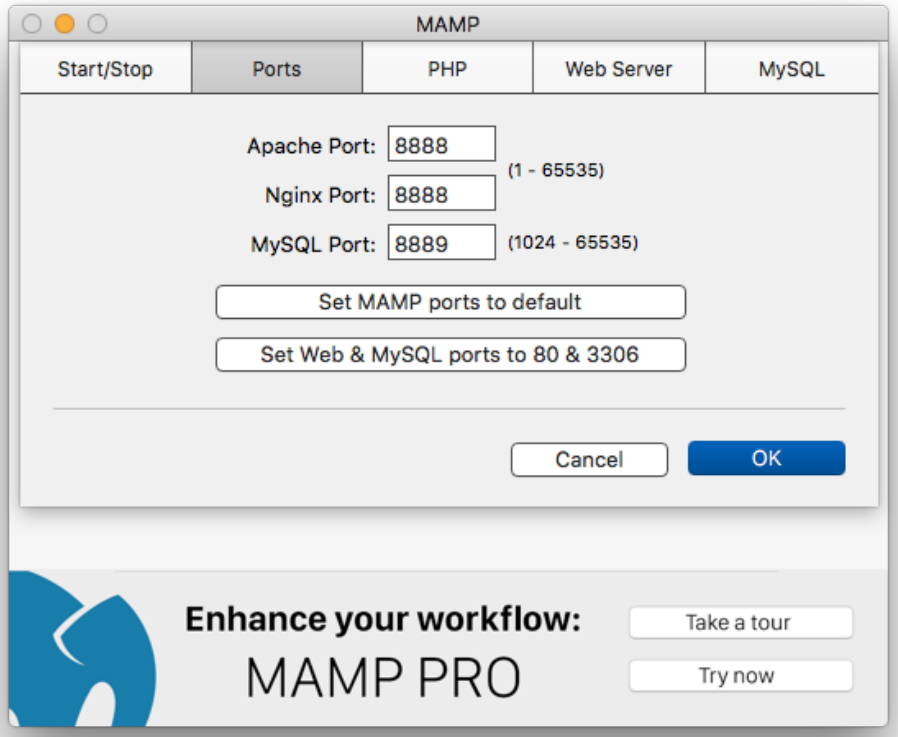

Figure 4.20: Mamp Ports

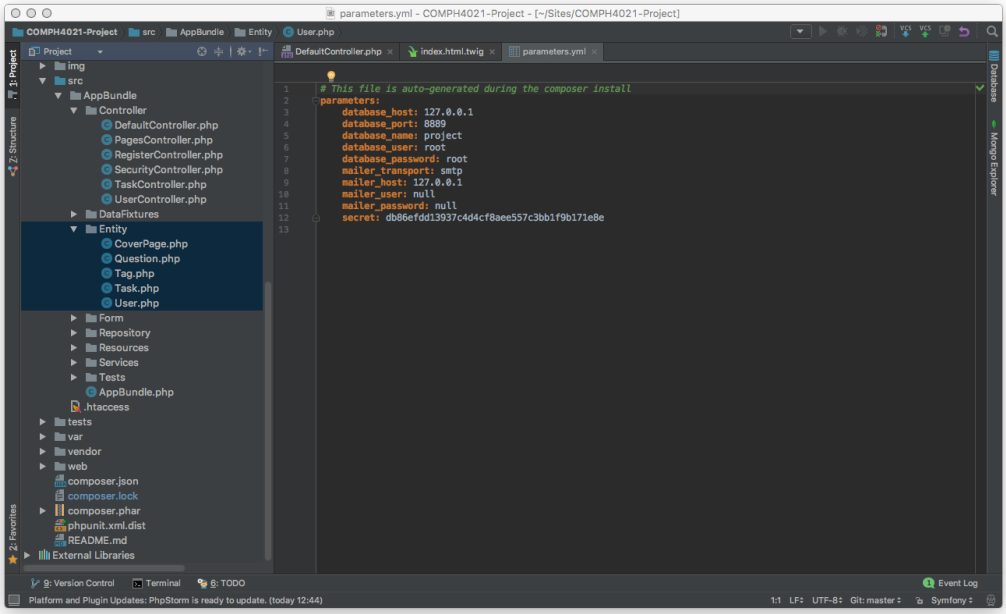

Figure 4.21: Entities

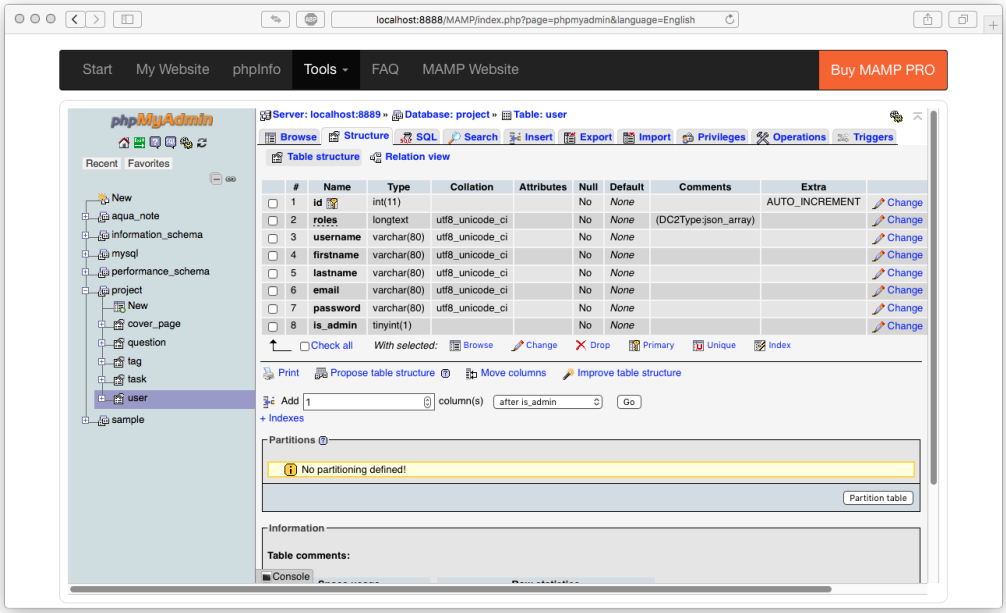

Figure 4.22: User Table

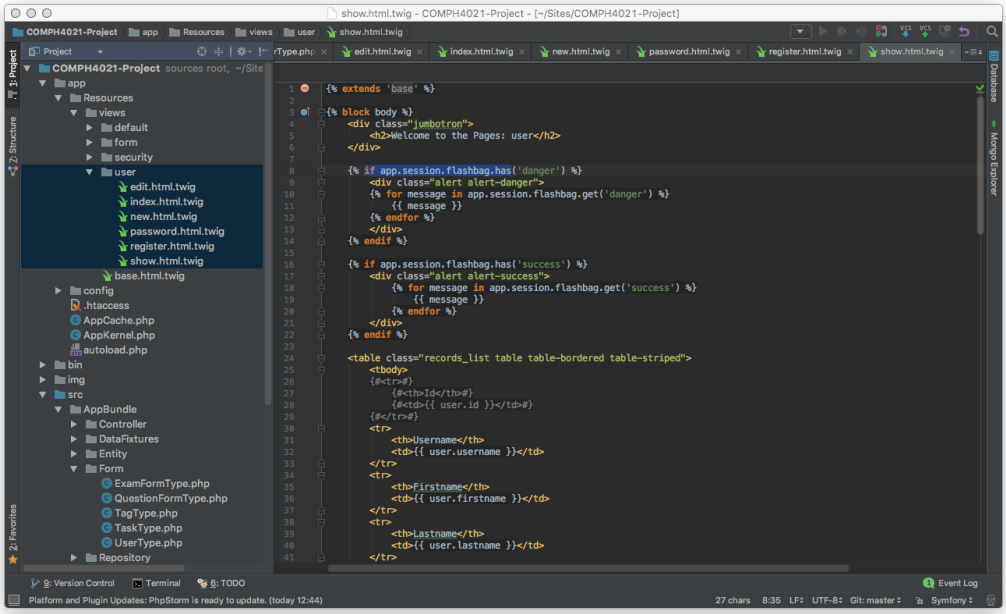

Figure 4.23: Twig templates

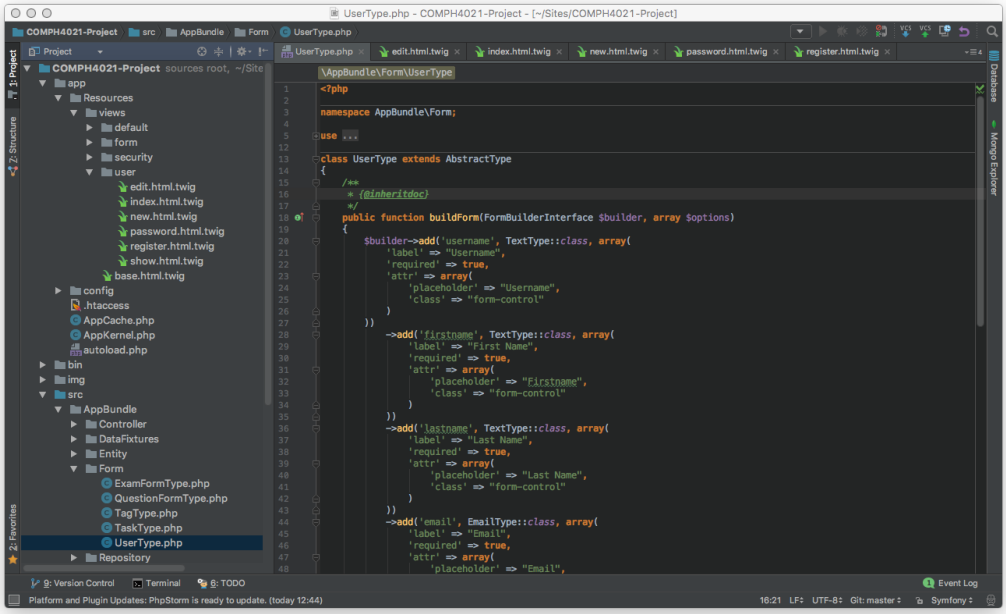

Figure 4.24: Form Type

Issuing a command in the terminal. php bin/console doctrine:database:create will create a database with the name project which was added to the parameters.yml file. Creating the entities is done in the same manor. php bin/console doctrine:generate:entity brings up a wizard which asks where to put the entity. Giving the command AppBundle:User. The entities were all placed in a directory called Entity which resides in src/AppBundle. All the work was was done is in AppBundle. The fields in the database are then added along with a repository class which will be discussed later. There is now a username, firstname, lastname, email and password with type String and field length of 80 characters. An example of this is in figure 4.22. Symfony created two different files. The first is the user entity which is the model which is used to base database manipulation, object validation or form validation. The second thing which was done was to create the schema which is done in the terminal with the command of php bin/console doctrine:schema:create. This creates a table structure ready for use off of the user object. To create the CRUD in the terminal it was php bin/console generate:doctrine:crud. The wizard asks where the entity would be found and based off. As discussed it is in AppBundle:User and giving write actions which allows for the update and delete. All the templates were created and added to the Resources directory which are used to create the users in figure 4.23 also a class called UserType.php which is a FormType and is what all the forms are based off in figure 4.24. With this section completed a bit of functional testing was done to make sure that users could be added to the database, edited and removed.

4.6 Twig

4.6.1 Templates are fine tuned

The templates which are created by Symfony was raw html with no styling. Each form had a basic layout. They were ugly. This is why form theming which is discussed later in the chapter and styling needed to be added to make them look as they do in the below figures. Styling included:

- Addition of placeholders.

- Adding a striped table.
- Button styling.
- Removal of the unordered lists.
- Adding column spanning.
- Changing links to be buttons.
- The bootstrap classes were added.
- Button styling.

Some of the data was adjusted which would have been displayed such as the id and passwords which were removed as they should not be visible from the client side.

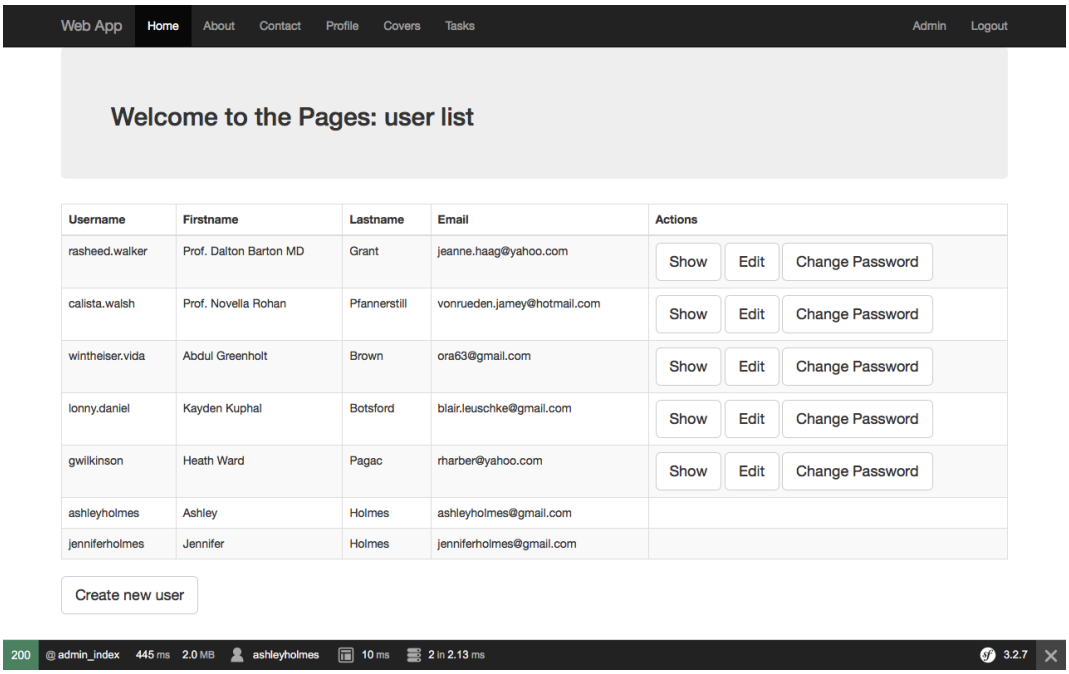

Figure 4.25: index.html.twig

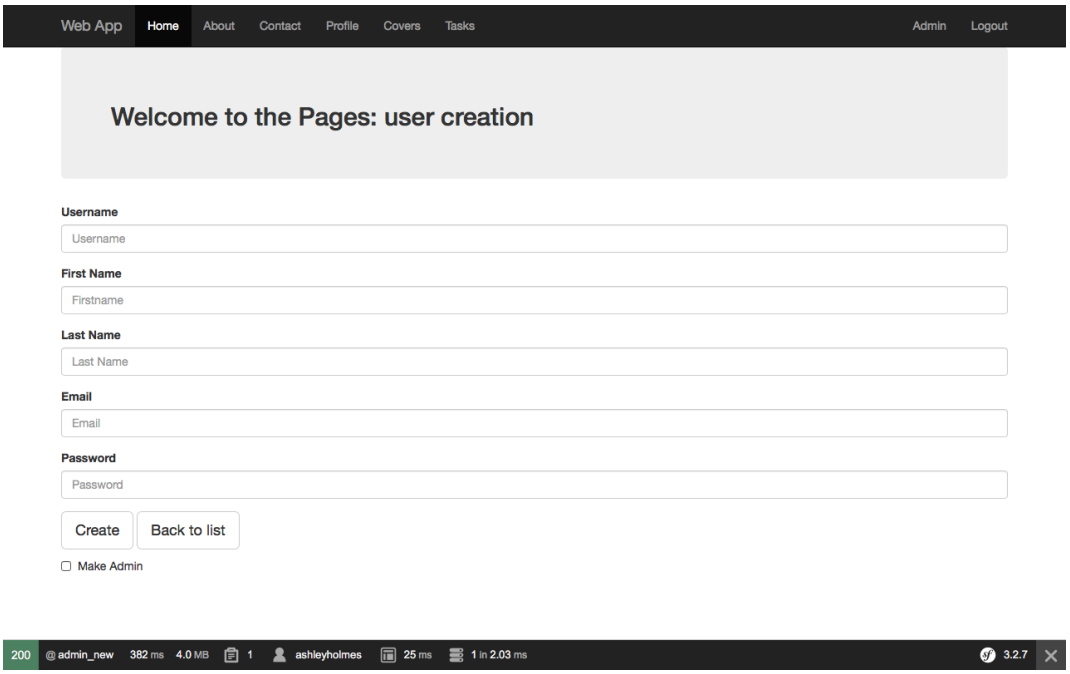

Figure 4.26: new.html.twig

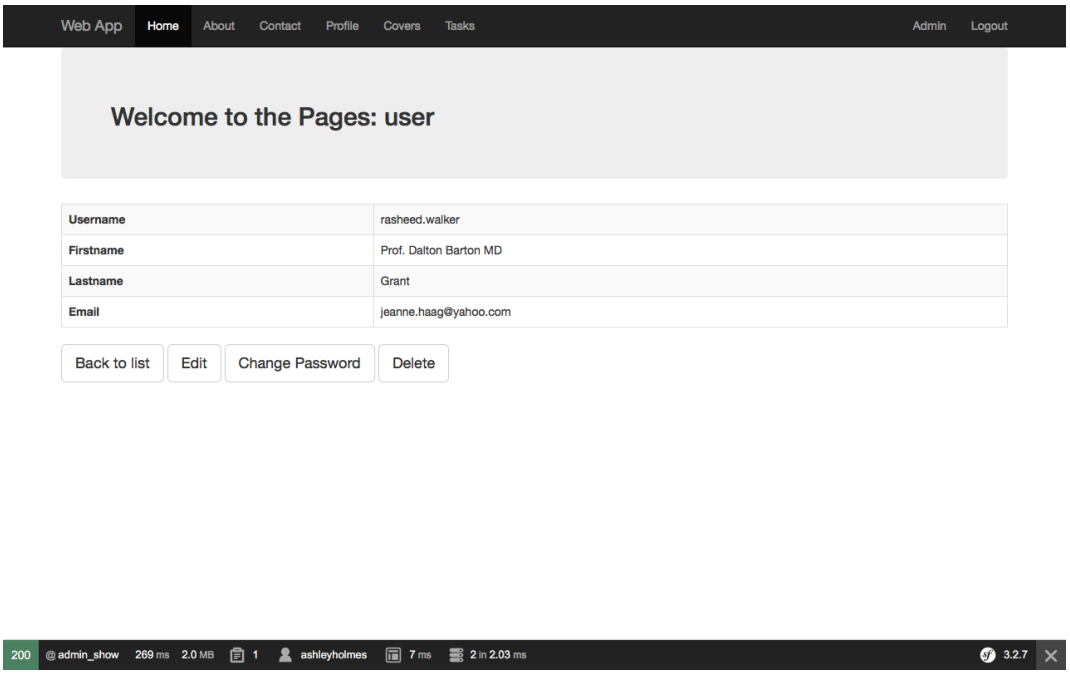

Figure 4.27: show.html.twig

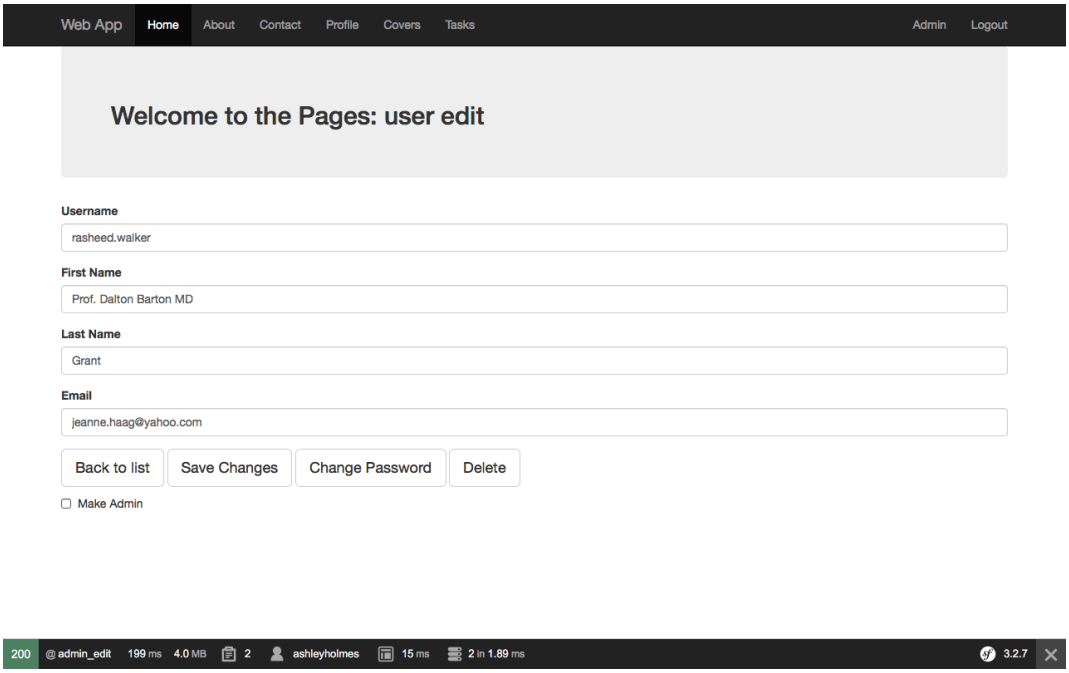

Figure 4.28: edit.html.twig

4.7 Forms

4.7.1 Understanding FormTypes

The FormType works hand in hand with the user object which is created and thus is called the UserType which can be found in src/AppBundle/Form/UserType.php. This is what Symfony created when going through the process of creating CRUD. The FormType is used to take control of the input tags and labels. As mentioned in the previous Twig section with regards to adding the placeholders. This is actually done in the UserType.php class including the CSS classes. When the user clicks on the Create new user button. Symfony extracts information from that FormType and it builds the form. In figure 4.29 one can see that the add method takes up to three arguments. If only one argument is added, Symfony will make the best guess about what is being done. If the second argument is passed in, the second argument is the type of input that is being provide. It could take an Entity or several other arguments. However it does help Symfony make a better decision about what is being done. The third

argument is where one can explicitly say what the options are such as input tags a class as mentioned before. When an array of options are provided Symfony stops guessing what is being done and it takes explicit direction. This is why an array of options are given to explicitly say everything. From line 20 to 25 there is an array of options such as:

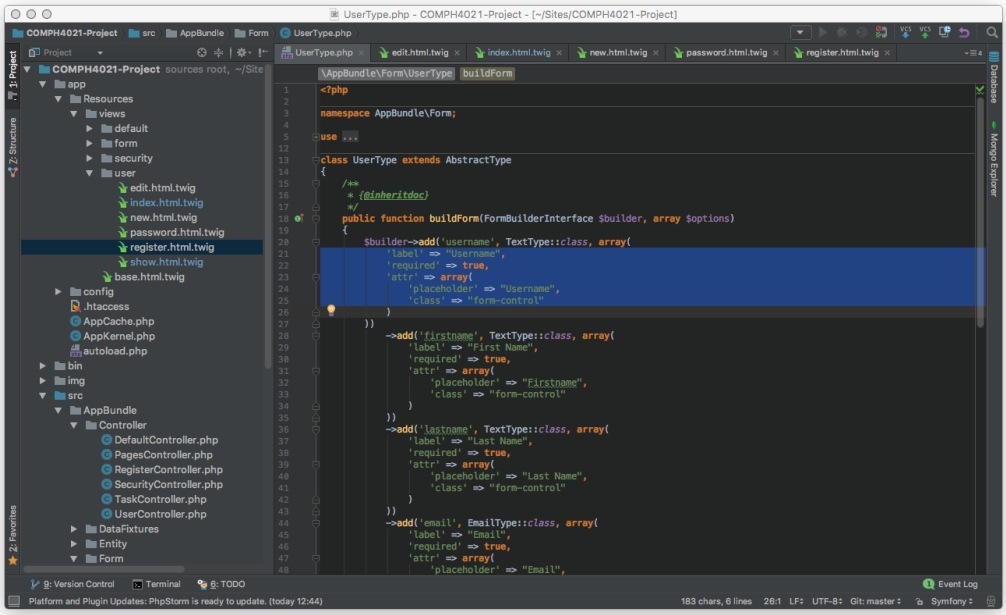

Figure 4.29: UserType.php

- The input label - is expressly put in with the form rendering where in the Twig template figure 4.30 line 12, 17, 22, 27, and 32 is the form label.
- Attribute required is true forces the user to complete this section of the form.
- The array of other attributes are which go inside the input tag.
- Placeholder and class are added here.
- Class is a Bootstrap class of form control.

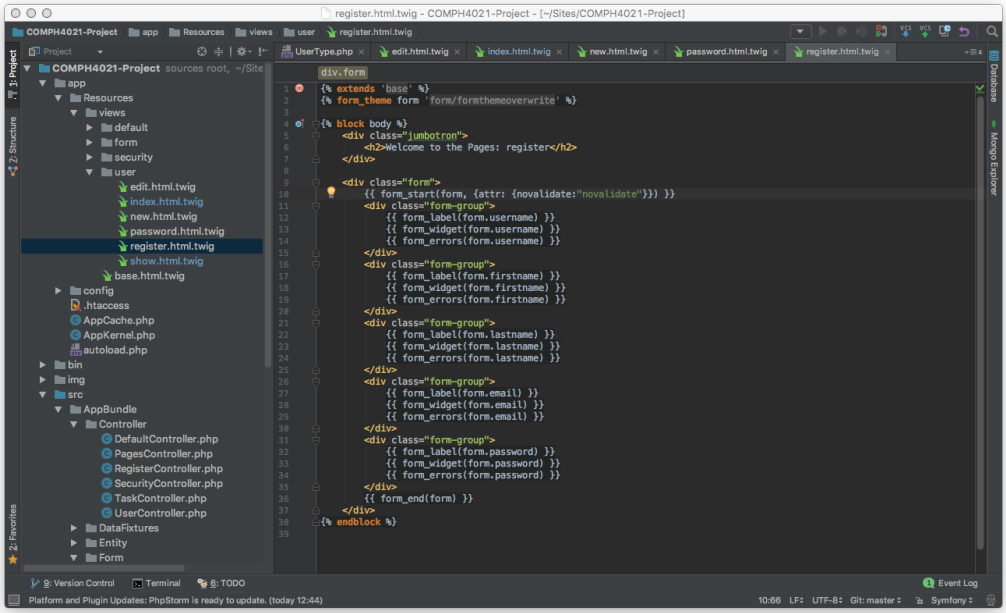

Figure 4.30: Form Label

4.8 Validation

4.8.1 Validation of the Forms

At this stage without populating the form and by using the create button to create a new user in the form there was HTML5 validation. A notification message would pop up with an instruction such as, ”Please fill out this field.” In order to validate the form or objects on the server side. The form validation needs to be turned off. Figure 4.31 shows how this was done.

