HEALTH CENTRE IRELAND - AN APP DEVELOPED IN SYMFONY 3

Institute of Technology Blanchardstown

Fourth Year Computing Project

Individual Project

by

Dylan Byrne

Submmitted in part fulfillment for the degree of

B.Sc in Computing in Information Technology

School of Informatics and Engineering,

Insititute of Technology Blanchardstown,

Dublin, Ireland

April 2018

DECLARATION

I herby certify that this material, which I now submit for this assessment on the program leading to the award of Bachelor of Science in Computing in Information Technology 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 fulfillment of the requirements of that stated above

Signed____

Dated____

Acknowledgements

Thanks to …

My project Supervisor Aoife Fox.

My friends Adam Horan, Owen Flannery, Karl Jones, James Plunkett, Ross McMahon, Sean Grennan, Thomas Daly and Danut Hij for all the support and friendship they have given me throughout my years in college.

My girlfriend Kim Brady, who has shown nothing but support and patience to me this past year.

My mother, father and all my family for helping me throughout all my college terms, I could not have done it without them.

And finally, thank you to all of the lecturers from The Institute of Technology Blanchardstown for the resources and help they have given to me throughout the four years of my education.

ABSTRACT

Table of Contents

DECLARATION **i**

Acknowledgements **i**

ABSTRACT **i**

1 Project Introduction **1**

1.1 Overview of Project . . . 1
1.2 Project Objectives & Goals . . . 2
1.3 Main Research Questions . . . 2
1.4 Technologies Used . . . 3
1.5 Methodologies . . . 3
1.5.1 Choose Area and Project . . . 4
1.5.2 Become Well Antiquated with the Literature . . . 5
1.5.3 Decide on and Implement Methodologies . . . 5
1.5.4 Produce Results . . . 5
1.5.5 Write up Methods, Results, Conclusions, Implications . . . 5
1.6 Expected Results . . . 5
1.7 Overview of Report . . . 6

2 Literature Review/Research Conducted **7**

2.1 Introduction . . . 7
2.2 Review Of Literature . . . 8
2.2.1 Lung Cancer Exercise Program . . . 8
2.2.2 Rheumatoid Arthritis Exercise Methods . . . 9
2.2.3 Dealing With Depression . . . 9
2.2.4 Exercise Training in Alzheimer's . . . 11
2.2.5 Type 1 Diabetes Treatments . . . 12
2.3 Related Work . . . 12
2.3.1 Relevance . . . 13
2.3.2 Success/Effectiveness . . . 13
2.3.3 Similarities . . . 13