Line 10 is being passed an instance of the FormType in the form start method. Adding arguments to that such as attr: novalidate:”novalidate”. With this attribute added there was no server validation. This was performed on the object which is the entity User.php class by adding a use statement such as in line 7 of figure 4.32.

A use statement in Symfony is a class. With the use statement added it was possible to assert constraints on top of the properties for each of the form fields with reference to figure

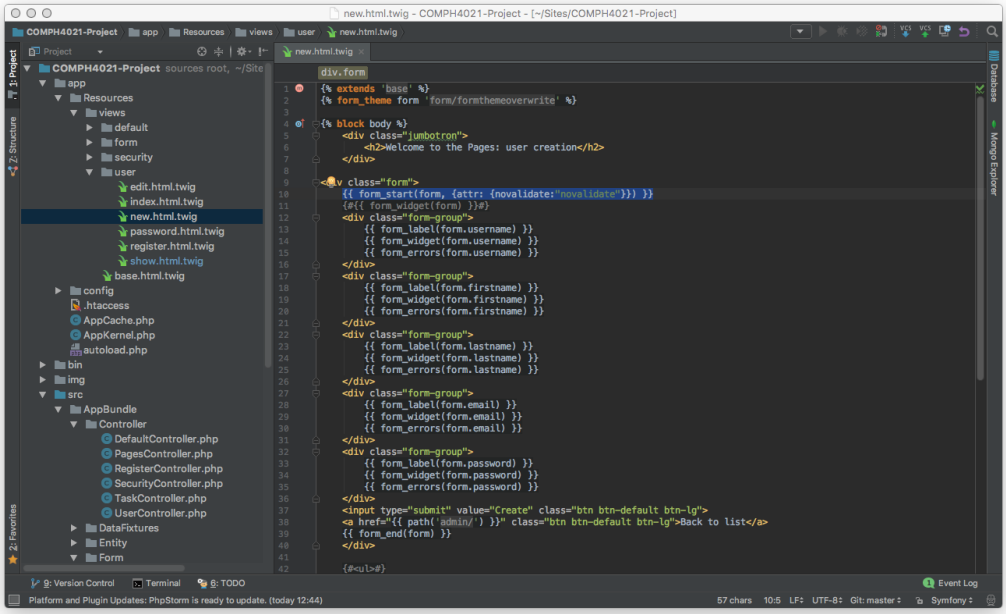

Figure 4.31: Form Validation

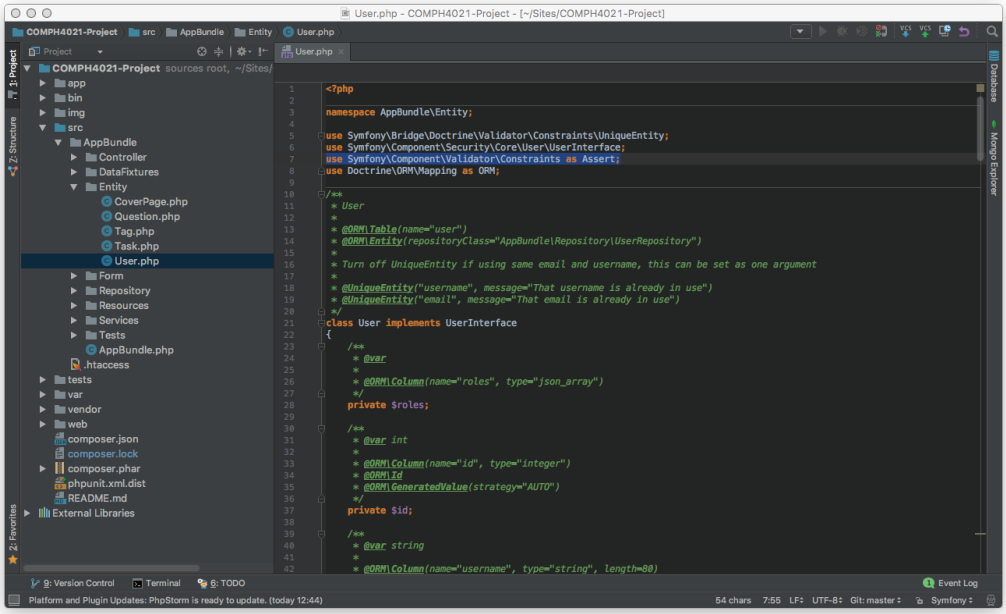

Figure 4.32: Entity Validation

4.33.

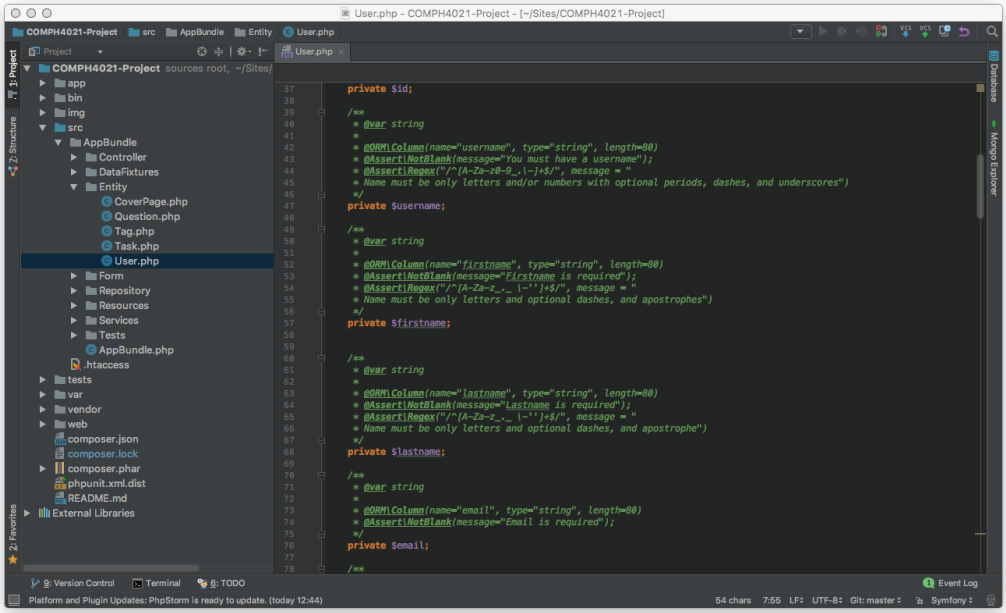

Figure 4.33: Constraints

Functional testing was done at this point in order to make sure everything worked. Another feature which was added was having a unique username and a unique email. This was achieved with yet another use statement which can be found in line 5 of figure 4.32. In the class declaration there are two arguments passed for each property in line 18 and line 19. The property name that needs to be checked against uniqueness. And a message was also passed in, incase the test failed. The same was created for username. It is possible to set both the username and the email as double arguments for one unique entity call however, this will allow the uniqueness of the email to get through as long as the uniqueness of the username is not there. This is why they were passed in separately. Functional testing was performed in order to confirm the changes which were made worked appropriately.

4.9 Theming

4.9.1 Form Theming

Form theming is a concept which made all the forms look appealing such as the error messages and buttons. Things to remember are, never to adjust the core of Symfony. Instead it is overridden by doing things like telling Symfony where things are that need to be overridden. To proceed with this the error code can be found in the core of Symfony. Figure 4.34 shows the error message which was overridden.

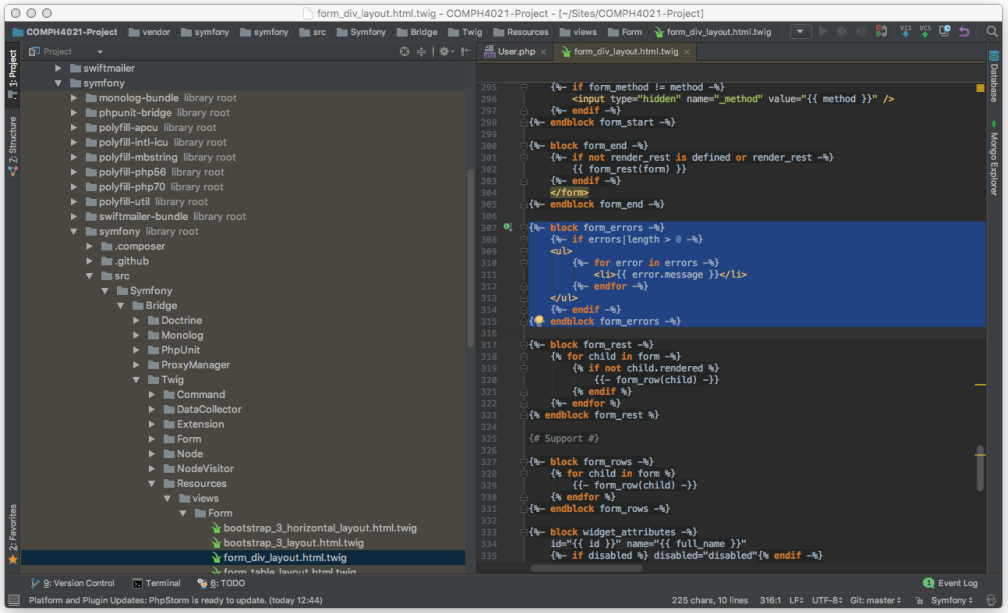

Figure 4.34: Form Errors

It was overridden in the Twig template figure 4.35 with a Bootstrap dismissible alert placed inside the Symfony error message. The files which use this Twig template are:

- edit.html.twig
- new.html.twig
- password.html.twig

- register.html.twig

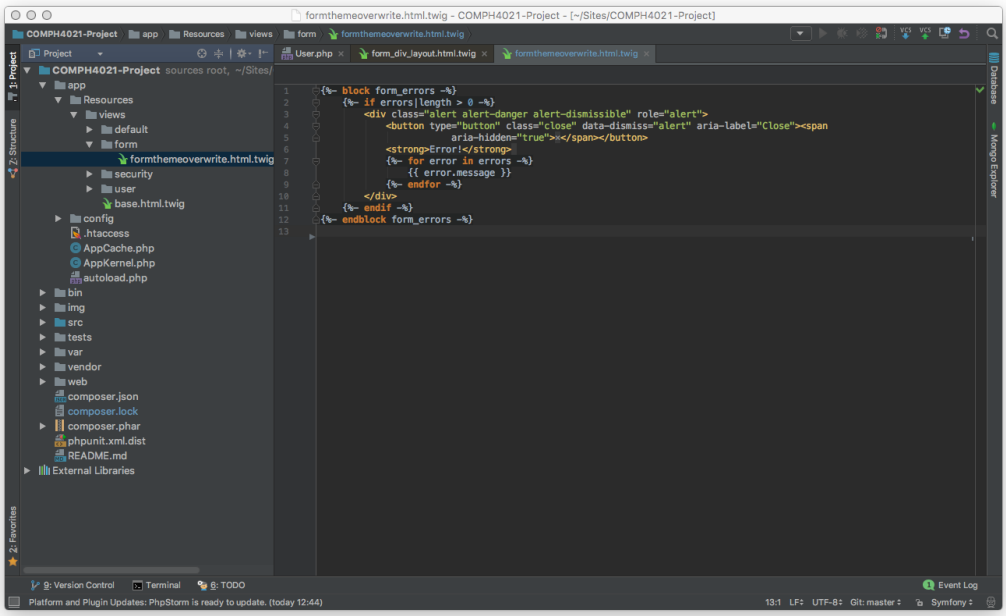

Figure 4.35: Form Theming

The buttons were given their own theming and implemented as follow in figure 4.36.

4.10 Fixtures

4.10.1 Fixtures with Faker

Fixtures are used to add data into a database in a controlled manner for the purpose of testing or for the initial data which is required for the application to run. The four actions which were necessary for this was as follows:

- Use Composer to download the appropriate dependencies.
- Adjust the AppKernel.php to use the dependencies.
- Write the fixture.

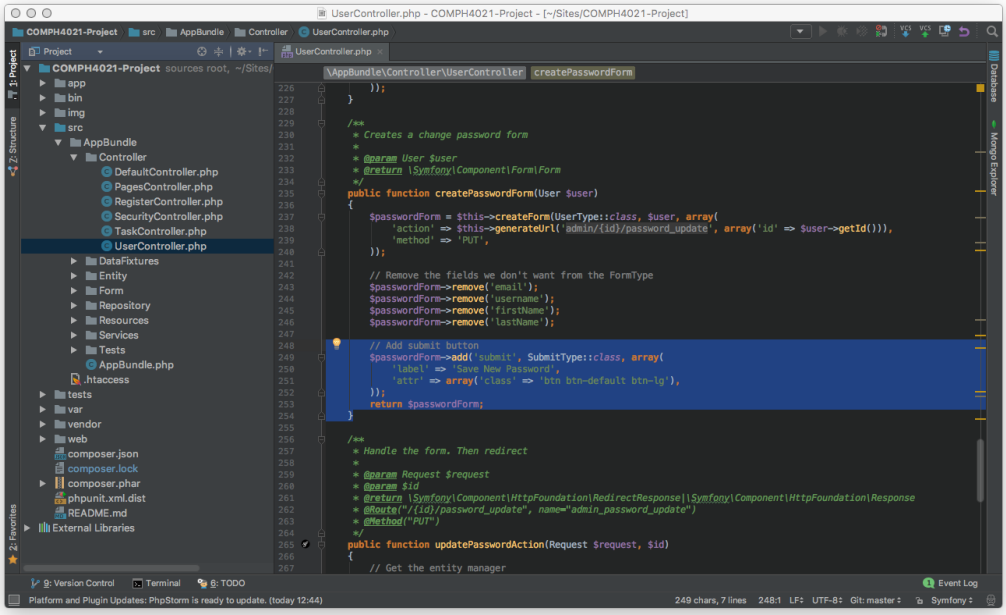

Figure 4.36: Button Theming

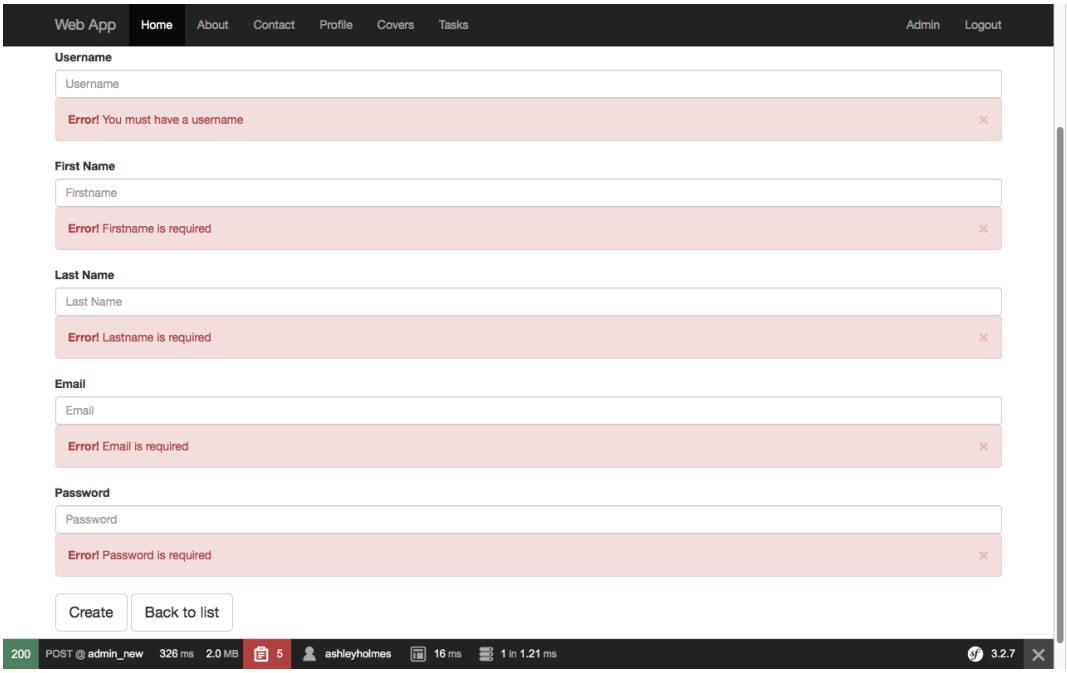

Figure 4.37: Form Theming Result

- Use the fixture in the console.

Every good application needs some way to test the data which is being worked on. This is why fixtures is a good place to start. Before this can be done Composer needs to be installed on the computer and it needs to be installed globally. Instructions for this can be found on the Symfony website. Composer is a dependency manager and it is used to require a bundle from Packagist. Packagist is the main Composer repository. It links packages with Composer and shows Composer where to get the code from. With that completed a terminal command may be issued: composer require –dev doctrine/doctrine-fixtures-bundle with reference to figure 4.38.

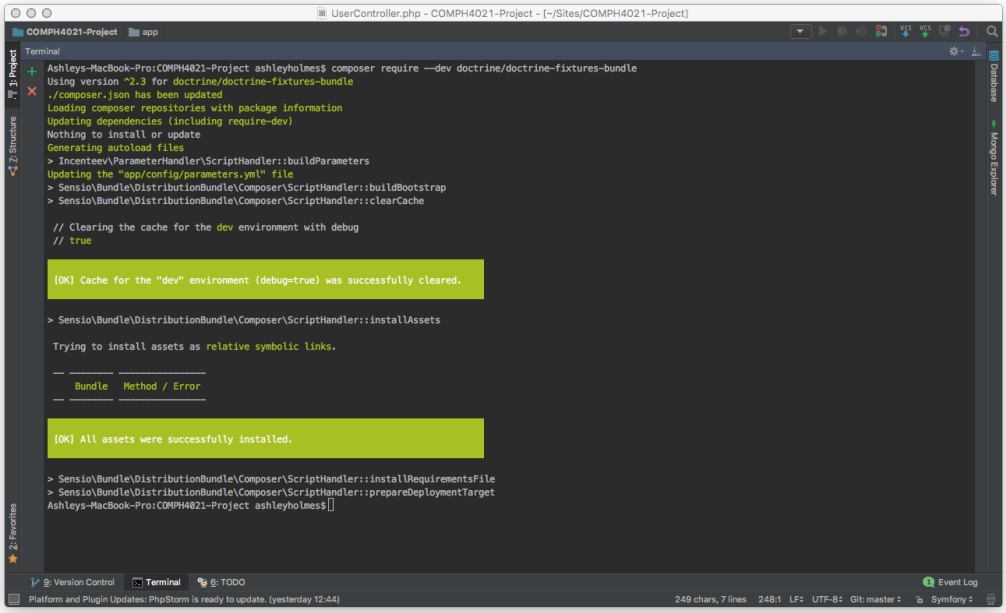

Figure 4.38: Fixtures

The following line of code figure 4.39 was added to AppKernel in the bundles array of which activates the bundle. It was now possible to build a load in fixtures class. In AppBundle a directory is created with the name DataFixtures and a class called PopulateUserTable.php. In figure 4.40

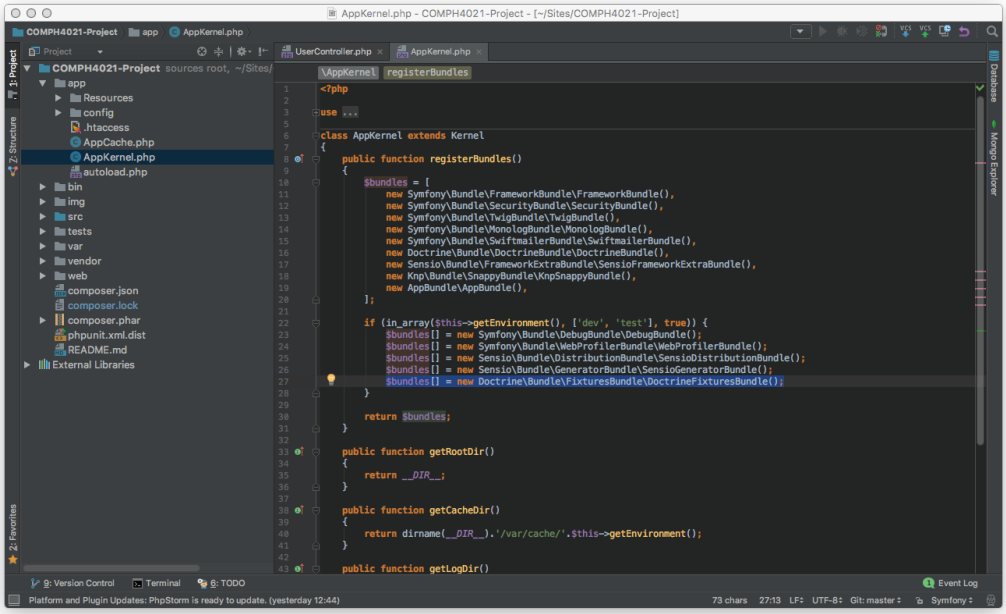

Figure 4.39: Bundles

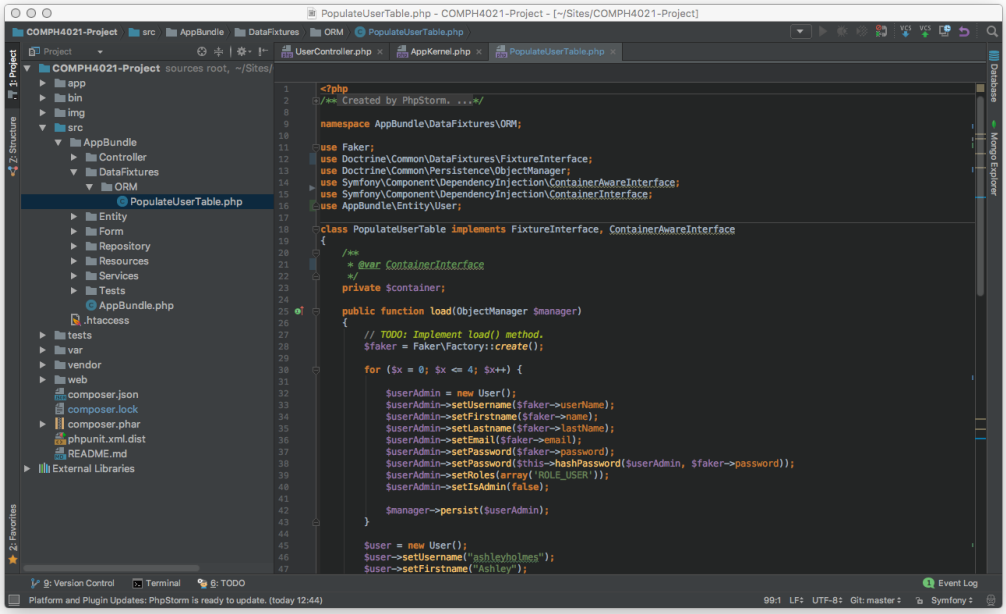

Figure 4.40: Faker

on line 9 there is a namespace for the file which is also a bundle much like a directory. It becomes a bundle when a bundle class is added to it. Adding a namespace to a class is organising files from one directory, into a sub directories. The PopulateUserTable class lives in a directory called ORM which lives in a directory called DataFixtures which lives in AppBundle. Which is essentially a folder hierarchy. Each one will be unique as each class name is different. Use statements are included from line 11 to line 16. The one on line 15 has a method which is required as it implements the FixtureInterface. Line 13 is the ObjectManager which enables the manipulation of the EntityManager in order to persist the object to the database which is the use statement in line 12. Line 25 has a public function called load and is what is required by the FixtureInterface and will bring in the object to the function which is also called dependancy injection. Line 11 is the Faker use statement. This is a PHP Library which generates fake data and can be seen in the admin index page. Line 42 shows how the ObjectManager persists it to the User object and line 69 in figure 4.41 which writes the data to the database.

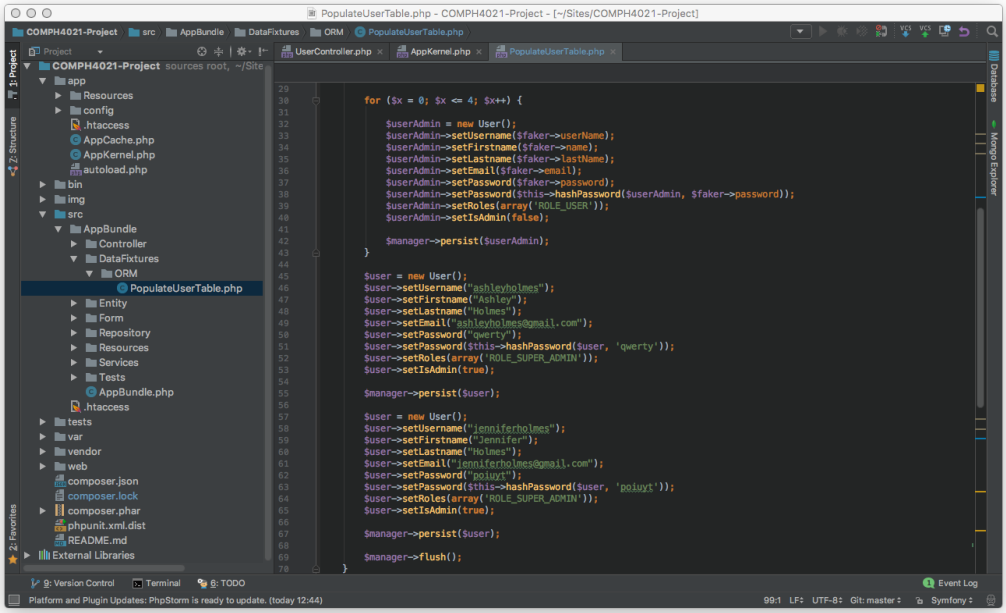

Figure 4.41: Object Manager Flush

The line in the console window is what is used to purge what is currently in the database

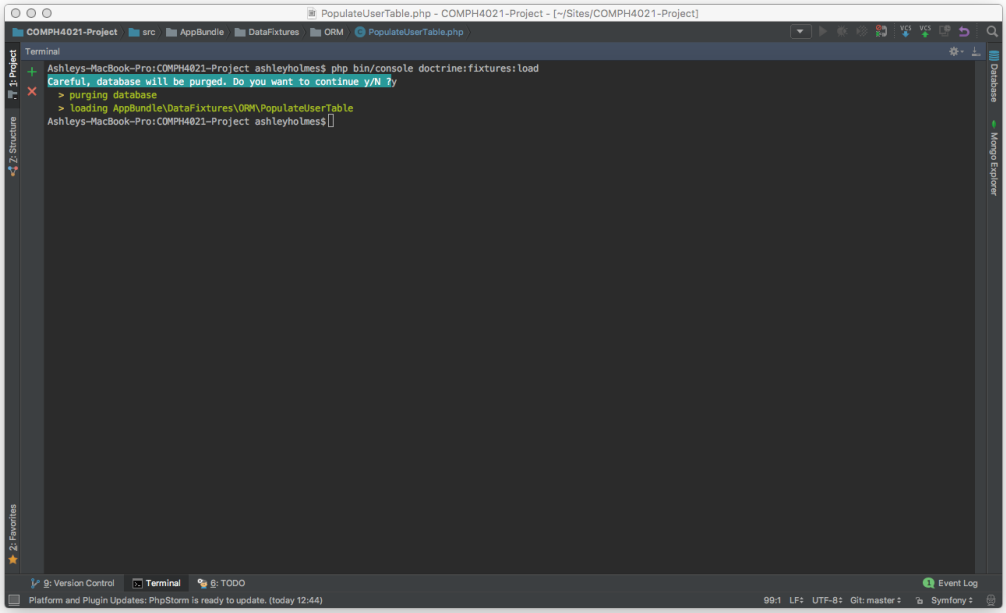

Figure 4.42: Purge Database

and add new data to it. Each user is persisted separately.

4.11 Passwords

4.11.1 Password Fixtures

This section covers passwords and all that was done to encode them. There is a place where to make passwords and there is also a place in the fixtures where users are inherently uploaded. The first thing which was done was to work on the fixtures. There were a few things to do to make that work.

- Make the fixture ContainerAware by means of the ContainerAwareInterface.
- Adjust the security.yml file with the bcrypt encoder.
- Write a method which uses the bcrypt to hash the password.

- Encode the passwords in the fixture.

When the ContainerAwareInterface was implemented in the class declaration all of the services are then made available for use. This is needed to encode the password in the fixtures file. The file needs to be made ContainerAware. The encoder was added to the security.yml which is figure 4.43 lines 4 and 5.

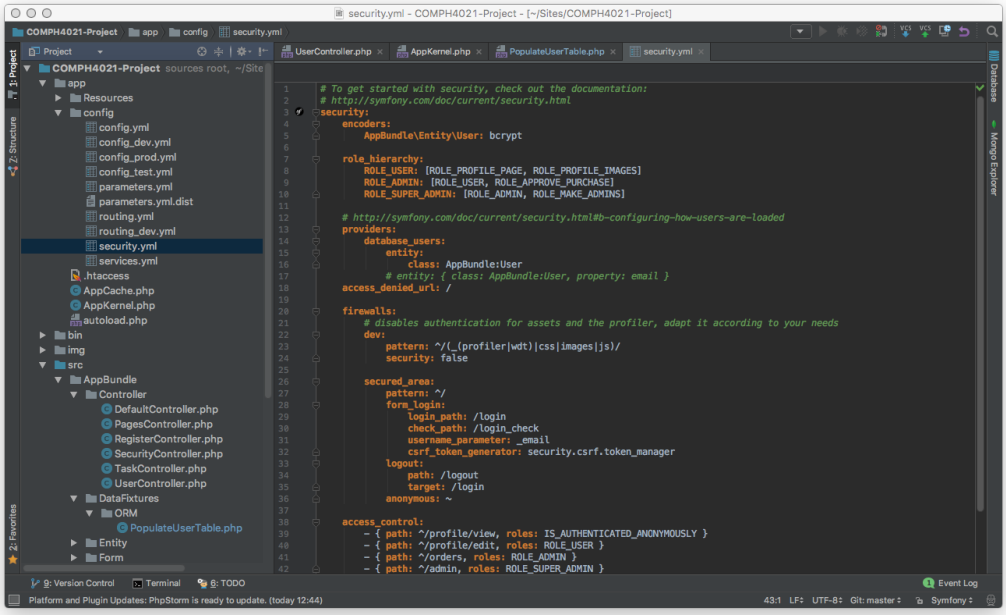

Figure 4.43: Security.yml

With that being done bcrypt was available for use. Use statements were also needed to make it work which needed to be added to the PopulateUserTable.php class. The use statements can be found on lines 14 and 16 of figure 4.40 and a private property called container was added in line 23. There is also a public function called setContainer which takes an argument of ContainerInterface and initially sets the container variable to null. Followed by this is a function which does the password encryption in figure 4.44.

The entity gets passed into the hashPassword method and the password which needed to be encrypted. With the fixture being containerAware it can be used to get the password encoder from the inbuilt security service. The security service was enabled in the security.yml

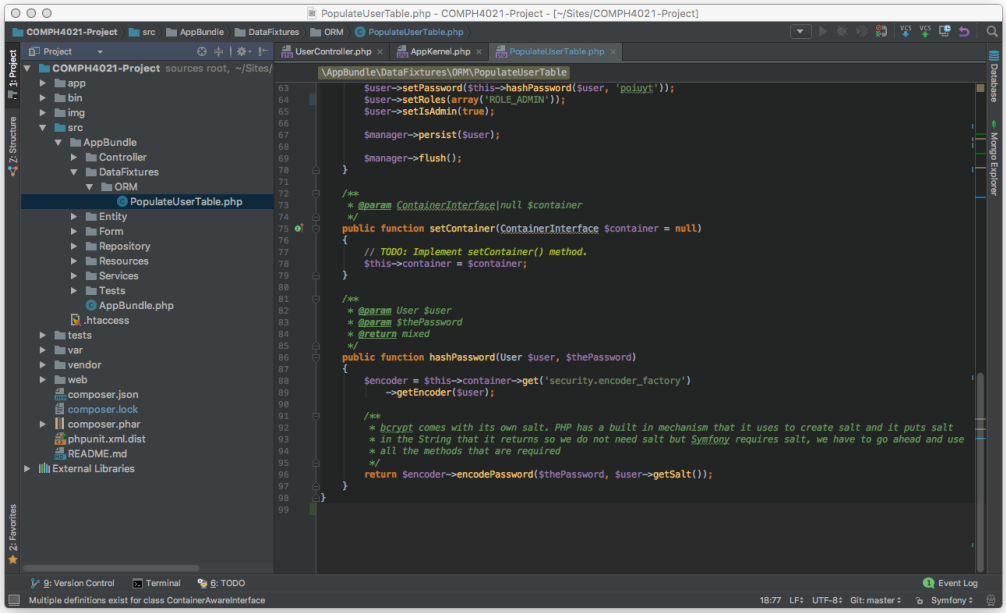

Figure 4.44: Hash Password

file and specified which encryption method which was used. Line 88 is where the encoder variable name is to store that object or service. The service is then captured by calling the containers get method and passing in the service which is to be used. In this case, it is the security encoder factory. That service has a method called getEncoder and the entity is passed in once the method is invoked. The object which has been specified encoder has a method called encodePassword and if passed a password and salt to encode it and it will return and encrypted password. The salt to use also needs to be defined however, bcrypt comes with its own salt, PHP has a built-in mechanism which it uses to create salt and it puts the salt in the string which it returns. So salt is not needed however, it is required by Symfony as all the methods need to be used which are required. This is why a getSalt function had to be created in the User entity figure 4.45 and returns null as bcrypt does not require anything.

Encoding the password is executed in line 38 of figure 4.41 by capturing the hashPassword method which was created and passing in the user object and the String of the password. The encrypted passwords are in figure 4.46

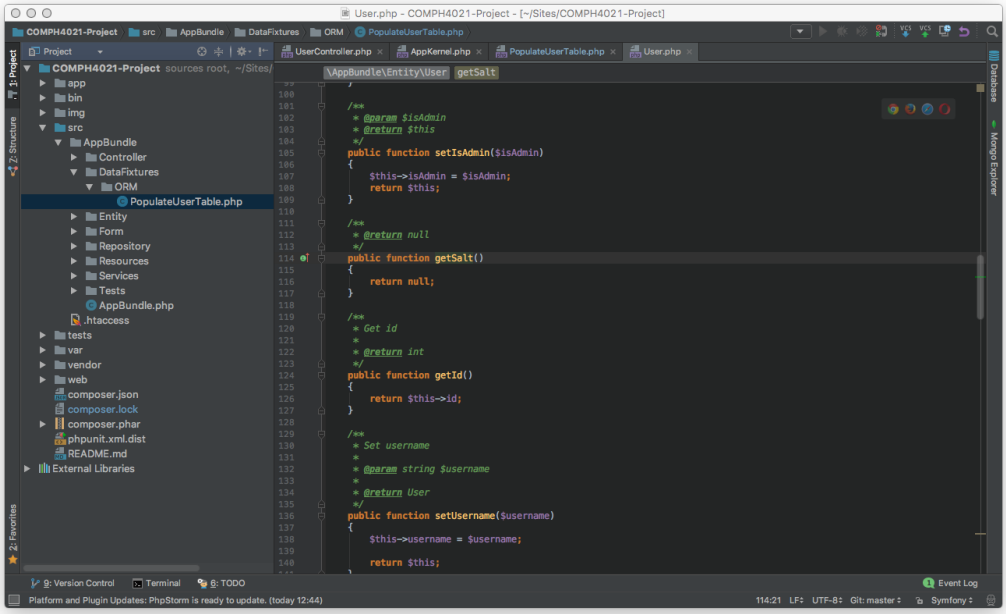

Figure 4.45: getSalt Method

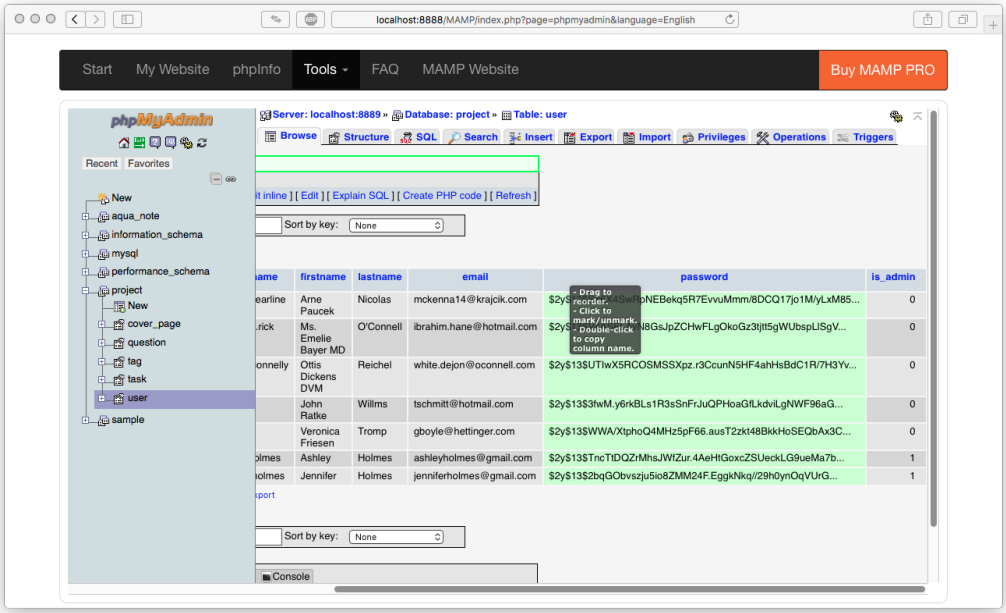

Figure 4.46: Password

4.11.2 Password Services

The last section covered encoding passwords from a fixture. This section is oriented around encoding passwords from a controller and writing the service. To do this, the following steps are needed:

- Implement the UserInterface.
- Create the service for encoding the passwords.
- In the sevices.yml file implement the service.
- Use the service container to call the encoding method.

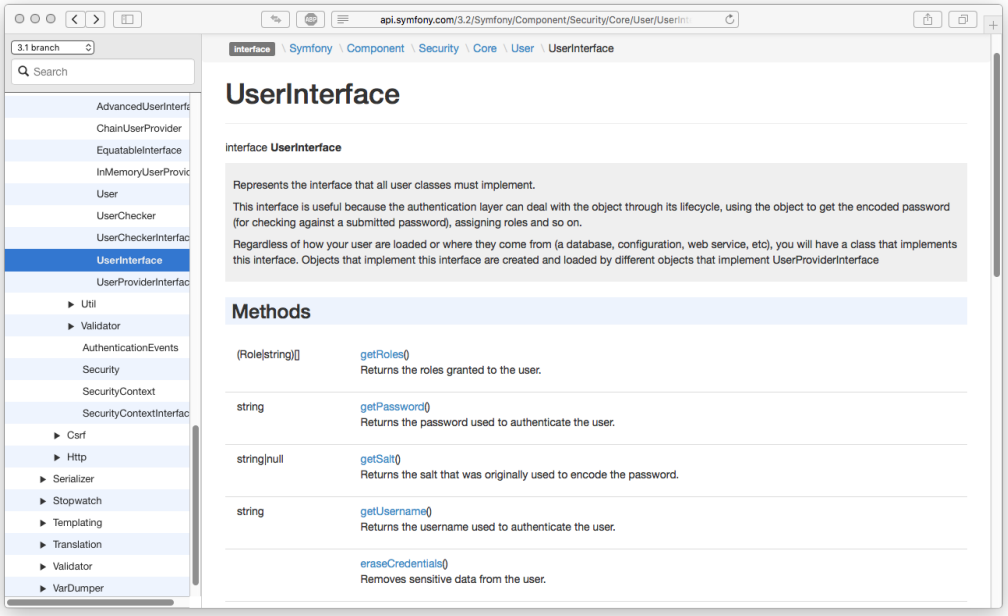

Figure 4.47: UserInterface

Symfony is a set of bundle which does what it is asked to do. It is possible to create a security system from scratch if that is what was intended. However, Symfonys in-built system is so robust. Using it saved much coding time and gave much more features than

it would be possible to code from the bottom up. To use Symfonys security system the UserInterface needed to be implemented. From the documentation. The UserInterface has five different methods to implement figure 4.47 shows this.

To implement these methods a statement was added to the User entity class. From figure 4.32 which we originally shown in the validation section of this chapter. Line 6 is added and the interface is implemented on line 21. Some of the methods were already in the User entity class. Only the ones which are not already there were added. Such as a function by the name of eraseCredentials which was left blank. Although it is not used in this application. The class is required by the interface. The same use statement is added to the service. A service is an object whose responsibility is to perform a specific task. The service which was created was to encrypt the password. Services are a three step process.

1. Create the service.
2. Enable the service in the services.yml file.
3. Use the service in the controller by calling it with the containers get method.

Starting with creating a file called UserManager.php which lives in a folder called Services which was added to AppBundle. This file was namespaced in order to match the hierarchy. And then the use statement from the User entity was added. In addition was this line 11 in figure 4.48

The private variable encoderFactory in line 19 is where the encoderFactory is stored. The constructor in line 24 is injected with the encoderFactory object and therefore no setter method was needed. The method that encodes the password is on line 33 and takes two parameters such as the userInterface and a String which represents the password which is to be encrypted. A variable was set in line 38 to store the encoded password and the it needs to be encoded which is done by the EncoderFactory which takes an argument and the UserInterface is passed in. The encodePassword method in line 39 takes two arguments. The String that needs to be encoded and the salt which is used. Which is actually bcrypt. The

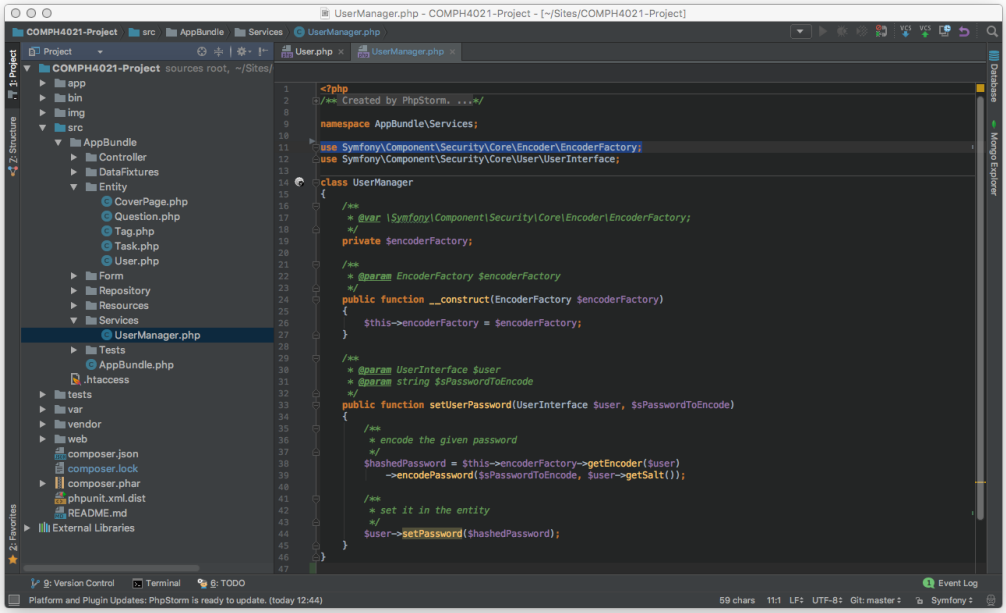

Figure 4.48: UserManager

password was set into the entity by user which implements the UserInterface and by calling the setPassword method pass in the hashedPassword.

The second step was to go into the services.yml file and enable the service. The service was named which could be anything but it was named in order to indicate where it resides. Then tell the caller of the service where the service is on line 8 of figure 4.49. So providing the namespace of the service which was created. So when the AppBundle, Services, UserManager is called in the controller it will grab this class. Then tell the container what arguments to pass into it. The arguments on line 9 are passed in which is the encoder factory object.

In the UserController under the newAction the service needed to be called in and encrypt the password which would be passed. This was done on line 56 of figure 4.50 with the containers get method which was the same name used in the services.yml file. The encoded factory is automatically pulled in as an argument was attached to that class. Line 57 calls the method that was created in the UserManager. Passing the entity which is the user and the password.

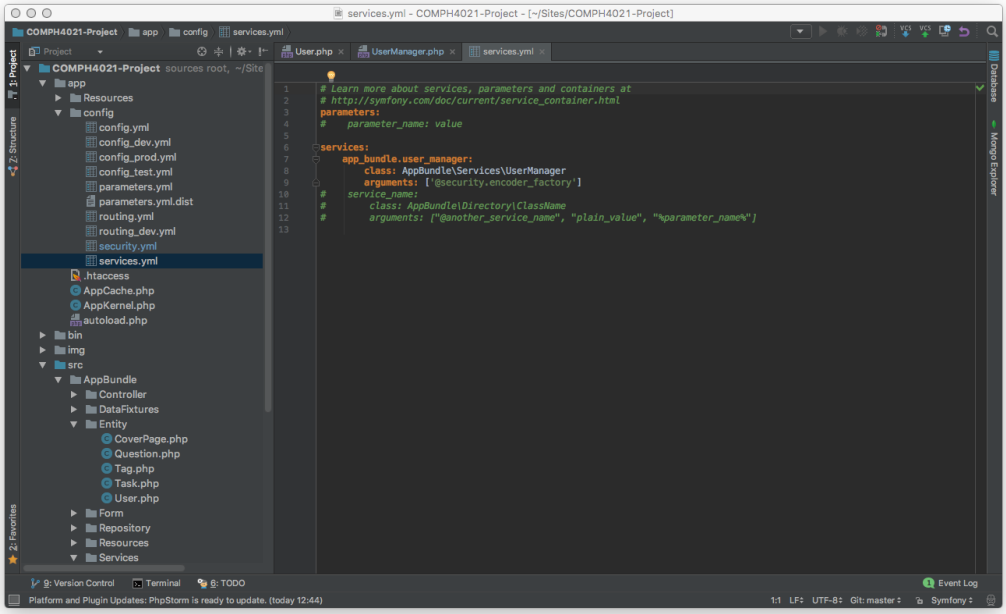

Figure 4.49: Yaml Services

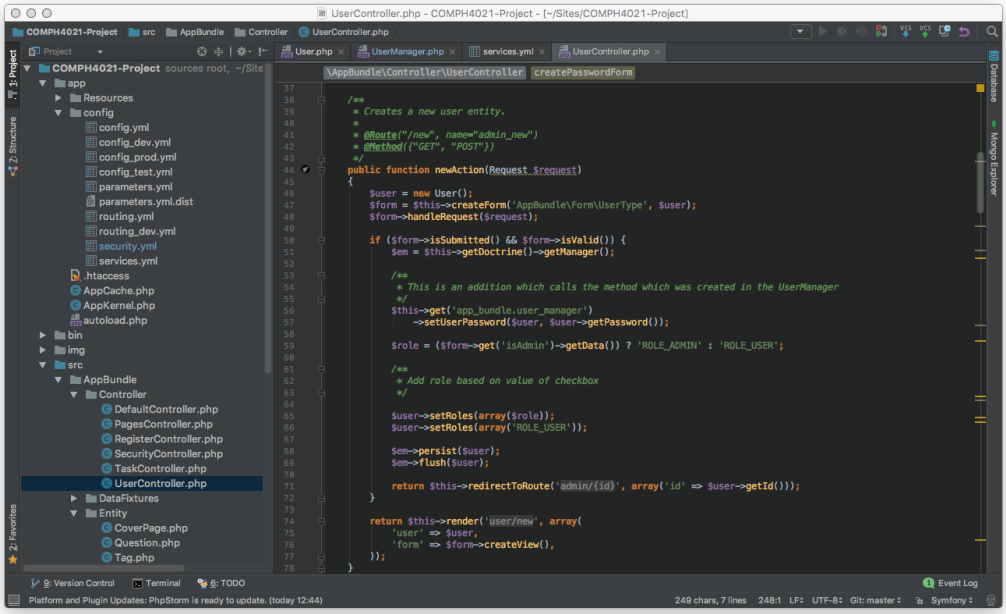

Figure 4.50: UserController

4.11.3 Change Password Form

The following is explained in order to create and handle the new password form.

- Create a new password.html.twig file.
- Create a password controller.
- Password form generator method.
- Form submission controller.
- Redirect.

A new file was created in the user directory called password.html.twig as with figure 4.51.

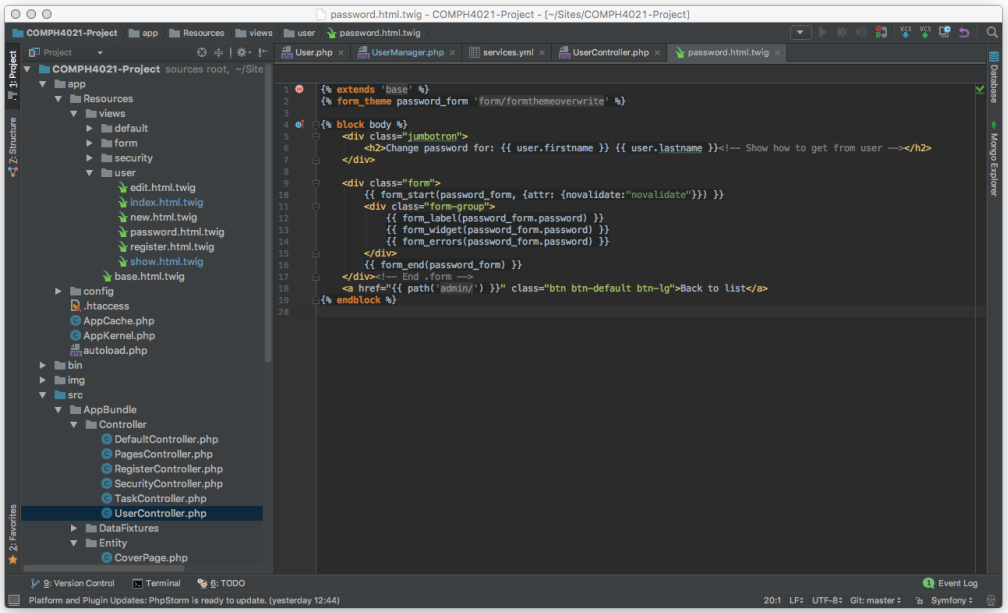

Figure 4.51: Password Twig template

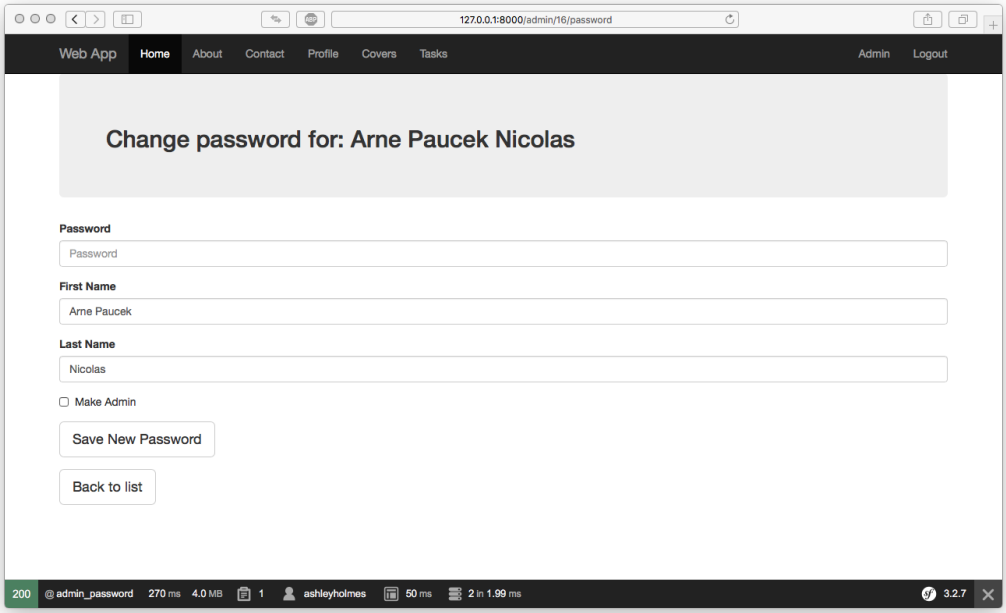

Figure 4.52: Change Password Form

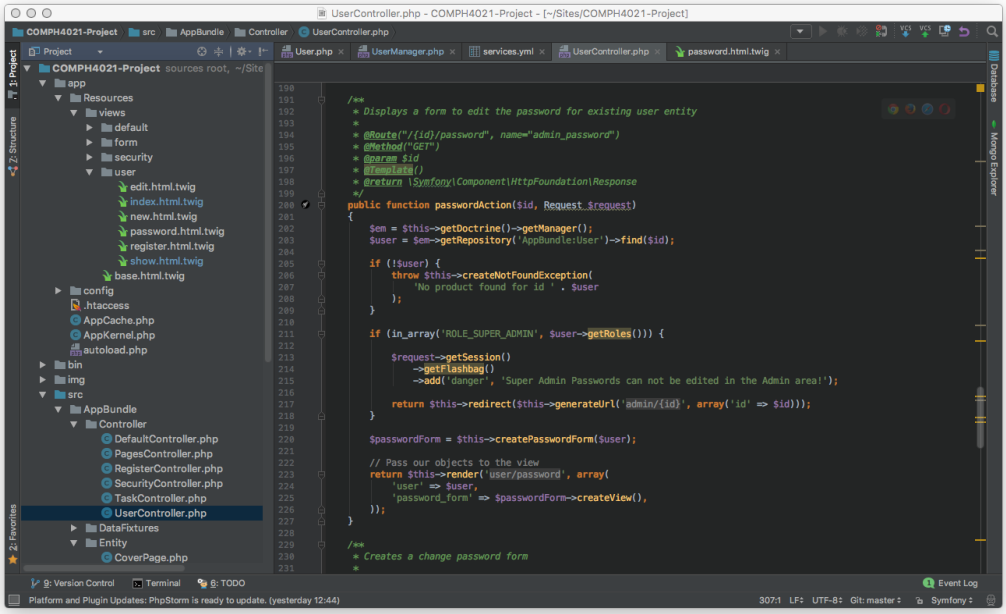

Figure 4.53: passwordAction Controller

Creating the controller in the UserController. There were two things to cover in writing the controller which handle the form. Which was create the view and display it. And then handle the form and redirect it which will be covered in a later section.