2.3.4 Summary of Results . . . 14

3 System Analysis 15

3.1 Functional Requirements . . . 15

3.2 Non Functional Requirements . . . 16

3.3 Use Case Diagrams . . . 16

3.3.1 Registration . . . 17

3.3.2 Log in . . . 18

3.3.3 Delete Account . . . 18

3.3.4 Reset Password . . . 18

3.3.5 Specified Illness Index . . . 19

3.3.6 Disease Information & Exercises . . . 19

3.3.7 Society Information . . . 20

3.3.8 Useful Numbers . . . 20

3.3.9 Calendar . . . 20

3.3.10 Create Event . . . 21

3.3.11 Update Event . . . 21

3.3.12 Delete Event . . . 22

3.3.13 Message Forum . . . 22

3.3.14 Create Message . . . 23

3.3.15 Delete Message . . . 23

3.4 Class Diagrams . . . 24

4 Implementation and Design of System 25

4.1 Overview of Implementations . . . 25

4.2 Implementation Components . . . 26

4.2.1 Database Entities . . . 26

4.2.2 User Registration . . . 35

4.2.3 User Login . . . 39

4.2.4 Log Out . . . 41

4.2.5 Delete User . . . 42

4.2.6 User Calendar/Scheduler . . . 43

4.2.7 User Blog/Messaging System . . . 52

4.2.8 Implementing Routes based on user credentials . . . 55

4.3 Graphical Components and User Interface . . . 57

4.3.1 Bootstrap . . . 57

4.3.2 Base . . . 57

4.3.3 Forms . . . 59

5 Testing & Evaluation 65

6 Evaluation & Self Reflection 67

7 Conclusions & Further Work **69**
7.1 Conclusions 69
7.2 Further Work 69

I Appendices **71**

List of References **73**

1
Project Introduction

1.1 Overview of Project

The following project which was undertaken is an online web application developed using the Symfony 3[1] framework. Symfony is a PHP based web application framework, with a set of reusable PHP components and libraries which first published in 2005. Other web applications which use this framework include; prestshop.com[2], sonota-project.org[3] and pimcore.com[4]. The purpose of this web application is to create an application for "Health Centre Ireland", which is targeted for users who are suffering from an illness, whether it be a physical illness or a mental illness. The idea for this project came when reading articles about Mental Health in Ireland and how there is a lack of general knowledge in the media and throughout the public when it comes to this affliction. So the idea of developing an app, which could both be informative, and used as an assistant to people suffering from this disease. The idea for including other diseases came from wondering if the public had knowledge about other illness which effect people in the country, and deciding to incorporate these other illness' into the application also.

There are many services and charities which offer help to people suffering with these illness', and this web application will act as a hub where users can find out information relating to their illness, find out exercises which they can partake in to potentially help or relive pain relating to their specified illness. This application can also be used as an assistant, to potentially help anyone who suffer with these diseases' in a similar vein to previous assistant applications such as Pill Reminder[5] and MedCoach[6], two mobile applications for both Android and iOS which remind users when to take medication. This application will include a calendar which users can set these kind of reminders, along with reminders of when to do exercises that can potentially relieve pain for their illness. Users signed up to this application should also have a method of communication, which can be used to talk about their personal experiences in battling with illness, and medication which they have taken to other users on the site.

1.2 Project Objectives & Goals

The main objectives and goals for the project are as follows.

- Research exercise methods which can be used to relive pain of diseases and if these methods are proven to be effective
- Create a web based application "Health Centre Ireland" to assist anyone who is suffering from physical or mental illness
- Make sure that the application is ascetically pleasing, easy to use, along with having a good responsive design.
- Interaction with the app should be straight forward and every user should know exactly what to do on each page, along with what each page is for
- Implement functionality for:
  - User Registration
  - User Login
  - Routing Security
  - A Scheduler for each User
  - A Communication System
- Create databases which store:
  - User Information
  - Scheduler Information
  - Information Regarding User Communication

1.3 Main Research Questions

When developing an application such as this which deals assisting a wide variety of people many questions must be asked in regard to the project, questions such as;

- How will this application be implemented to suit each individuals affliction?
- What illnesses and afflictions will be dealt with.
- How will any difficulty that arises during the research life cycle be dealt with?

- What topics must be researched to make this application as effective as possible for its target audience?
- Why use Symfony when implementing a project like this, as opposed to another web development framework?
- Are there any other similar technologies or applications in the marketplace right now?
  - If so, research these and find out what makes these applications so successful and try to improve on them

There are multiple ways in which these topics can be researched, such as the many books and articles relating to these matters that would make for excellent research material, some of the likes including; Cancer Fitness: Exercise Programs for Patients and Survivors by Anna L. Schwartz[7], Overcoming Arthritis: The Complete Complementary Health Program by Dr. Sarah Brewer[8], and for implenting the project and learning about the Symfony framework; An Introduction to Symfony 3 by Dr. Matt Smith[9].

1.4 Technologies Used

- Software Used for Server:
  - PHP built in Web Server
  - PHP
  - MySQL Database
- Software Used for Web Application Development:
  - PHP Storm
  - HTML
  - Twig(Template Engine)
  - JavaScript
  - CSS
  - MySQL Workbench
  - Symfony 3
- Device Used for Testing and Demonstration
  - Lenovo Z50-75(Windows 10, 8GB of RAM, AMD FX-7500 Radeon R7 Processor)

All of the required software is free of charge besides PHP Storm which requires a student subscription to be free of charge, otherwise it will cost 199€ for a year subscription.