The Action suffix is required on line 200 as it is a controller which handles a view. The view is the Twig template. Passing in the id which creates a hook into the entity manager on line 202 using the container. And then using the entity manager to get the data with the repository using the find method and pass it inside the entity object. If the entity is not found and exception would be generated. This happens on line 205 to 209. A route is then added as annotation in line 194 in order to get into this function. The route needs to be passed an id and give it a unique name. In other words a variable id and then password separated by a forward slash. The name in the route is what was used in the Twig template. The next annotation is what method was used to get to the route which is a GET String. The last annotation is Template which does not take an argument as the suffix Action was used. Symfony can figure out where it is in order to get the template. In this case it would look for a password.html.twig within the Resources folder and a folder underneath that called user as that is the name of the controller. The form is then created in line 220 by calling the createPasswordForm method. Lastly, render the view by passing the view the entity and the form in an array on line 223. A link to the route was created in line 61 of figure 4.27 for the user which has access to look at the page can change a password. So far this only displays the form. Once a change was made to the password a way in which to handle the form submission was required. This is shown in figure 4.54 which creates a new form.

Line 235 gets passed the entity which is the User.php class and then the form gets returned to the invoker in line 253. The validation is managed in figure 4.55 on line 288. If the form validates, the data gets added to the database. If not, the rest of the action will execute and return another form with the errors. This is done through annotation. Line 291 is the service which was previously written and encode the password. The password has been encoded and the entity already persisted. In line 294 it is flushed to the database and then redirected to another page which is the admin show page in line 297 which requires an id.

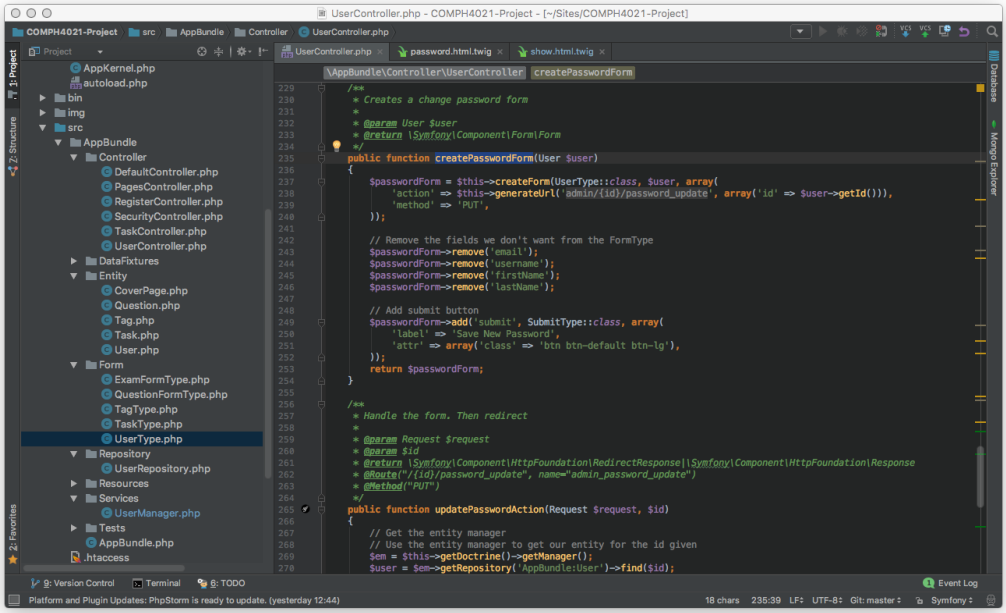

Figure 4.54: createPassword Form

Some functional testing was done to ensure all implementations were in order.

4.12 Authentication

4.12.1 Login Form

Authorisation is about logging a user in. If a user is on the site whether they are logged in or not. They are still authenticated, however the authentication is anonymously as was seen at the bottom of figure 4.12. This is where the login form is appropriate. To do this is as follows:

- Adjust the security.yml.
- Implement role hierarchy.
- Look at how the security system takes over the login process.

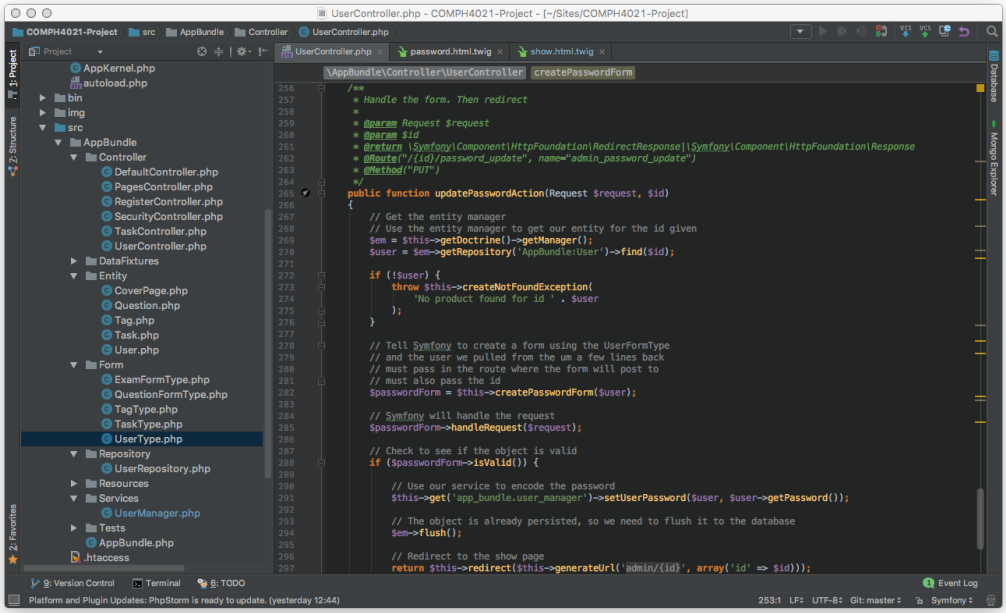

Figure 4.55: Password Form Validation

- Go through the SecurityController.
- Adjust the twig template for login.

The five areas where the configurations were set are role hierarchy, providers, firewalls, secured area and access control in figure 4.43. Role hierarchy is the role of the user. With the roles inside of role hierarchy defines the page level or folder level specific actions which the user can do. Role Admin inherits all the roles which the user has in addition to other page or folder level roles. The same for role Super Admin. Providers has database users meaning the roles will come from the database. The providers need to be told where to get the entity from, since using the database user and this is AppBundle:User and the property to check against is the username. Firewalls is just as it sounds. It gives specific access or meaning to areas of the website. The one on line 22 is for development purposes and it states that there will be no security on any profiling pages. In secured area the pattern was set to anything. Paths were given which would later be used by the SecurityController in AppBundle Controller directory. Anonymous was set to everything in order to allow access to the login page by

all. The Symfony documentation provides a sample of the SecurityController which was available for use of which was modified to suit the needs of this project. The controller can be seen in figure 4.56.

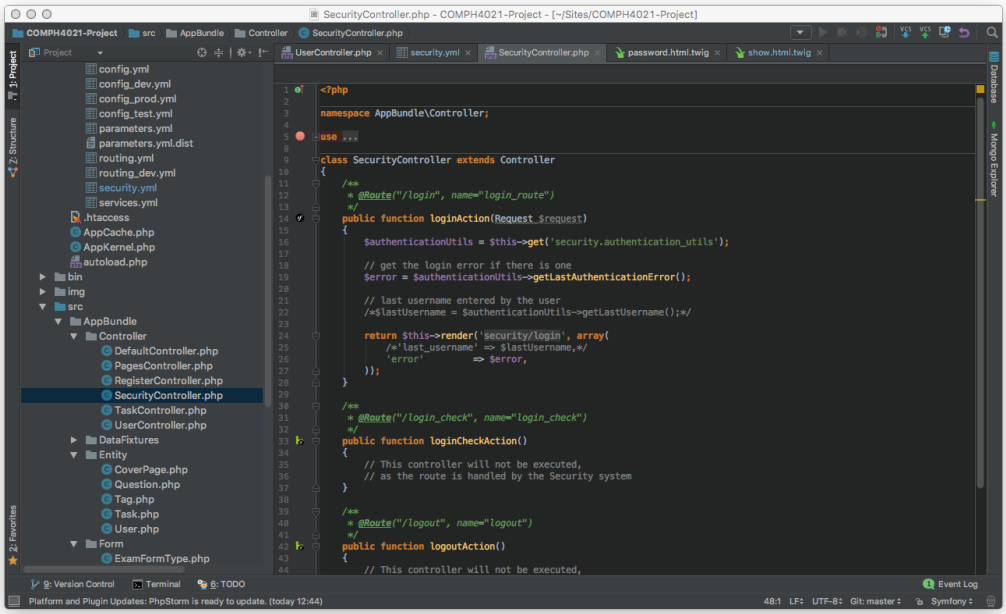

Figure 4.56: SecurityController

A sample login form was also available from the Symfony documentation page of which was used and changed and can be seen in figure 4.57.

Once logged in the Symfony debug toolbars profiler page shows which roles belong to the user and these match what was added to the security.yml and used in the SecurityController.

4.13 Controllers

4.13.1 Extending the controller

The login form and controllers were extended to have a successful user experience. The below section covers this.

- Make errors look better.

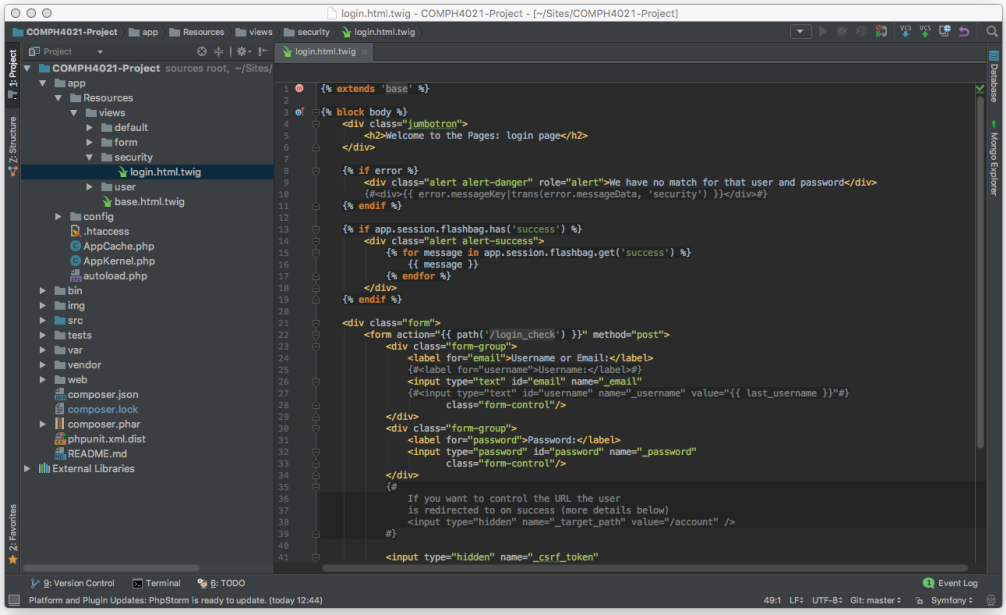

Figure 4.57: Login Form

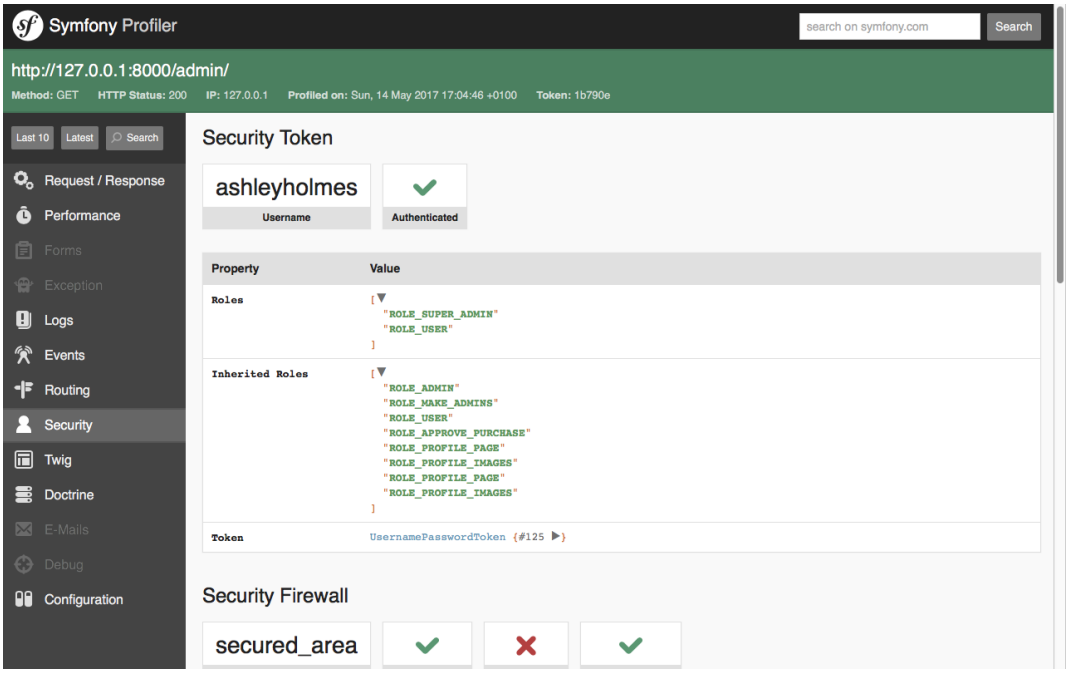

Figure 4.58: Symfony Login Profiler

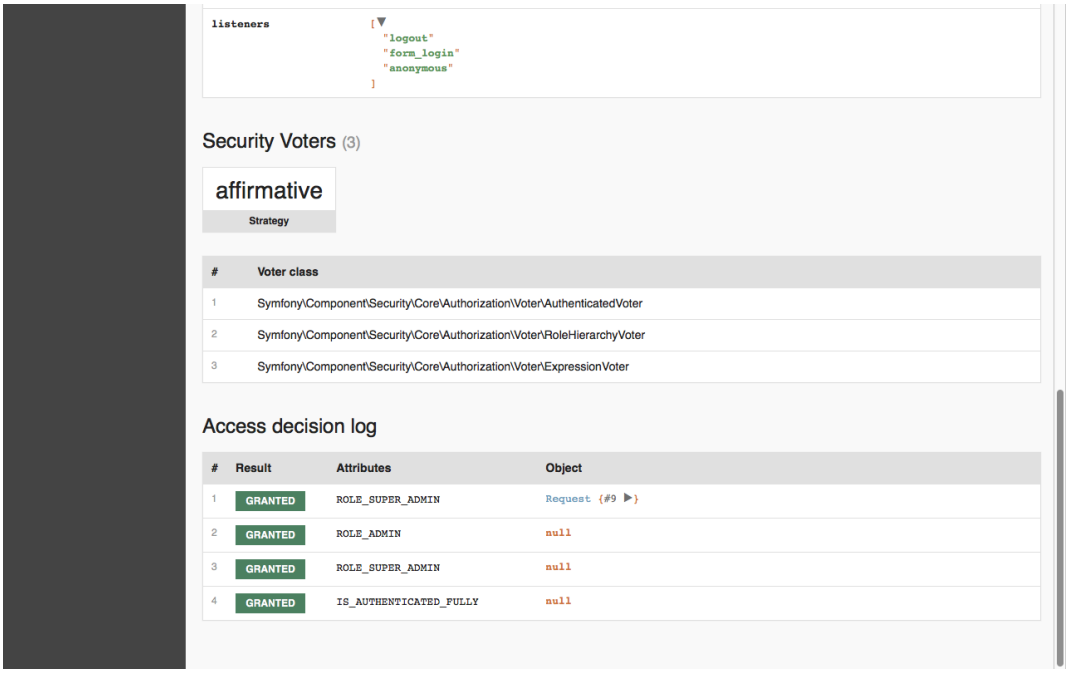

Figure 4.59: Symfony Login Profiler

- Adjust base template with the proper links.
- Make login by email available.
- Use twig function for assigned roles.
- Using the repository for username or email.

As of now the template which was used from the Symfony website was using a translation service, which was commented out in line 10 of figure 4.57. This was changed to a Bootstrap error message as used in line 9 to achieve the result in the image below. Adjustments were then made to the base.html.twig file where links such as login and logout were added. Twig logic was also added using its own is granted function to do a check for whether the user was authenticated with the relevant role and if so display the login and logout link at the top of the page. In figure 4.61 this can be seen from line 54 to 69.

Line 39 of the security.yml shows how to achieve a login through email. On line 24 the input tag for the label in the login.html.twig was change to accept the email instead of

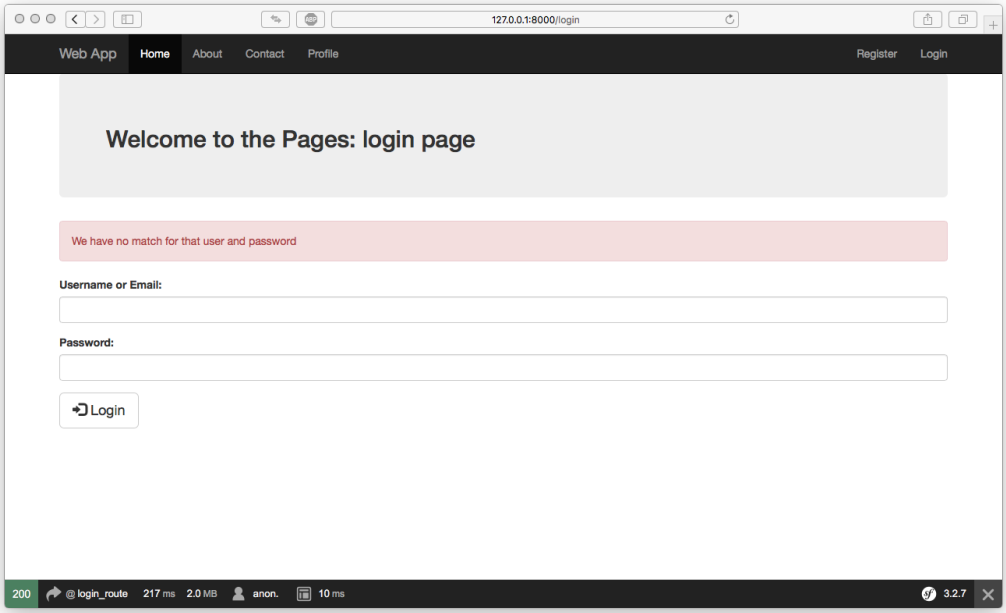

Figure 4.60: Login Form Error

username to make the change which was done in the YAML file. In order to login with both username and email address an extra step was needed. The code for this was picked up from Symfony documentation page and placed in the UserRepository figure 4.62.

4.14 Roles

4.14.1 Putting roles in the database

Roles are placed into the database and interacted with through the Twig templates and controllers. Below is how it was done.

- Add a property in order to use JSON array type.
- Update the schema.
- Set role by making adjustments in the controllers.

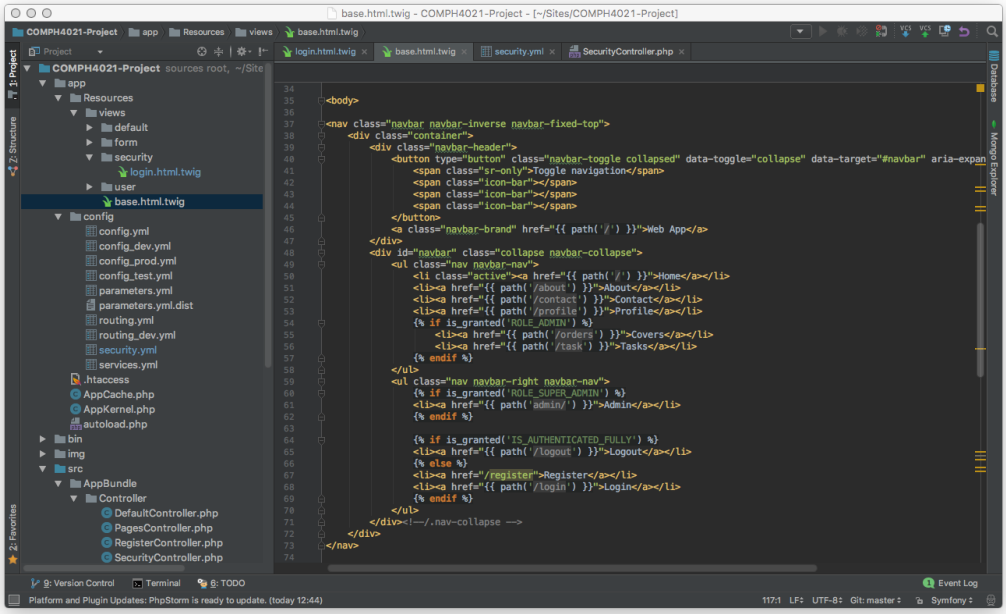

Figure 4.61: User Roles in the the base template

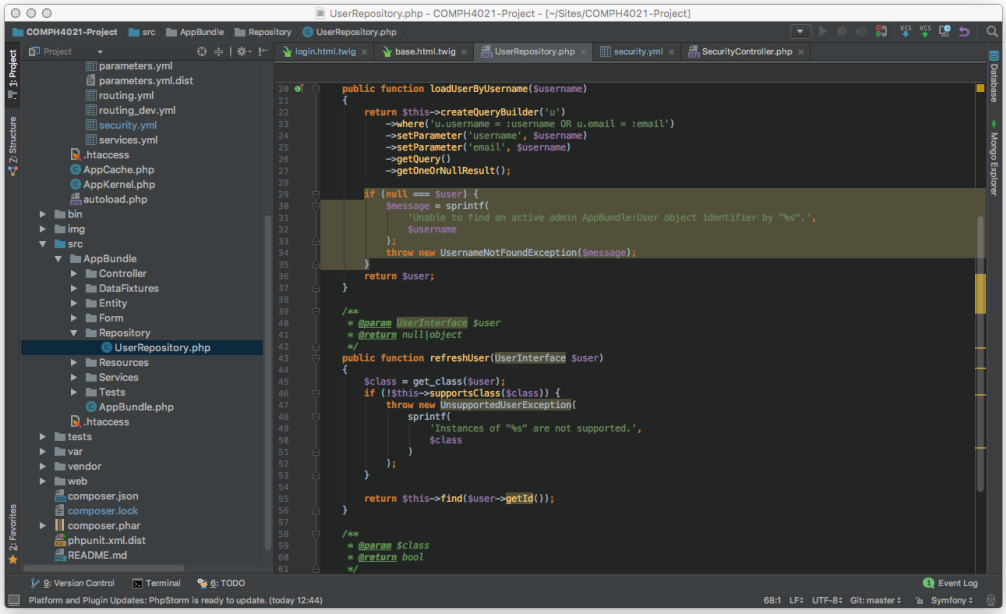

Figure 4.62: UserRepository

- Adjust the fixtures.

So far the getRoles method has been used as with the UserInterface. However, there has been no mechanism to store values. To store values, a private property was addd to the User.php class in line 28 of figure 4.63 with the JSON array annotation above it. A getter was already in place. Only a setter method needed to be added. The setter method would be passed in an array of roles. Once this was complete the schema was updated through the command line.

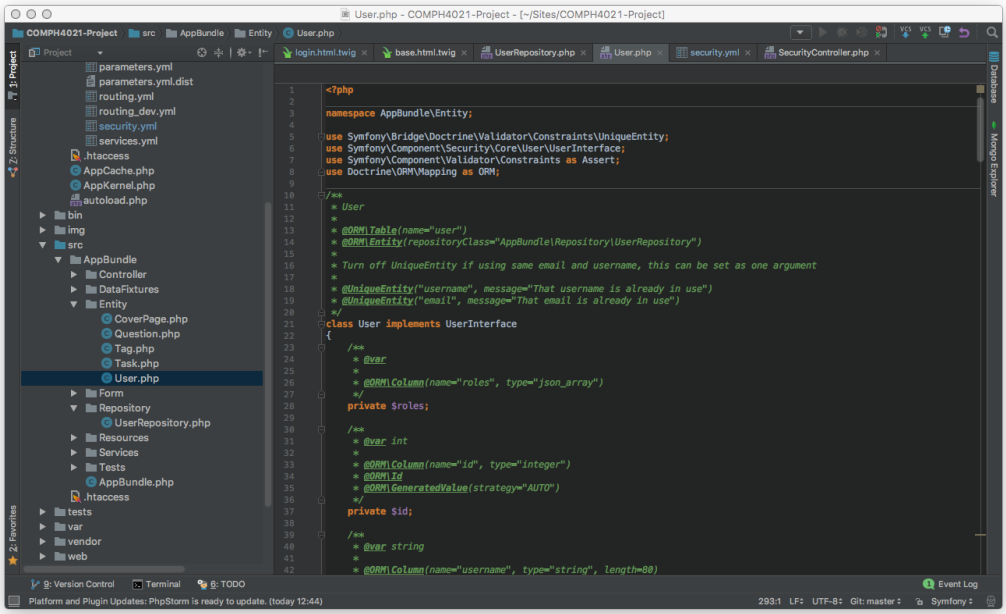

Figure 4.63: JSON array

At the initial creation of every user. That user was assigned the role of user by adding the two lines of code in the UserController. Line 64 and 65 show this with reference to figure 4.50. In PopulateUserTable the fixtures were also adjusted to add the roles to the users which were created. Back in figure 4.41 it was possible to see.

4.14.2 Using the roles in the database

This section covers how to use the roles from a database.

- Make User Admins from Super Admins.
- Adjust the fixture to reflect this.

Making the user an Admin involved creating a private property in User.php called isAdmin of type boolean. As there is no boolean in SQL, instead this was replaced with a TINYINT of one or a zero using annotation in line 89 of figure 4.64. A getter and a setter were also created for this attribute.

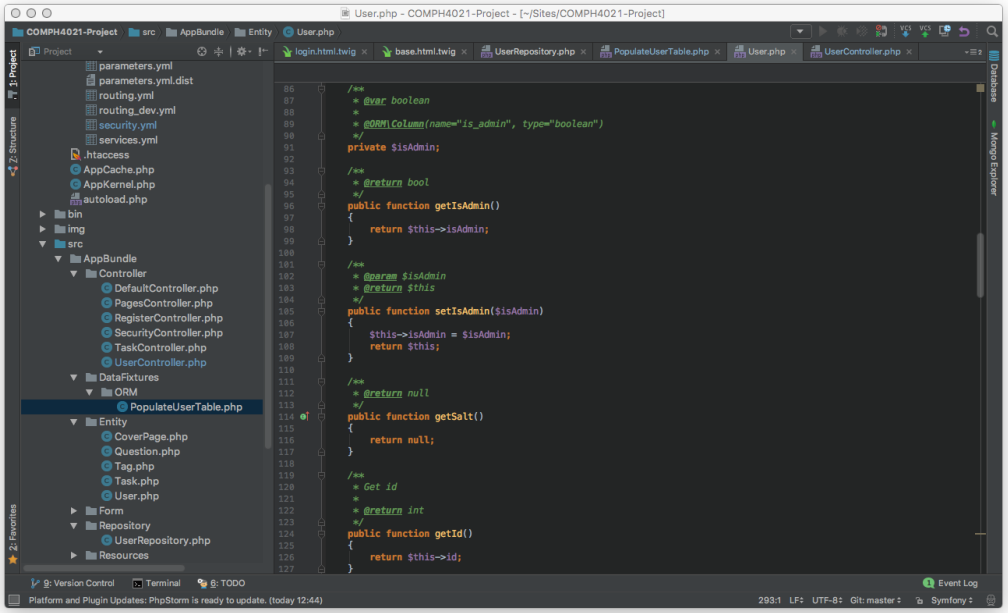

Figure 4.64: isAdmin

The schema had to be updated after this, in order to update the database with the changes which were made and add the new column. A checkbox was then added to the view in figure 4.65 lines 60 to 63, in order for the super user to add the role of Admin.

The logic for this was placed inside the UserController.php in the newAction function on line 59. The isAdmin was also set to true for both the Super Admin and the Admin with exclusion to the regular users in the PopulateUserTable.php

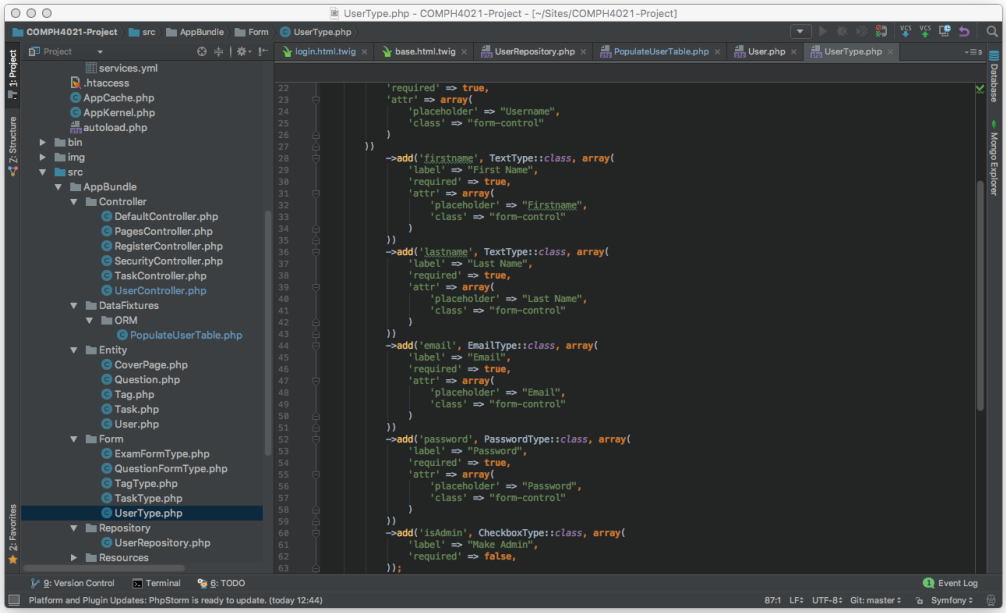

Figure 4.65: CheckboxType

4.15 Authorisation

4.15.1 Access Control

- Access control in security.yml.
- Access denied parameters.

In the section of authentication. The access control attribute was left empty. In this section it will be completed. Creating the control list asserts the type of role to have access to a set of pages. It is available to see at the bottom of the security.yml file 4.43. It is a hierarchy for users to view specific pages. On line 18 there is an access denied url and this will send an any user back to the home page incase they are denied something which only works for authenticated users and not anonymous users.

4.16 CSRF Protection

4.16.1 Testing the roles

By generating a few controllers to demonstrate some of the access control configurations and protecting agains Cross-Site Request Forgery attacks.

- Generate a controller in console.
- Adjust the generated twig files.
- Adjust base file.
- Test against ACL.
- Add CSRF.

Making a controller class in the terminal requires the following command statement, php bin/console generate:controller. Giving it the name AppBundle:Pages, with a routing format of annotation and using a template format. Then by giving the name of the controllers which is preferred to have inside the controller class. The first was called Contact which needed to be appended with the name Action as a suffix. A second is created for Profile. The third for About and the last for Covers and Tasks. By confirming the generation, Symfony creates all the necessary files. The Twig templates were then adjusted to suit as the paths to the newly generated pages needed to be added. With that in place the menu has now been updated. And the navigation works between all pages which are all publicly accessible. When logging in with Super Admin privileges the Covers and Tasks tab is available for view. To add the CSRF protection. This was simply in line 32 in the security.yml and then adding a token in the login.html.twig template. The token is on line 41 and 42 of figure 4.66. The token is available to view using inspect element in the browser in figure

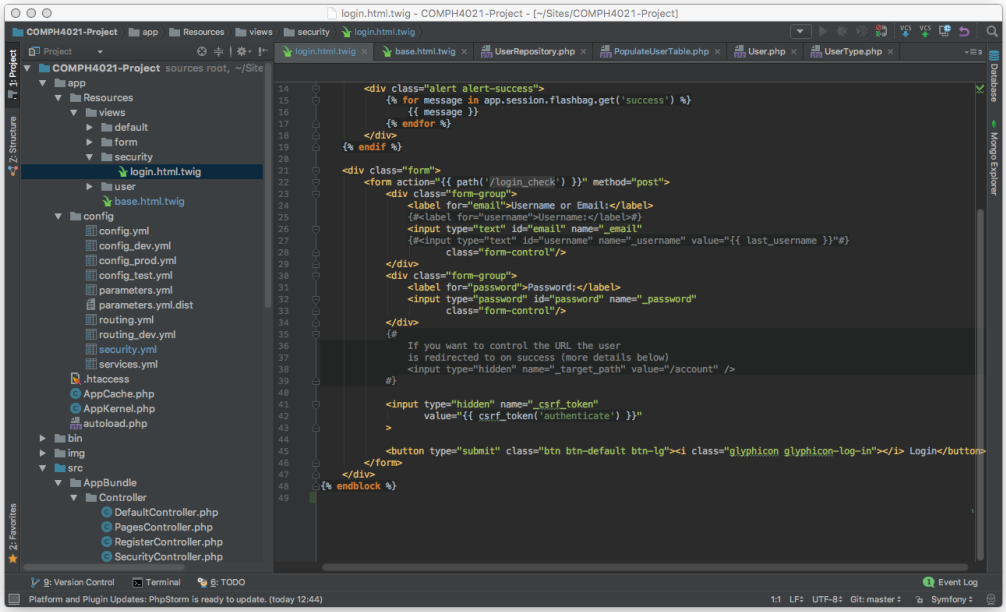

Figure 4.66: CSRF Token

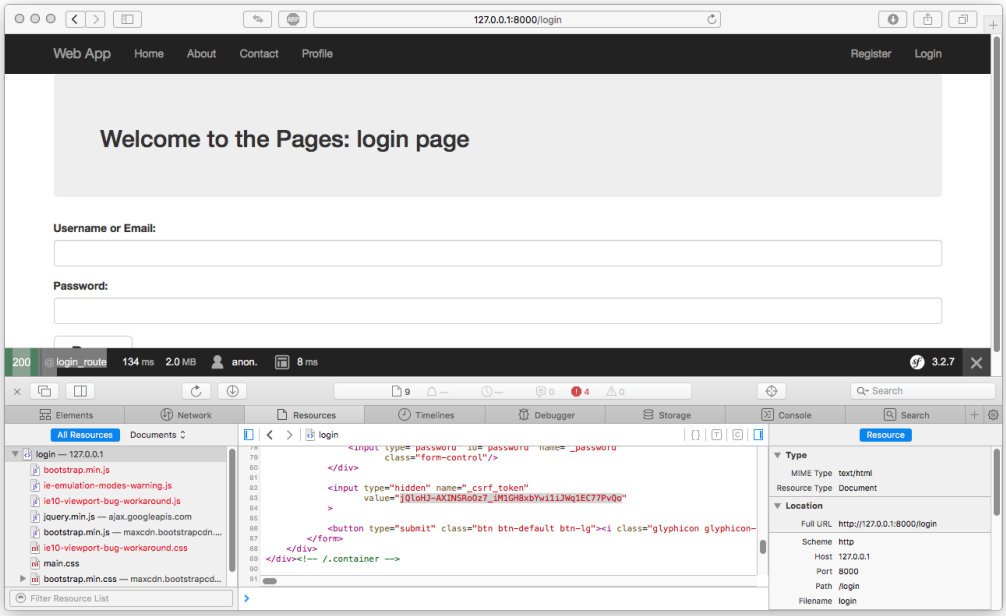

Figure 4.67: CSRF Token Browser

4.16.2 Controller and templates adjustments based on user roles

Controllers and templates are adjusted based on the roles which are stored in the database and which are stored in the entity of the users which are viewed.

- Restrict Super Admins from editing same Admin.
- Flash messages preventing confusion.
- Remove links for editing other Admins.

At this stage when logging in with the role of Super Admin it was possible for one Super Admin to edit the profile or password of another Super Admin. This needed to be prevented. In the editAction of the UserController a check was implemented in order to check if the user which is in the entity. Not the currently logged in but the entity which is viewed on the user list. There will be a check to verify if that user has the role of Super Admin or not. If they are a user then the authenticated user will be directed back to another page. This can be directly checked in array function on line 103 of figure 4.68. Looking for the role ROLE SUPER ADMIN and calling the getRoles method which returns an array. If that role was inside the entity which was being viewed then the user would be returned to the show page. With not being able to go into and edit another users edit page. That user was not able to see if anything was happening behind the scenes. Therefore a message was to be created to tell the user what was happening. Symfony has an in-built session management system. Within that system is what is called flash messages which is on line 106. Chained onto that is the add method which takes two arguments. The first is the name of the flash message and the message to the user. These flashbag messages are displayed in the show.html.twig template. Figure 4.69 has these for view on lines 8 and 16. This was also needed in the function passwordAction and deleteAction. A better approach to this was to not present those links to the user at all. So the adjustment was made and is on line 55 to 59 by means of an if statement in figure 4.70. It was also necessary to add this to the index.html.twig.

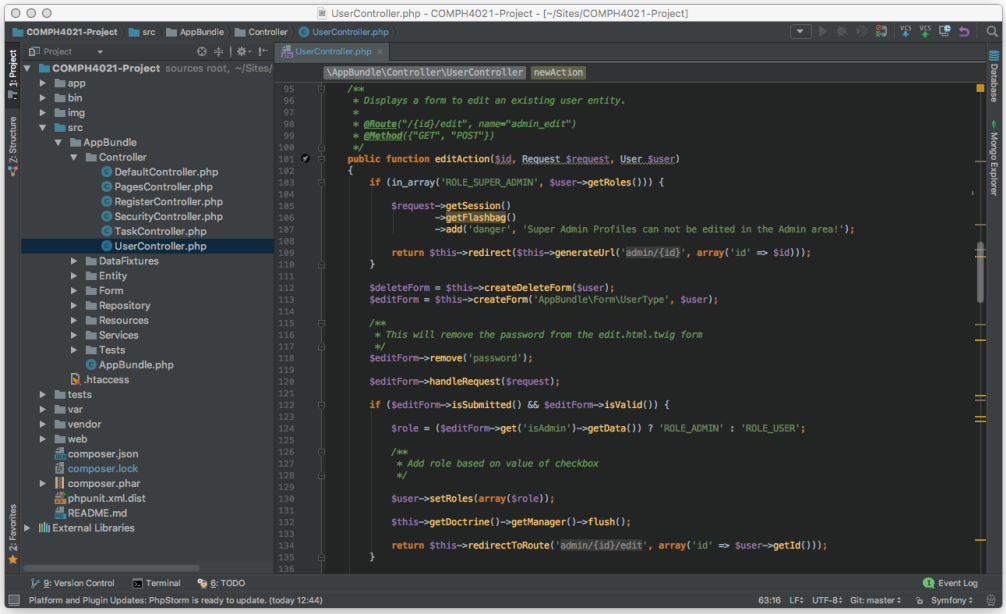

Figure 4.68: editAction

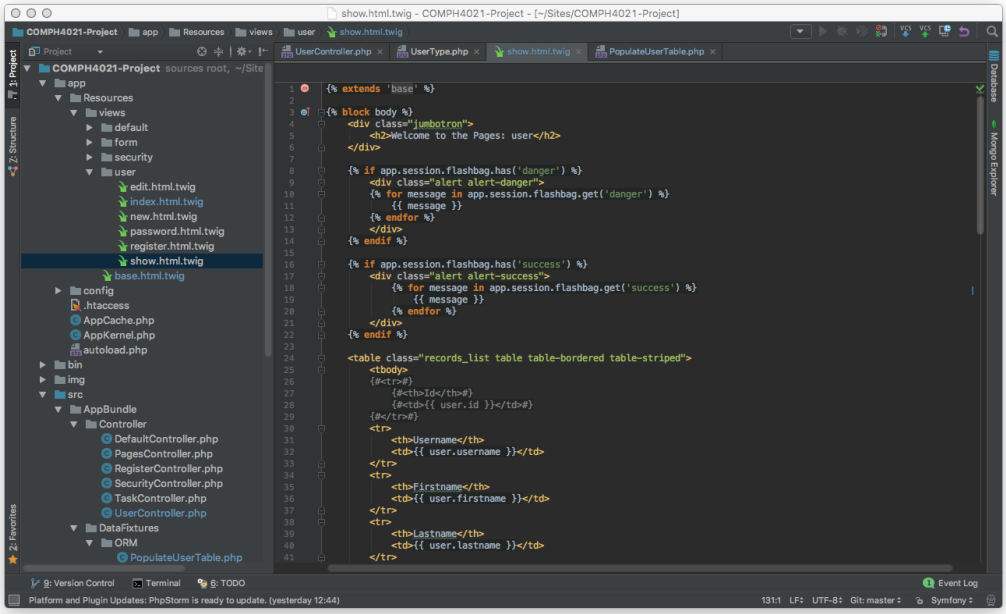

Figure 4.69: Show Page

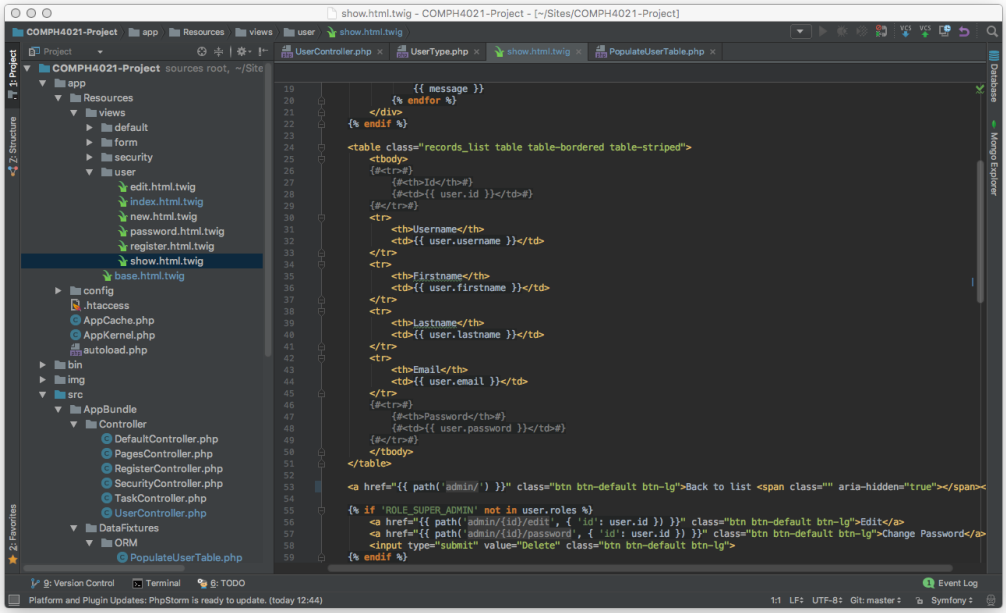

Figure 4.70: User not in ROLES

4.17 Registration

4.17.1 Registration Controller

The RegisterController was associated with creating an action which creates a form. Displays it and processes the form for a new user.

- Create RegisterController.
- Create a new register template.
- Add flashbag messages.

Using the entity of type user. A variable of type form was created. Inside that variable a createForm object was placed. It uses the new UserType and the entity which the form was created from. For this form the isAdmin was removed as it was not needed. And a button created for the submission. This is not using the template object so what needs to be returned

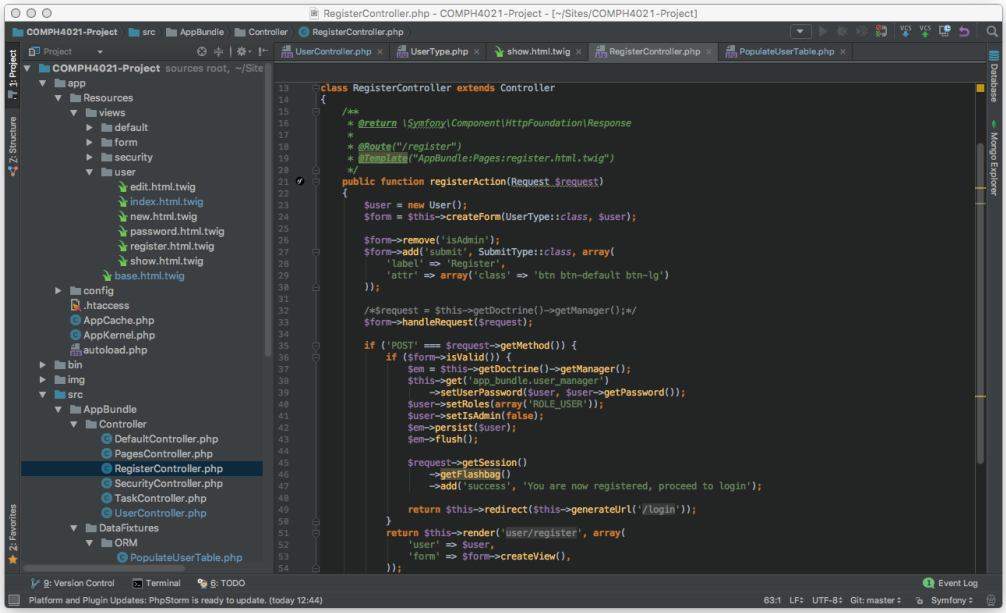

Figure 4.71: RegisterController

is a rendering statement as this render. Placed inside is the template which it will be using which is the argument. In this case it was the register template. An array was added and this is where the objects are passed to the response object which creates the view. The entity is passed into the array and then the form. A route is then defined as annotation using @Route /register. Another annotation needed was the @Template to tell that annotation which twig file it will be rendering from.

The register template is very similar in code to the new user creation form. This form request needed to be handled. It was achieved by whats on line 33 of figure 4.71. Then do a check to see if the form was posted and if it was valid. If the form was posted and was not valid there needed to be a redirect to the same place in order to be rendered with mistakes which come in the entity.

When handling the form incase it was valid. There were a couple of values which needed to be explicitly set as it was not handled by the FormType. The password needed to be encoded. And set ROLE USER inside the set roles method and that needed to be of type

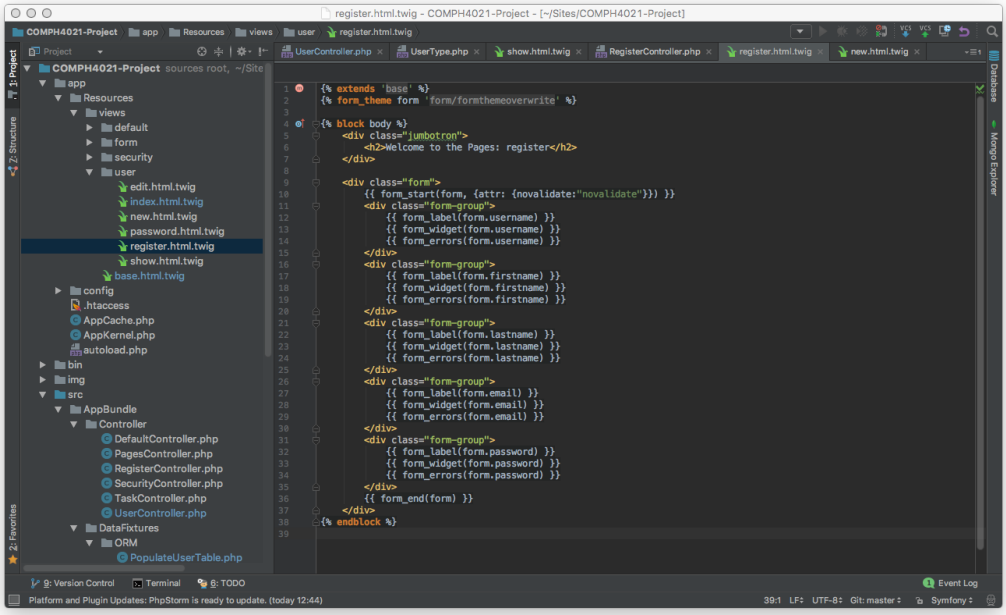

Figure 4.72: Register Twig Template

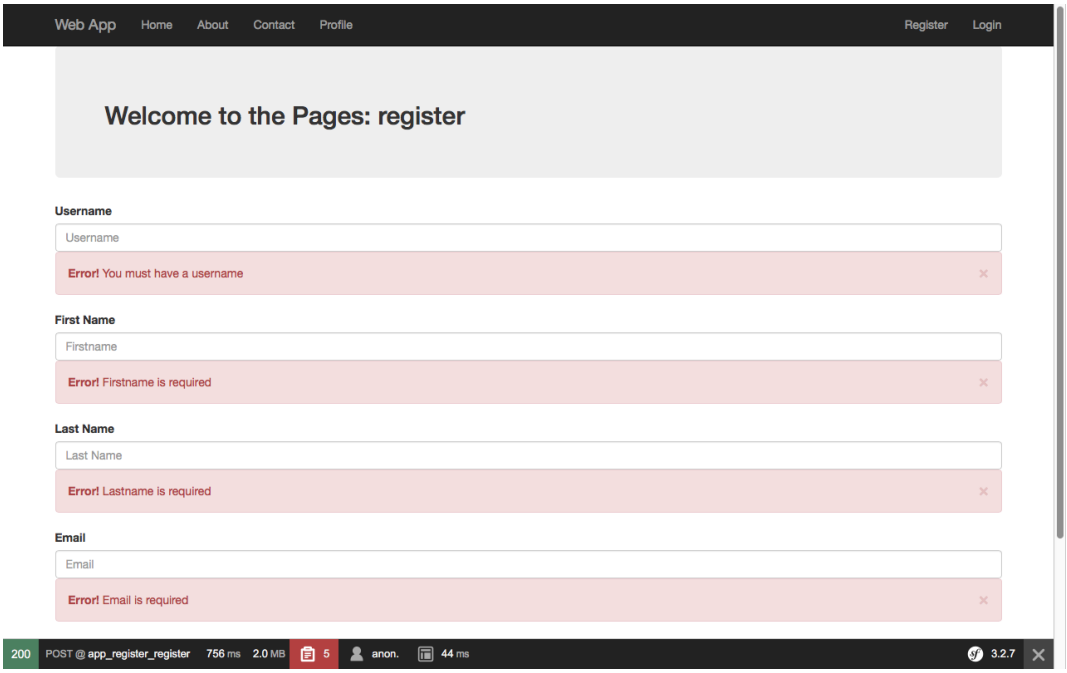

Figure 4.73: Register with Errors

array. And call the setAdmin method and called it false as the new user was not able to be made admin at that position. The entity was persisted and then flushed to the database. Once the user had completed the registration process they were redirected to the login page and asked to login.