1.5 Methodologies

Initially research will have to be conducted for the project in the area of Web Application Development, along with research for the illness' which the users can sign up with, such as how many people in Ireland suffer from these illness' and the exercises that can be done to relieve pain for these. The "Research Life Cycle"[10] will also be closely followed. The reasoning for this is so that focus is not lost at any point, and throughout the life cycle the current step as to what must be done is always known. By following the Research Life Cycle the implementation of the project will be much easier.

The following steps of the research life cycle will be taken into consideration;

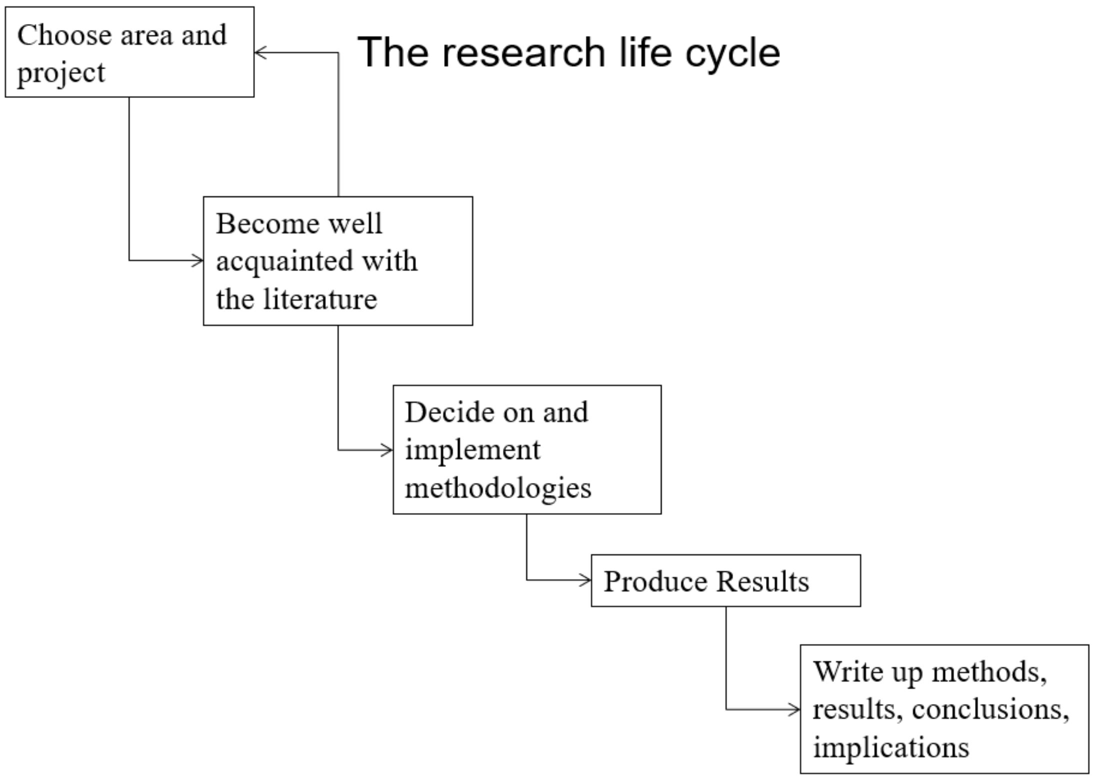

The research life cycle
Choose area and project
Become well acquainted with the literature
Decide on and implement methodologies
Produce Results
Write up methods, results, conclusions, implications

Figure 1.1: Research Life Cycle

1.5.1 Choose Area and Project

During this stage the project has been chosen. This includes the area of expertise for the project, in this case a web development based project. It is here where the system will be looked at as a whole, and whether or not the project is feasible and of a suitable standard for a fourth year based project. What type of research must be done to accomplish the project should be set out in this phase also.