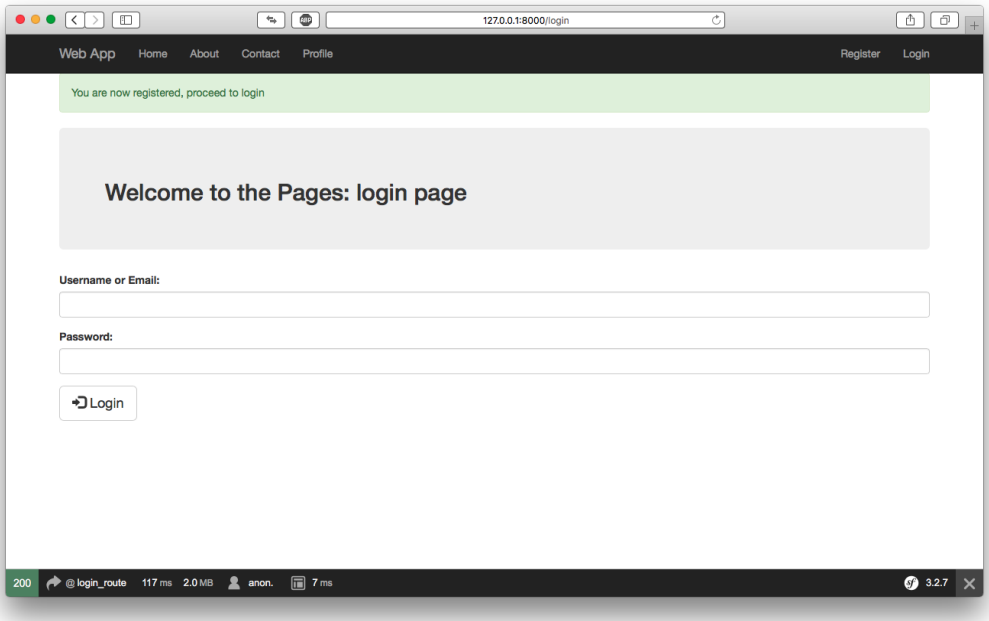

Figure 4.74: Registration Successful

In order for it to not be a surprise a flash message was generated in order to let the user know that the registration was successful. This is seen in figure 4.74.

Chapter 5

Testing - ”Testing and evaluation”

5.1 Introduction

With every new line of code, comes the potential of a new bug. Testing the code makes for a better built and reliable application. The type of testing undertaken for this application was functional testing.

5.2 Testing the Homepage

5.2.1 Functional Testing

Testing initiated with the homepage. Within the navigation bar at the top of the page, all links were checked in order to ensure that the user was brought to the correct link. In addition to this ensuring that the pages were loaded without any errors. The only errors which should be generated are those that are intentional. Intentional errors are included in figure 5.1 and 5.2. This shows that the code was doing what it was meant to do.

Errors are shown in the Symfony debug toolbar figure 5.3 and if one looks at the profiler it was clear that credentials were not provided by the user. Figure 5.4.

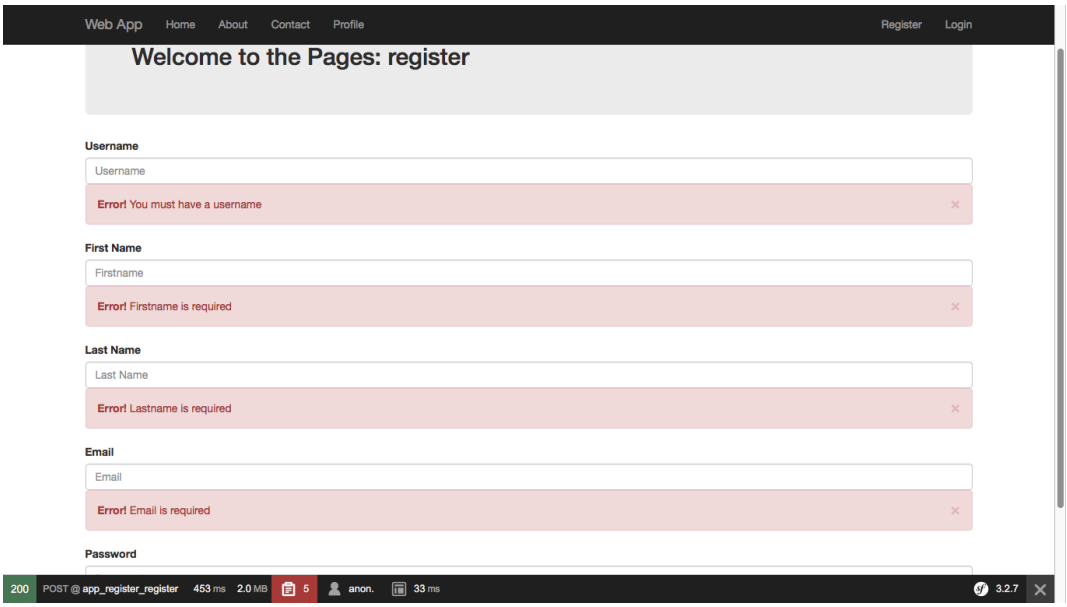

Figure 5.1: Intentional Error Register

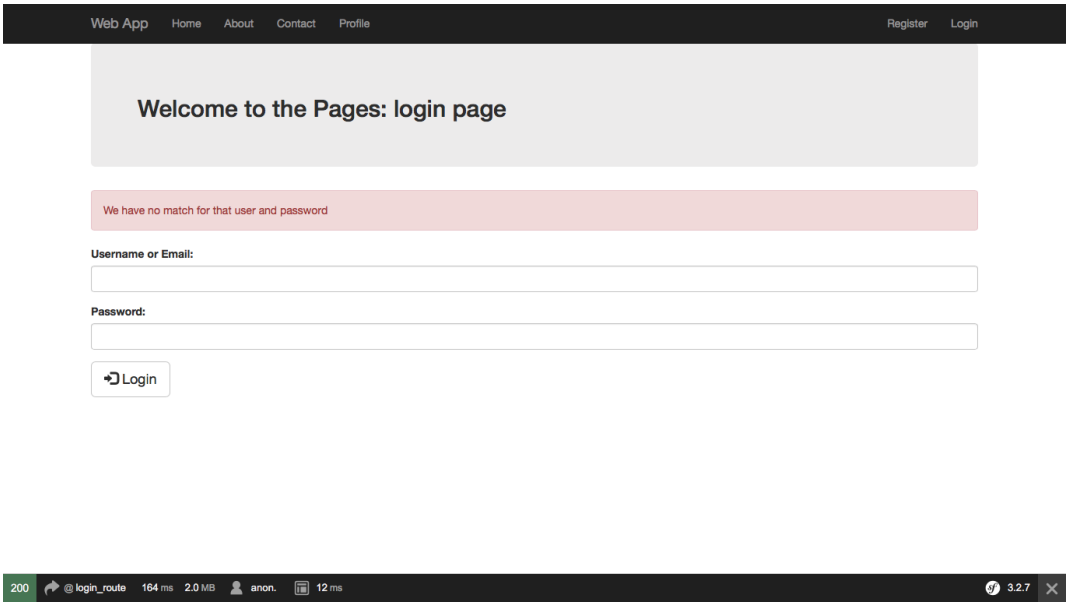

Figure 5.2: Intentional Error Login

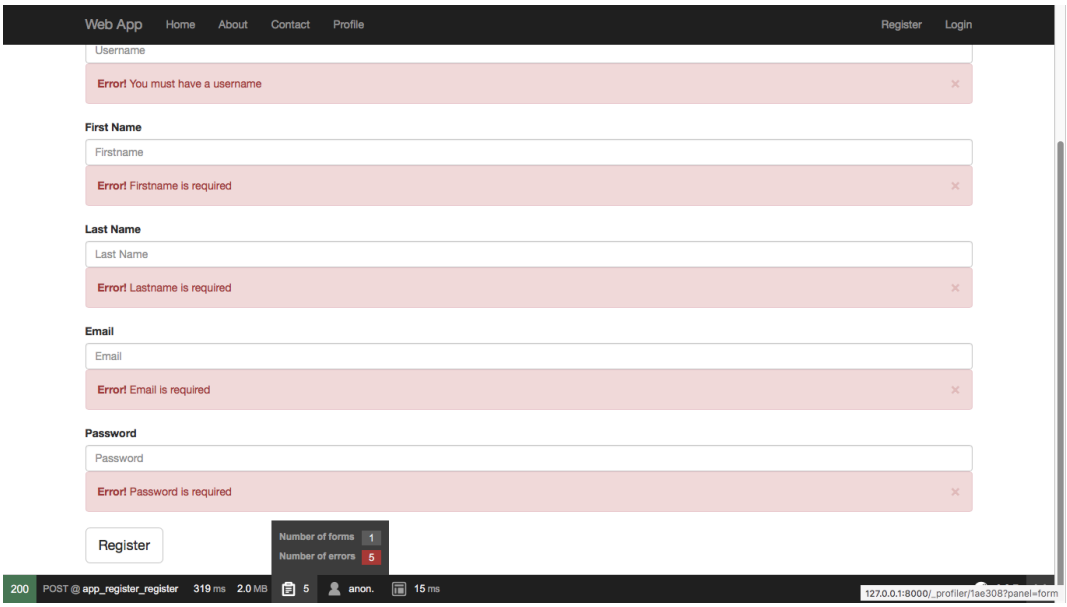

Figure 5.3: Debug Error Register

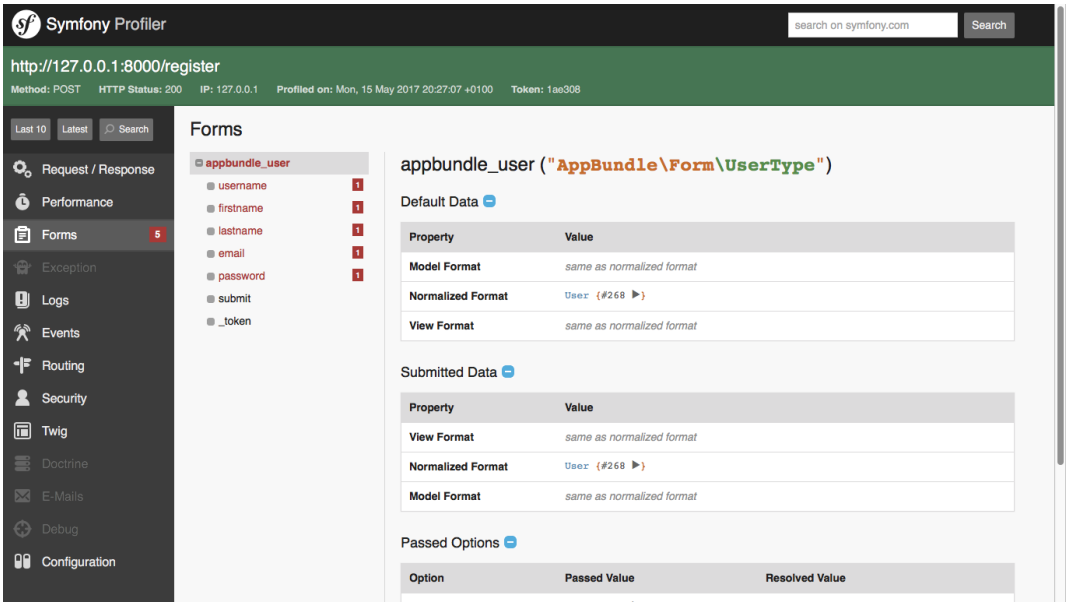

Figure 5.4: Profiler Error Register

5.3 Testing Registration

5.3.1 Successful Registration

Although the user enters their details into the form. Nothing might come of it. Which means the form was not handled correctly. This was why informing the user on the client side was important. It gives them some indication that the action which they performed had the potential to be a success figure 5.6.

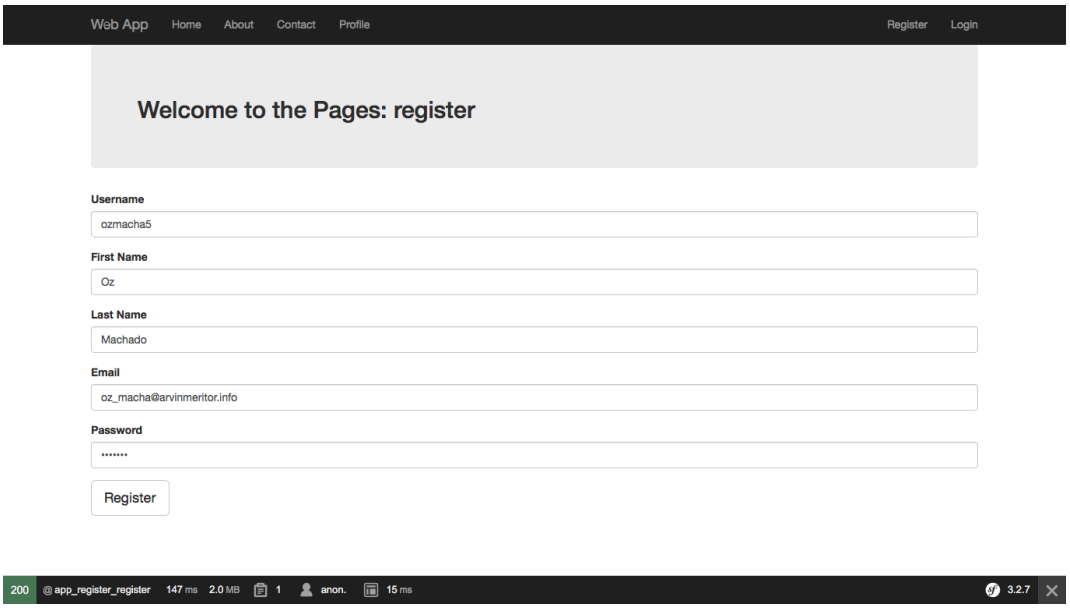

Figure 5.5: Register Test

On the server side it was possible to view what the user entered into the form 5.7 and give the role of ROLE USER in the role field.

5.4 Testing Login

5.4.1 Successful Login

In figure 5.8 this guarantees the user that it was a success as they are successfully logged in and authenticated.

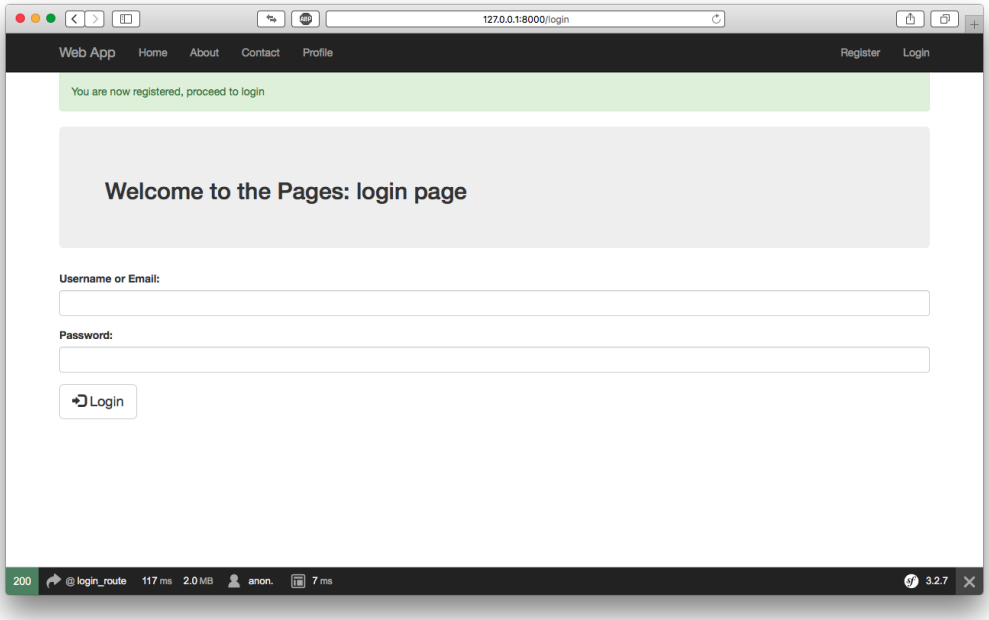

Figure 5.6: Successful Registration

As an administrator it was possible to view the recently added user to give confirmation that the application was working as it should.

5.5 Testing Editing

5.5.1 Successful Editing Function

In figure 5.12 it was clear that the edit method was functioning correctly. As the first name of the user was successfully changed to the Wizard of Oz.

5.6 Testing Password

5.6.1 Successful Password Editing

Here the users password was changed. This cannot be seen in the password view yet this change can be seen in the database in the password field. The encryption pattern on the

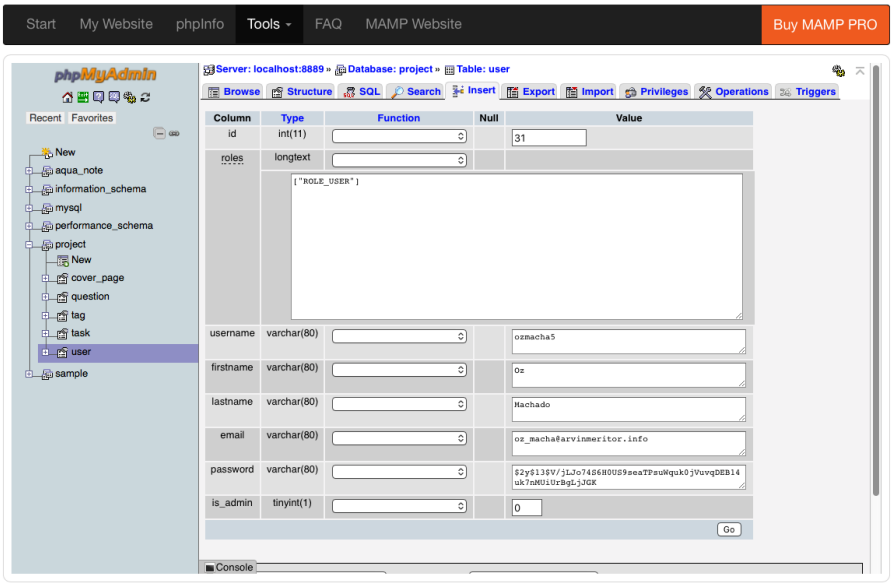

Figure 5.7: Database Check

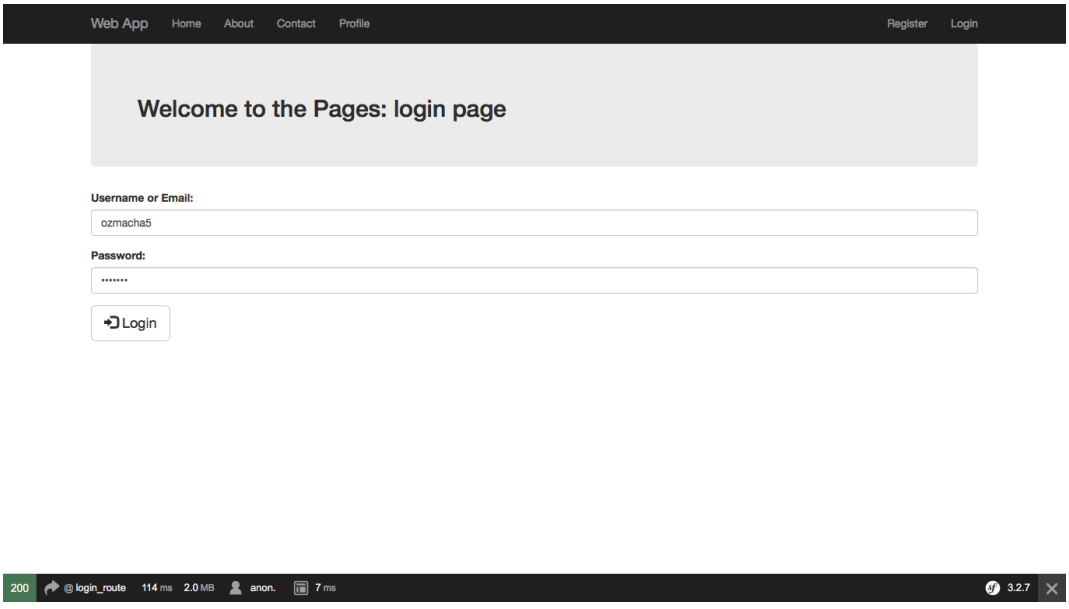

Figure 5.8: Mr Oz Machado Login

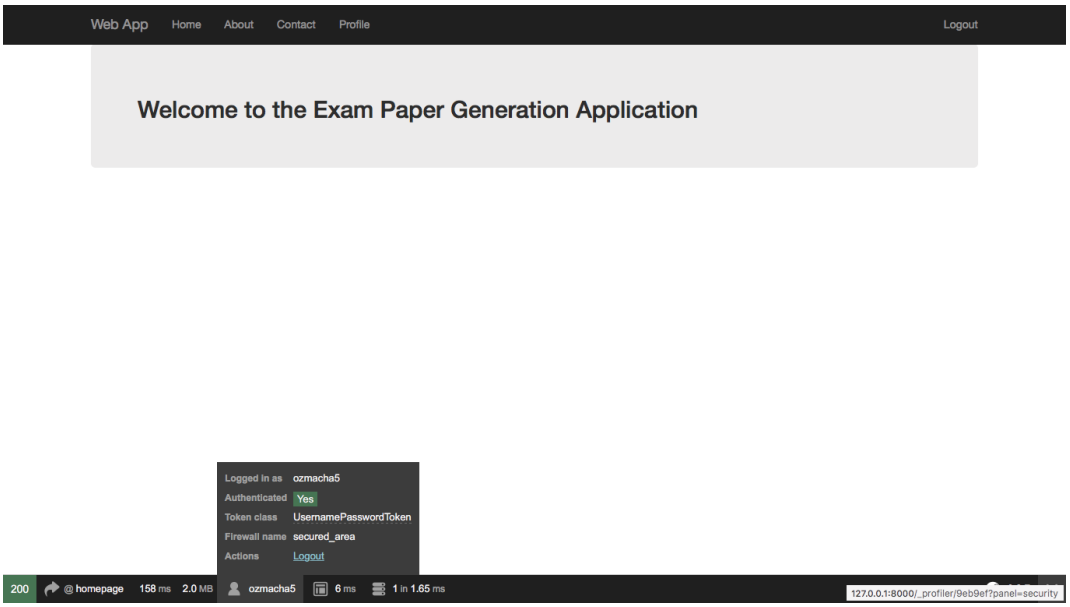

Figure 5.9: Oz Machado Authenticated

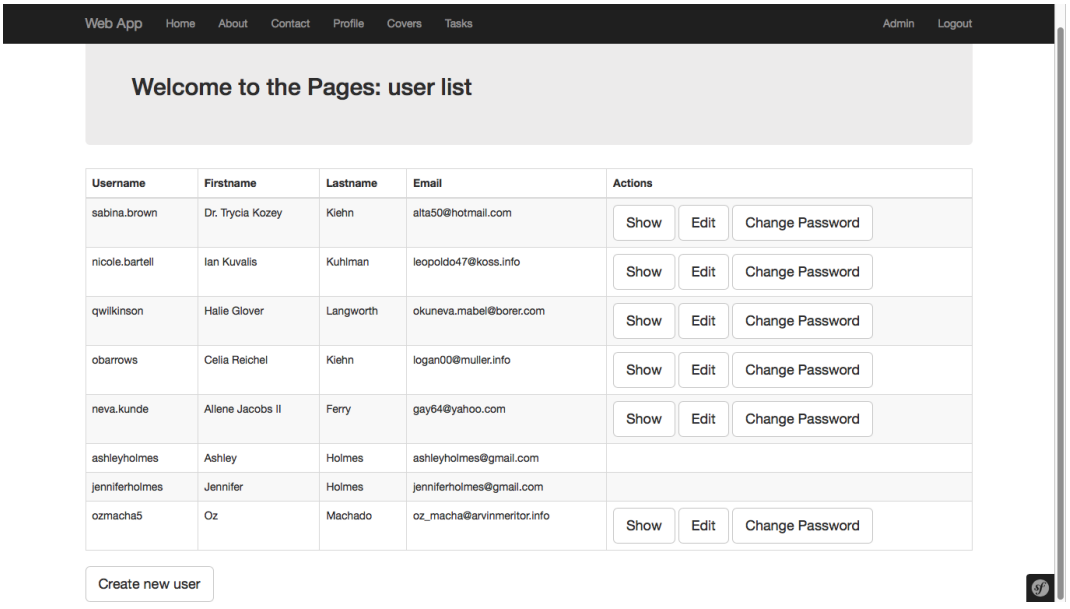

Figure 5.10: Oz Machado Index Page

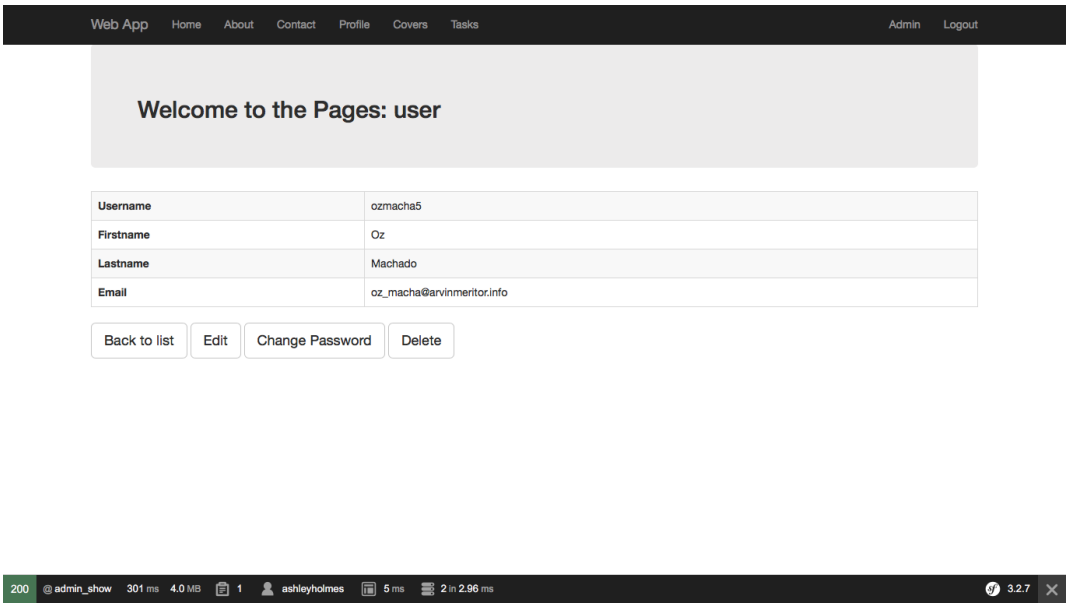

Figure 5.11: Oz Machado Show Page

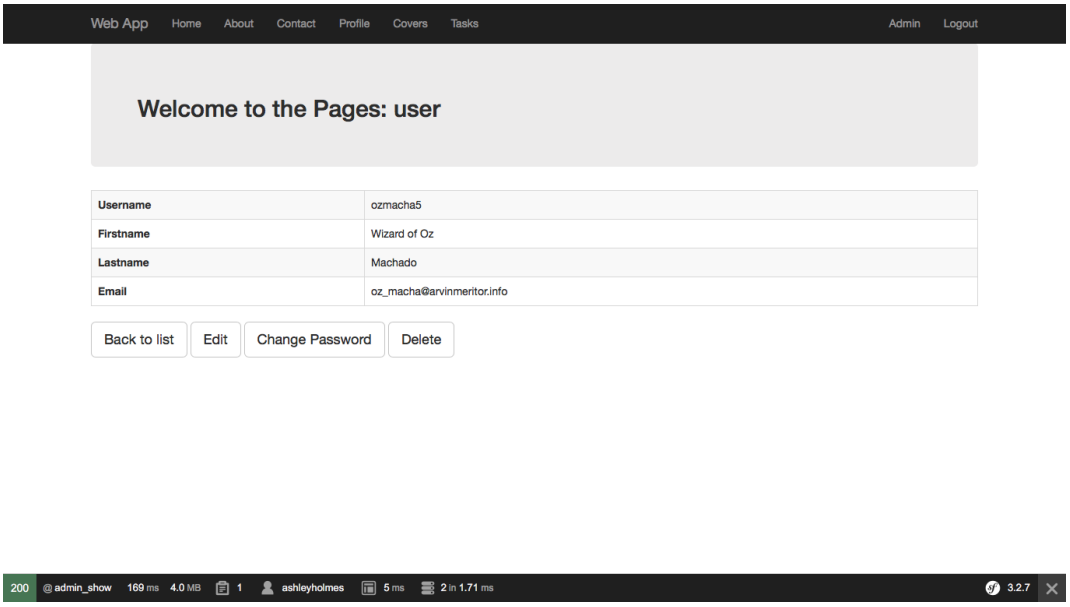

Figure 5.12: Oz Name Changed

password is different to the figure 5.7.

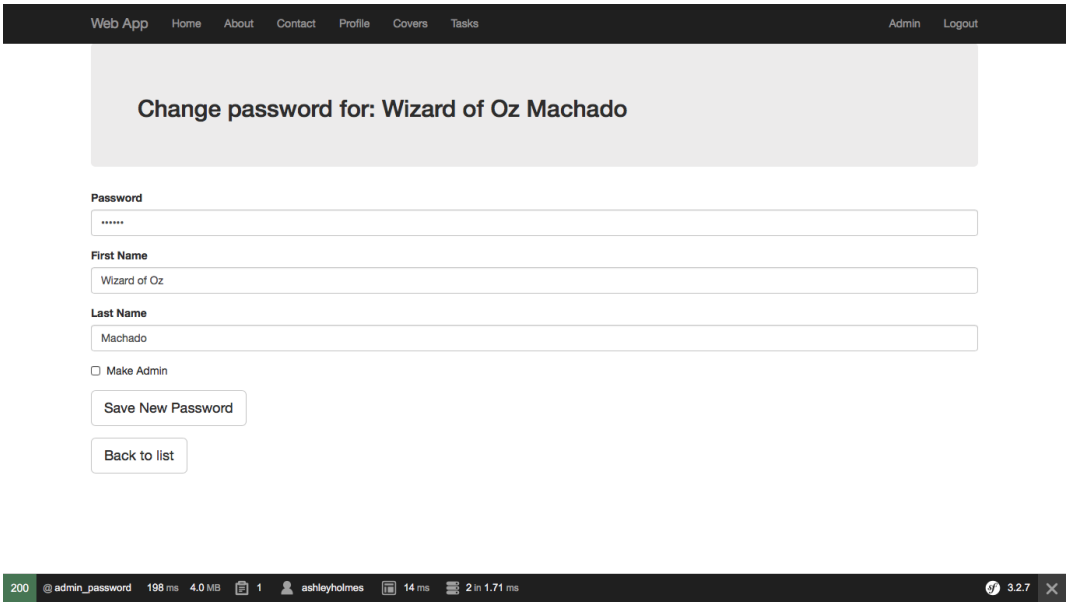

Figure 5.13: Oz Password Changed

5.7 Testing ROLES

5.7.1 ROLE Changes

Mr Oz Machado had his ROLE changed from a user role to an admin role which it seen in the roles field in figure 5.15. This operation was a success which was indicated by the change made in the database. Which gave him addition rights across the application.

5.8 Testing Exam Cover Page

5.8.1 Successful Form Validation

The form validation of the cover page was successful. The user was not able proceed after submitting the form without populating all fields. The errors are specified, explaining to the user what to follow with next. Once successfully completing the form the user was

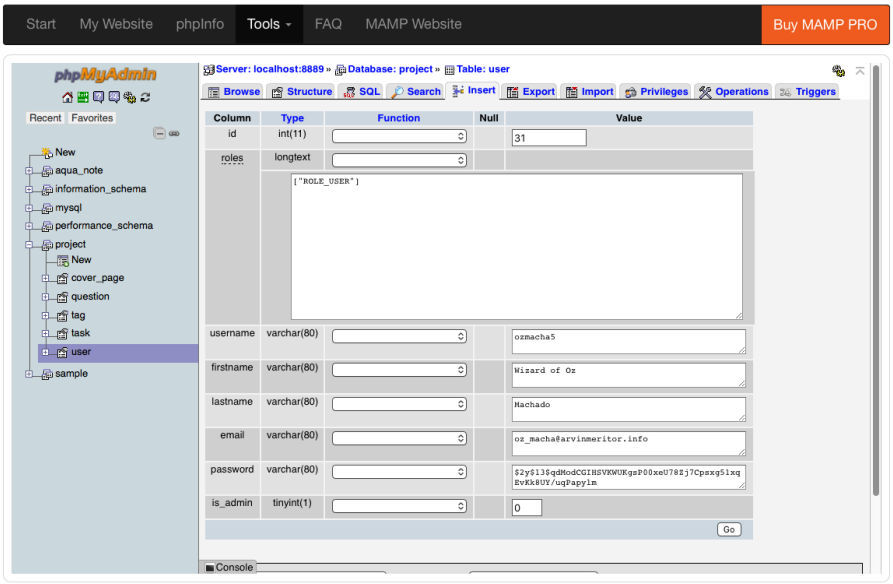

Figure 5.14: Oz Password Changed DB

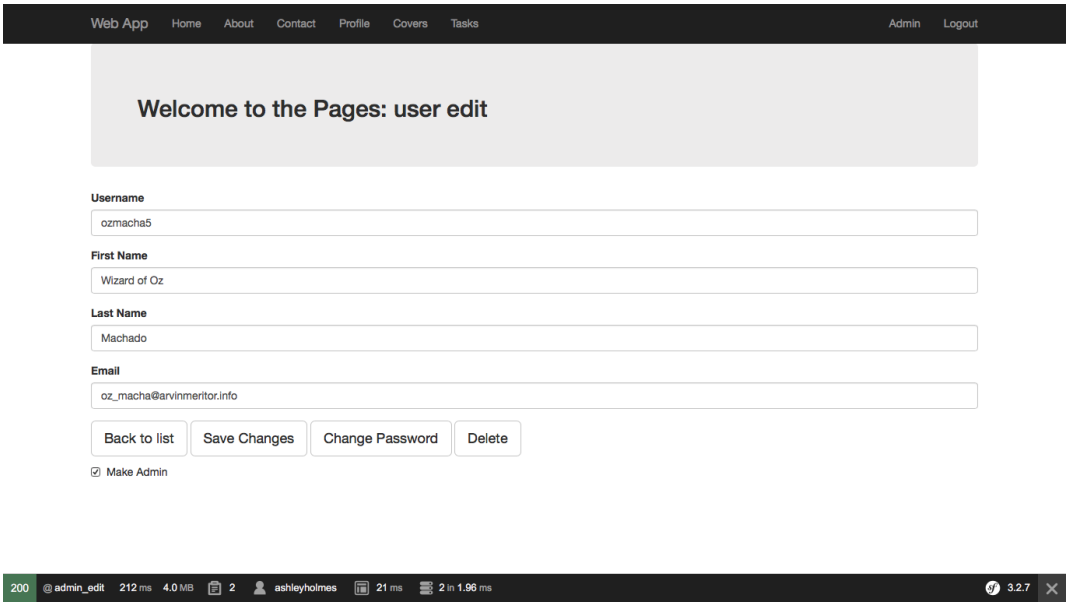

Figure 5.15: Oz ROLES Changed

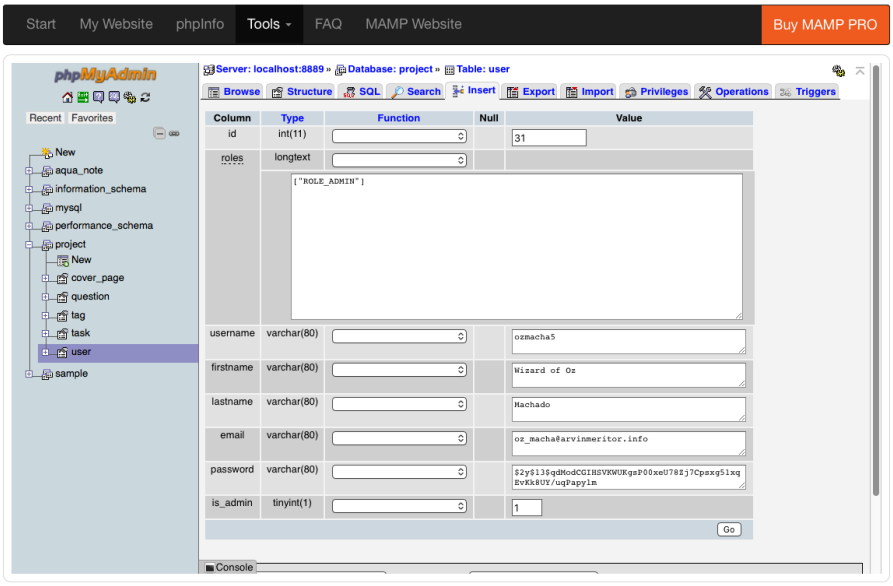

Figure 5.16: Oz ROLES Changed in DB

directed back to the admin and the information transferred to the database successfully. The indication of this are in figures 5.17, 5.18, 5.19, 5.20, 5.21, 5.22 and 5.28.

5.9 Testing Question Page

5.9.1 Successful Form Validation

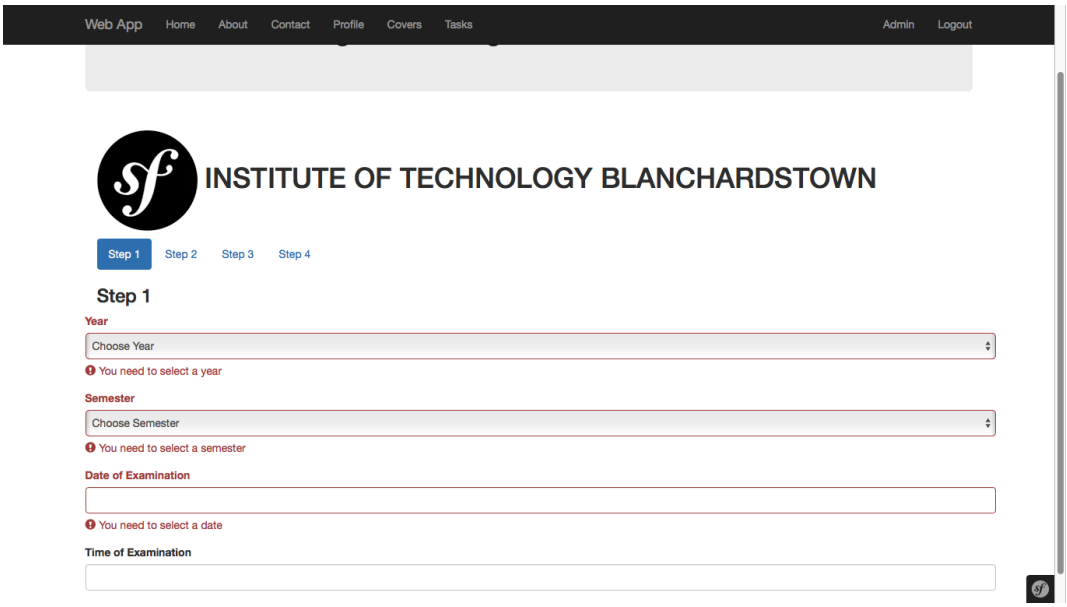

Figure 5.17: Cover Form Validation

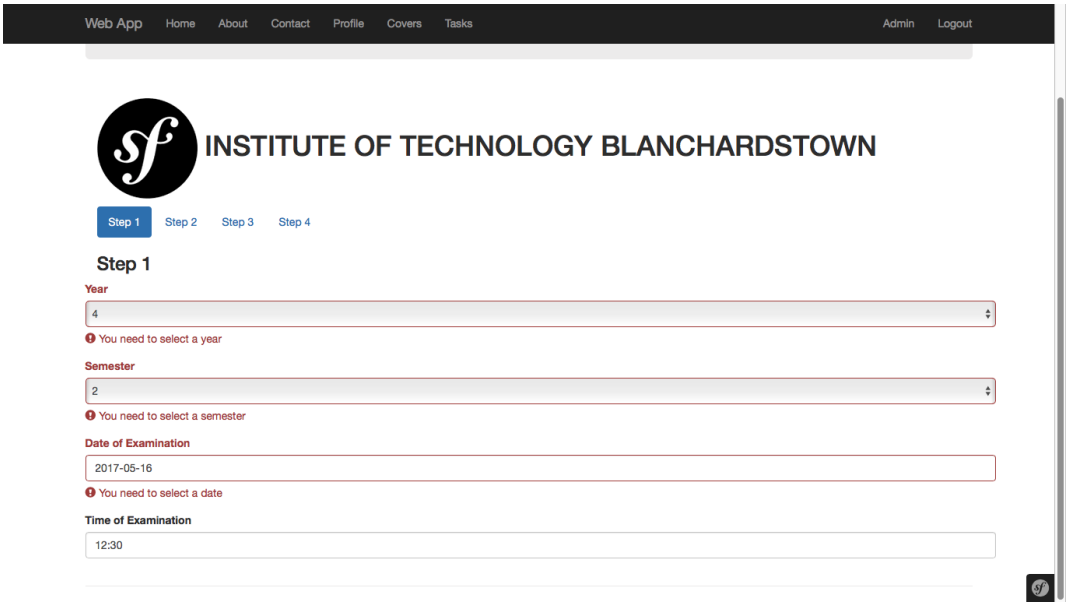

Figure 5.18: Cover Generation Step 1

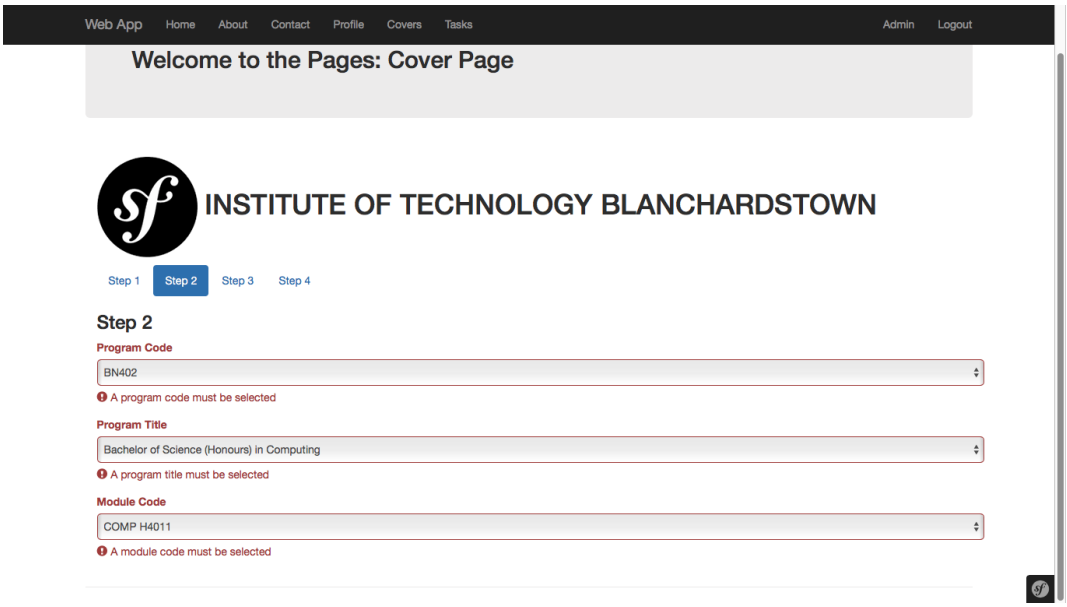

Figure 5.19: Cover Generation Step 2

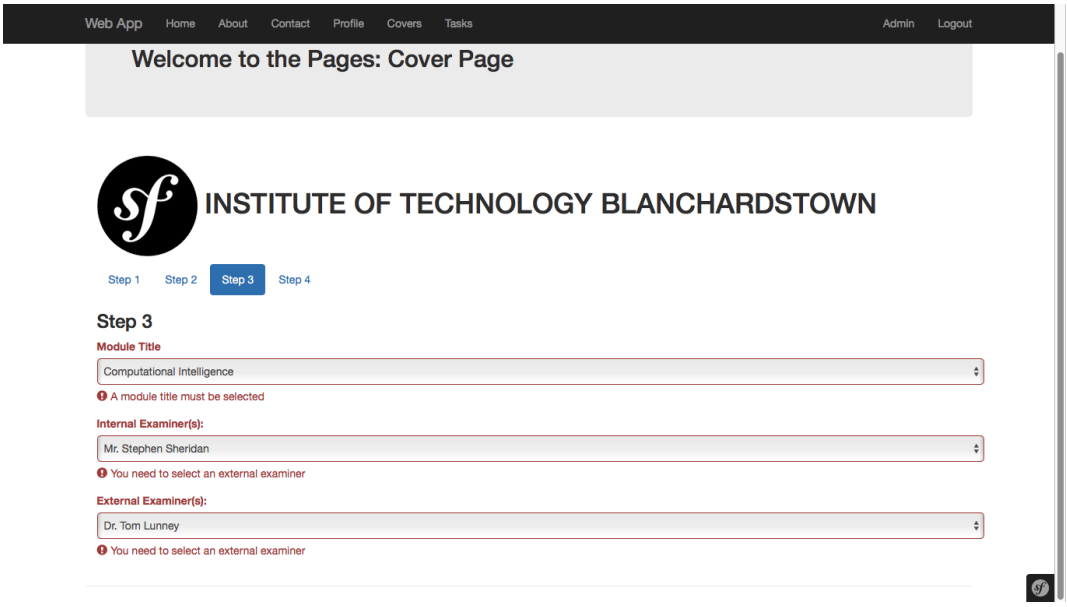

Figure 5.20: Cover Generation Step 3

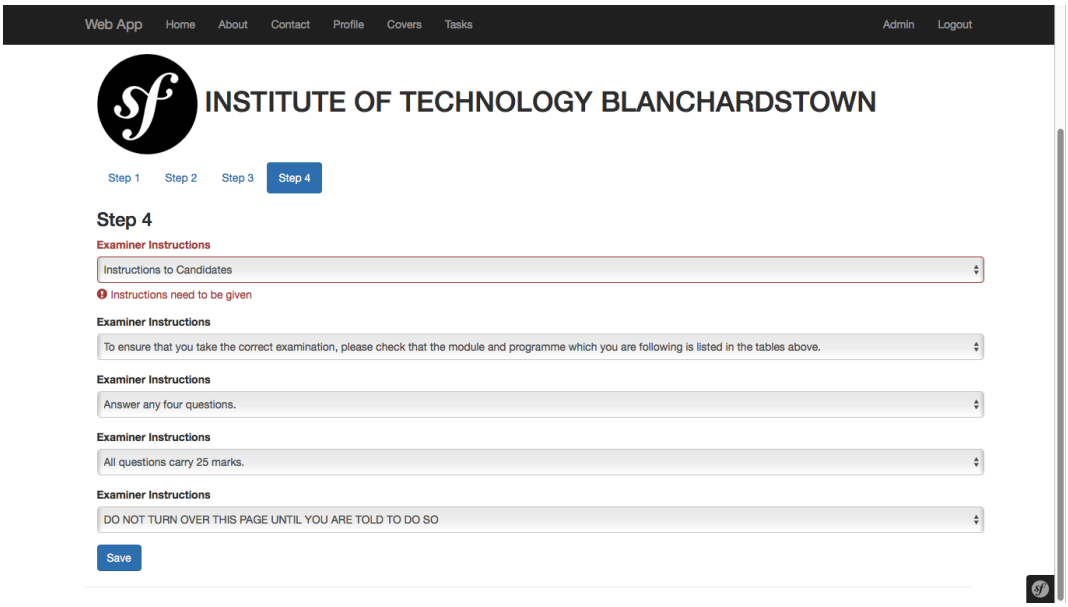

Figure 5.21: Cover Generation Step 4

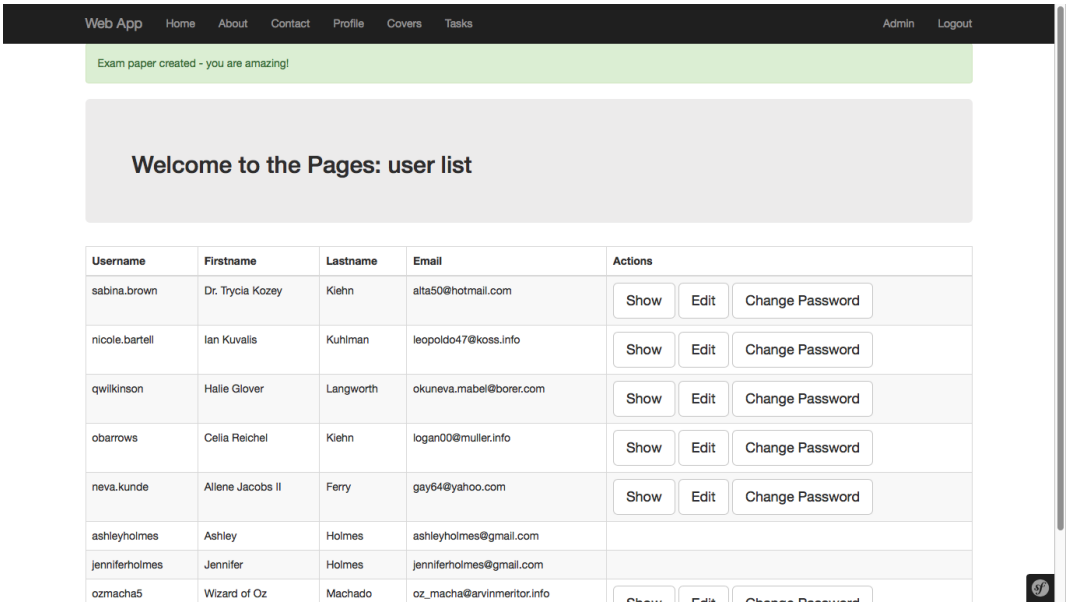

Figure 5.22: Cover Page Successful

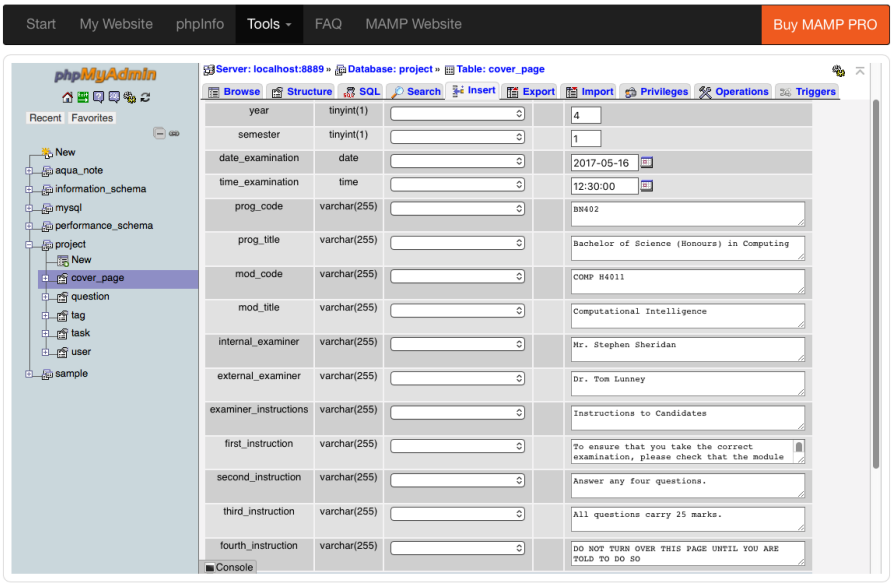

Figure 5.23: Cover Page Successful DB

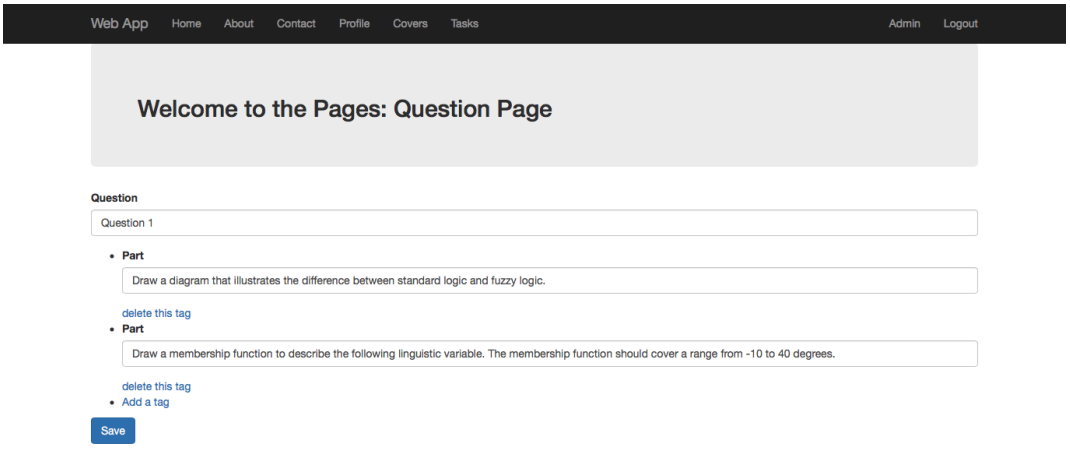

Figure 5.24: Question Page Successful

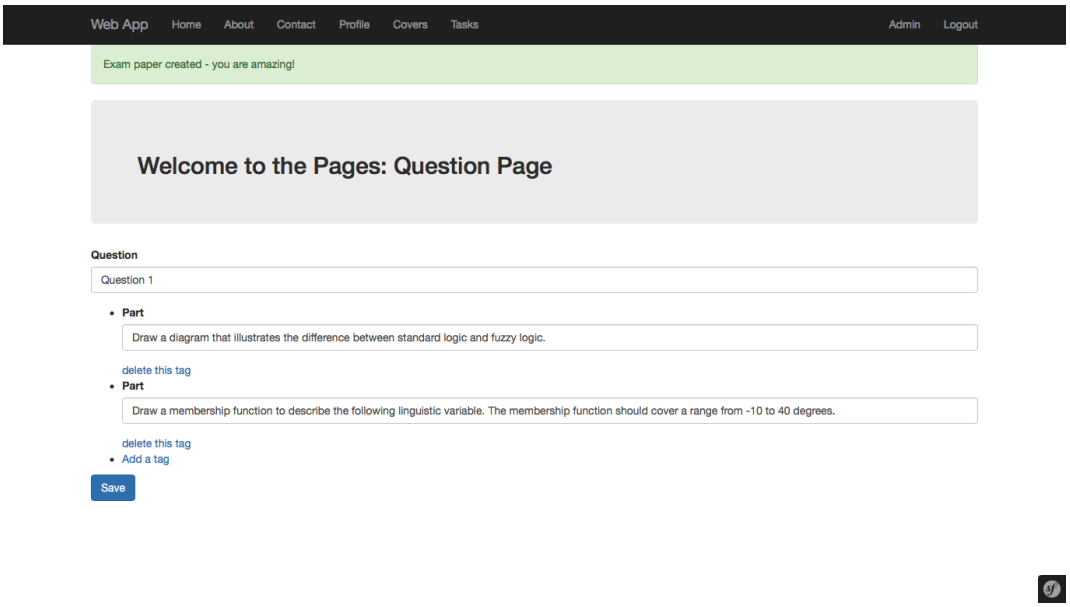

Figure 5.25: Question Page Successful DB

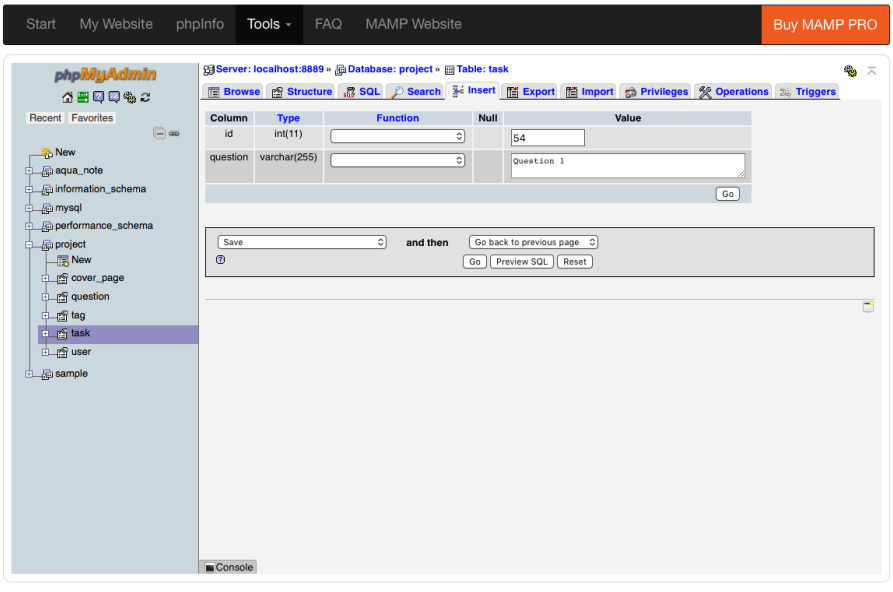

Figure 5.26: Question Page Metadata

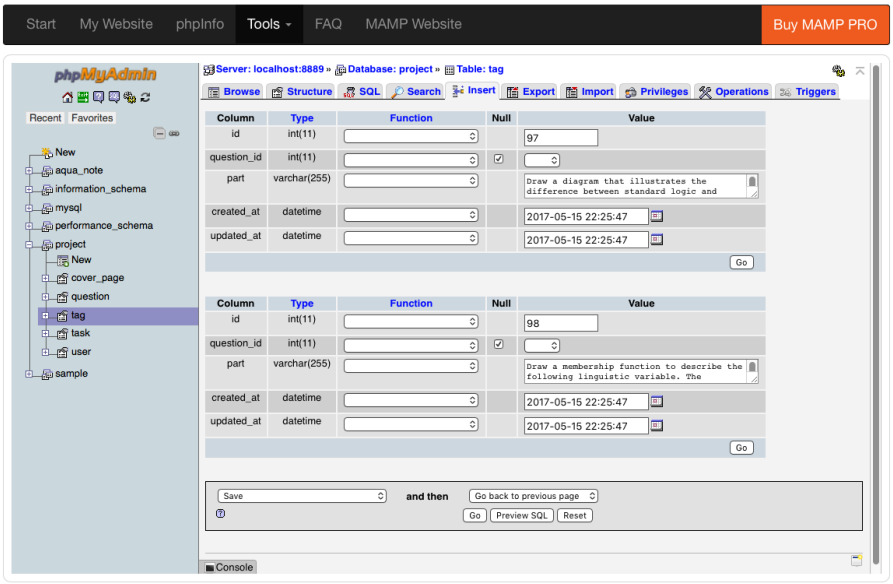

Figure 5.27: Cover Page Successful DB

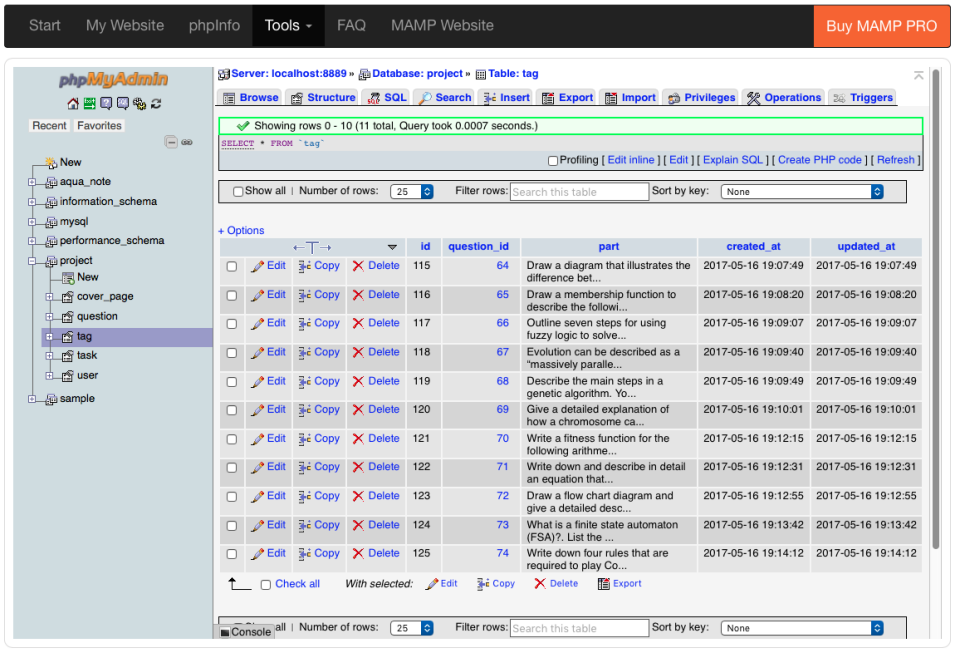

Figure 5.28: Metadata from database

Chapter 6

Implementation of the System

6.1 Implementation Principles

6.1.1 Object-Oriented Approach

How OO aided the system

6.1.2 Design Patterns

MVC withing the Netbeans IDE

6.1.3 Choice of Language

Java over PHP

6.2 Stages of Admin Implementation

6.2.1 Login

Discuss the login procedure

6.2.2 Administration

Discuss the Administation side

6.2.3 Subsection header 3

fdsdfsdfds

6.3 Stages of User Implementation

6.3.1 Subsection header 1

fdsdfsdfds

6.3.2 Subsection header 2

fdsdfsdfds

6.3.3 Subsection header 3

fdsdfsdfds

6.4 Design

6.4.1 Subsection header 1

fdsdfsdfds

6.4.2 Subsection header 2

fdsdfsdfds

6.4.3 Subsection header 3

fdsdfsdfds

Chapter 7

Testing and Evaluation

7.1 Introduction

Introduction into the testing

7.2 Tests Conducted

This will include do the tables work, checking encryption etc. Storing the data.

7.3 Algorithms

Algorithms used to randomise the tables. And colony, traditional. The FisherYates shuffle. The Knuth FisherYates shuffle

7.4 Summary

Summary of findings

Chapter 8

Conclusion and Future work

8.1 Contributions

Contributions of this project towards Faculty and the affect on the student

8.2 Limitations

Limitations of project

8.3 Future Work

Integrate into an undertaking currently being deployed at DCU called GURU

8.4 Data Collection

Data analysis performed and exploration. As to who has submitted their examination papers

Bibliography

Student Universal Support Ireland. online, https://susi.ie/, (last accessed: October 2016), 2016

Research Questions. online, http://www.twp.duke.edu, (last accessed: October 2016), 2016

Agile - Methodology. online, http://www.agilemethodology.org, (last accessed: October 2016), 2016

tutorialspoint - SDLC - Agile Model. online, http://www.tutorialspoint.com, (last accessed: October 2016), 2016

Central Statistics Office - Unemployment Rate. online, http://www.cso.ie, (last accessed: October 2016), 2016

European Social Fund - Investment Funds Programme. online, http://www.esf.ie, (last accessed: October 2016), 2016

Citizens Information - Third level places for unemployed people. online, http://www.citizensinformation.ie, (last accessed: October 2016), 2016

Guang Cen, Yuxiao Dong, Wanlin Gao, Lina Yu, Simon See, Qing Wang, Ying Yang, Hongbiao Jiang A implementation of an automatic examination paper generation system. online, http://www.sciencedirect.com, (last accessed: October 2016), 2016.

Ramandeep Kaur, Shilpy Bansal A Review on various Techniques for Automatic Question Generation. online, http://www.ijettcs.org/, (last accessed: October 2016), 2016.

Rohan Bhirangi, Smita Bhoir Automated Question Paper Generation System. online, http://www.ermt.net/, (last accessed: October 2016), 2016.

Yvonne SKALBAN, Le An HA, Lucia SPECIA, Ruslan MITKOV Automatic question generation in multimedia-based learning. online, http://www.aclweb.org/, (last accessed: October 2016), 2016.

Kapil Naik, Shreyas Sule, Shruti Jadhav, Surya Pandey Automatic Question Paper Generation System using Randomization Algorithm. online, www.erpublication.org, (last accessed: October 2016), 2016.

Sandeep Singh Yadav, Mandeep Singh Yadav Development of System for Automated & Secure Generation of Content. online, http://www.mecs-press.org/, (last accessed: October 2016), 2016.

Surbhi Choudhary, Abdul Rais Abdul Waheed, Shrutika Gawandi, Kavita Joshi Question Paper Generator System. online, http://www.ijcstjournal.org/, (last accessed: October 2016), 2016.

agile in a nutshell - What is Agile. online, http://http://www.agilenutshell.com/, (last accessed: December 2016), 2016

Appendices

Appendix A
Web Application Login Screen

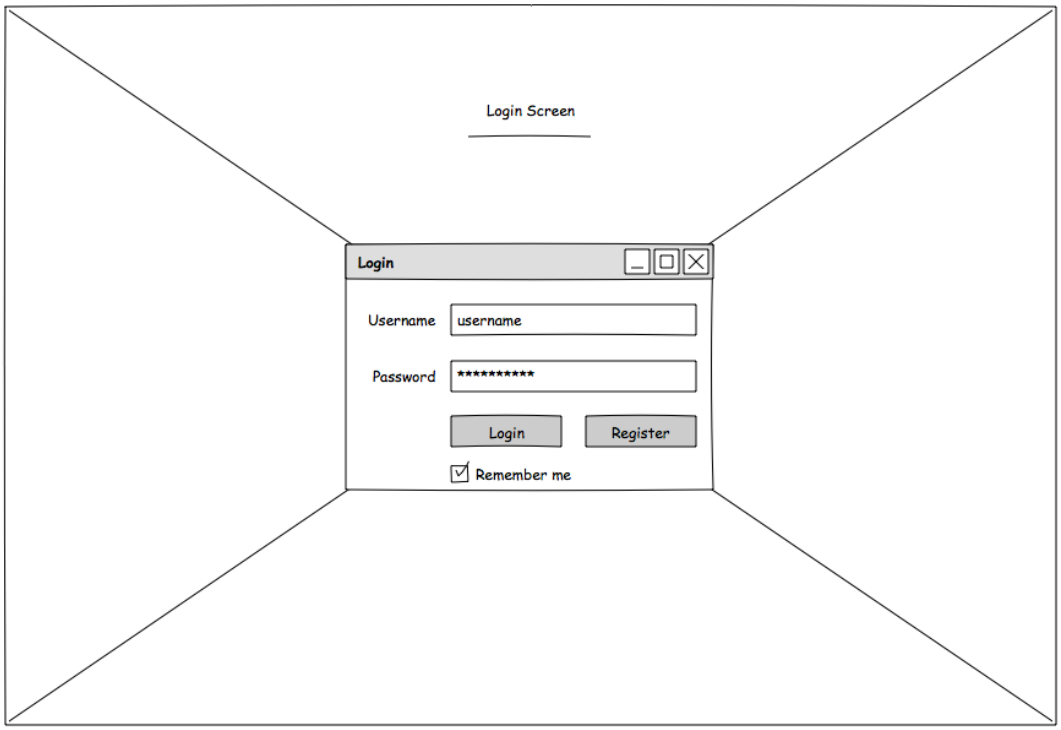

Figure 1: Graphical illustration of the Login Screen

Appendix B
Web Application Question Entry

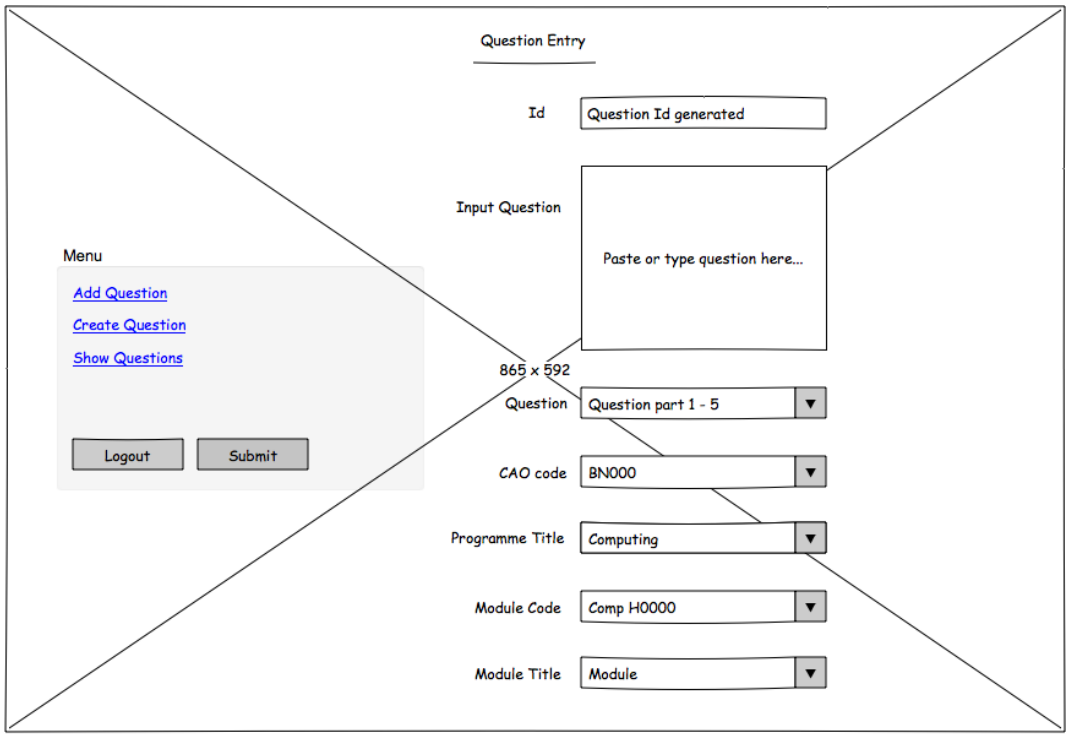

Figure 2: Graphical illustration of the Question Entry

Appendix C
Generate a Question

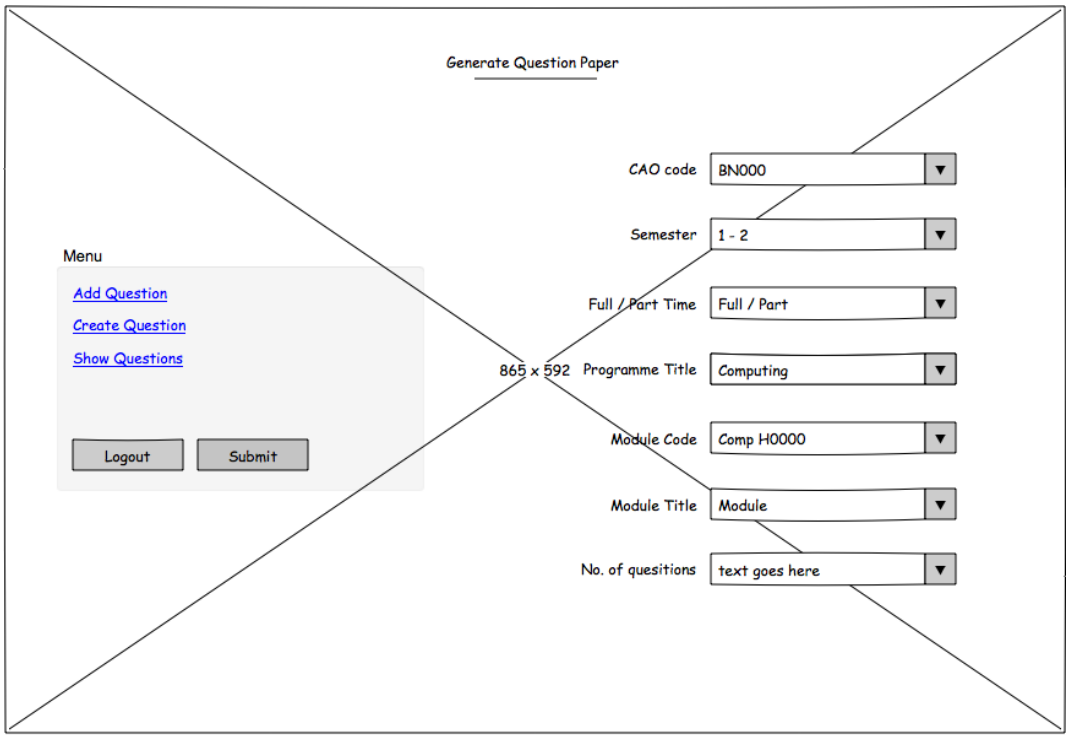

Figure 3: Graphical illustration of the Generate a Question Paper

Appendix D
Show questions

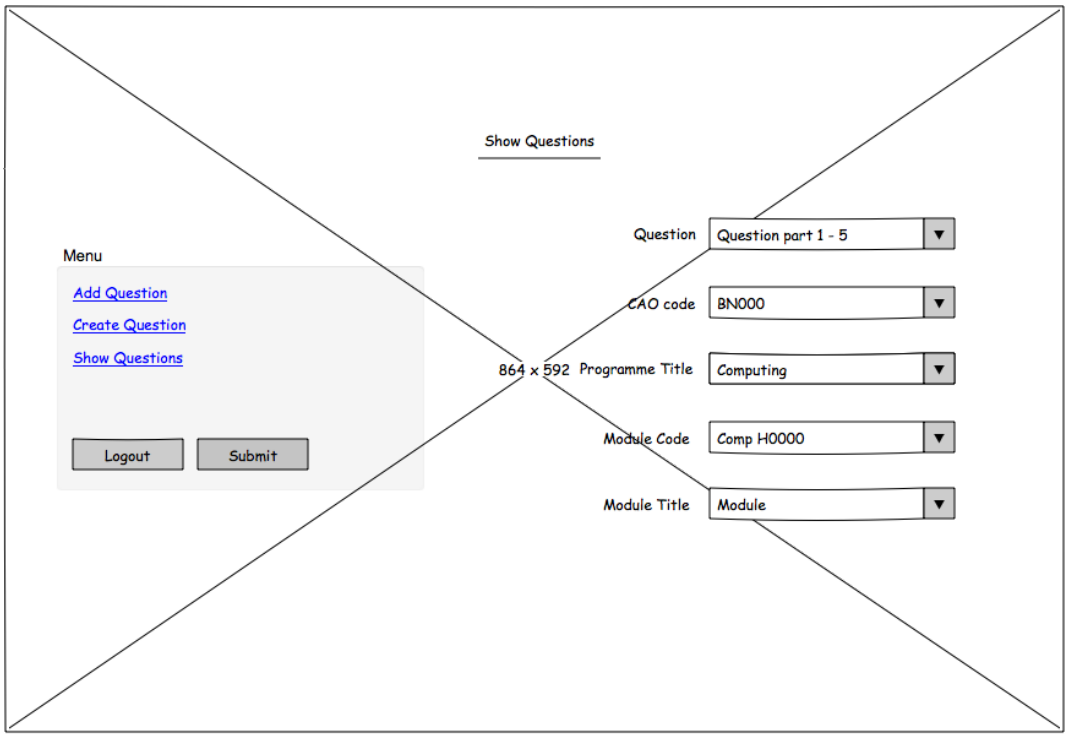

Figure 4: Graphical illustration of the Menu List to view the Questions

Appendix E

Show

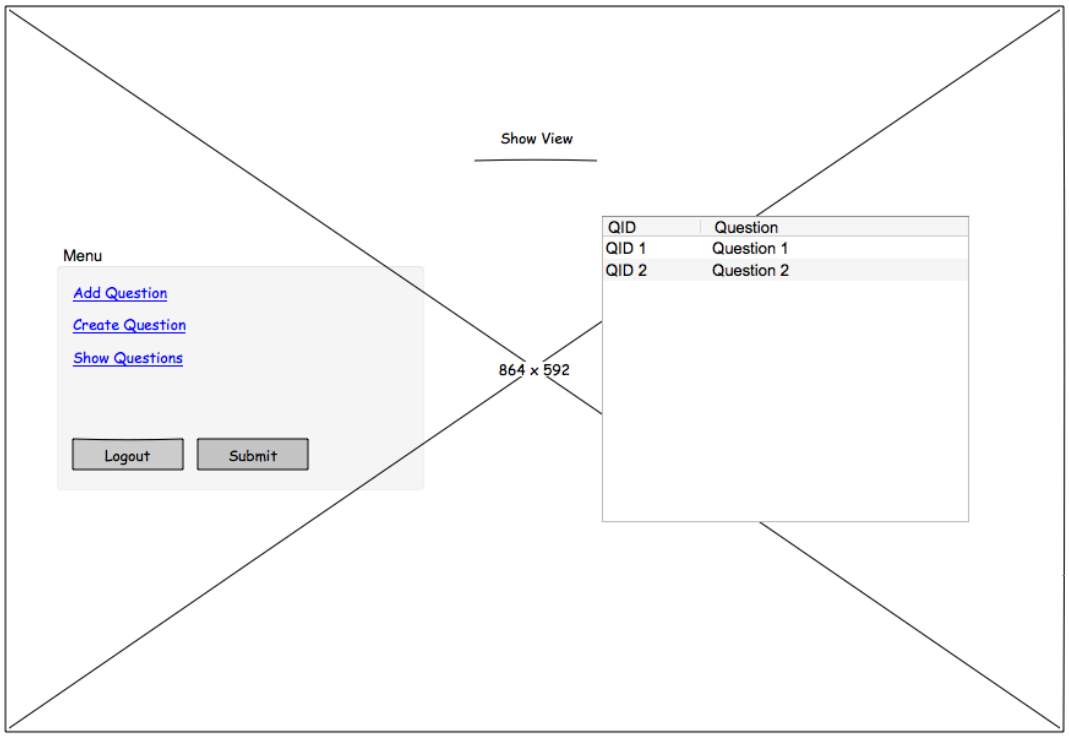

Figure 5: Graphical illustration of the Show list