1.5.2 Become Well Antiquated with the Literature

At this stage all the research will be conducted and you familiarise yourself with the literature which may help with the development of the project.

1.5.3 Decide on and Implement Methodologies

It is here where the project will start to be implemented and work on the project after all the necessary research is conducted and the developer is content with what the final version of the project will be. All of the assets involved with the application will be analysed and then this will form the final project.

1.5.4 Produce Results

The results of the research conduct will be produced and the final idea of what the project will be should be known, and the final implementations, along with the coding of the final project should be done here

1.5.5 Write up Methods, Results, Conclusions, Implications

This is the final stage, and this will include writing up methods used from the research that has been conducted, along with the results and conclusions came to from the research. From this point the coding, along with the deliverables of the software can be met in an effective and thorough way.

1.6 Expected Results

Once all the research has been conducted and the software is correctly implemented, what is expected is a fully functional dynamic web application, where users can register their illness, log in freely, be able to set reminders on exercises, or when they need to take medication, or if they have a doctors appointment. The users should also be able to find out information about their specified illness and view a live RSS social media feed of an organisations relating to their illness. They should also be able to communicate with fellow users on the application, through a messaging/blog type system. The user should also have wide access to a variety of information regarding their illness, such as useful websites, phone numbers, along with information on exercises they can do to help cope with the illness or disorder they may be facing.

1.7 Overview of Report

The contents of this report will show the research that has been conducted into this project, along a system analysis, the implementation and design of the system, the testing phase of the system, an evaluation of the project, along with a self reflection. The report will finish with conclusions drawn from the project and any further work which could be put into the project in the future to improve the application.

2 Literature Review/Research Conducted

2.1 Introduction

People have always had to deal with illness in a variety of ways. But like all things in today's modern society, and throughout history, everything evolves with technology and accessibility to medical care is now more convenient than ever. With the internet evolving in the way it has, along with social media too, it has become easier and easier to access valuable information, be it through research conducted online, or just simply by communicating with people through internet message boards, forums and social media. But with these advancements come negative implications too, a key example of this is being people who self diagnose online. According to Srini Pillay M.D[11] "In this day and age of limited time with doctors coupled with ample opportunity to Google anything, the temptation for people to reach their own conclusions about their illness is strong". He goes on to say how when people self diagnose online they are assuming that they know the subtleties that a diagnosis constitutes. He goes on to list the problems that can come with a self diagnosis. Some of the examples given are; self diagnosis in psychological syndromes, where you miss a medical disease that "masquerades" as a psychiatric syndrome and if a typical user is to miss the subtleties that a diagnosis constitutes, they may also miss the subtleties that come with the treatment of a diagnosis.

The purpose of this research is to show methods of treatments though exercises and therapy as conducted and researched by medical professionals and verified sources of information, this will be included in the final application to help users understand their illness better and be able to treat themselves as best they can through exercise or other methods, leading to no self diagnosis on treatments of their illness being necessary. This is not only beneficial for people with a physical illness, but also a mental illness.

2.2 Review Of Literature

2.2.1 Lung Cancer Exercise Program

Lung cancer is a very common form of cancer which effects both men and women in Ireland, with over 2300[12] people being diagnosed each year according to the Irish Cancer Society. This is a disease which can often leave people horribly impaired and having trouble doing even simple tasks, mainly due to issues with breathing. The journal which was researched to try and improve this is A Structured Exercise Program for Patients with Advanced Non-Small Cell Lung Cancer[13], with research conducted by Jennifer S. Temel, MD, Joseph A. Greer, PhD, Sarah Goldberg, MD, Paula Downes Vogel, PT, MS, Michael Sullivan, PT MBA, William F. Pirl, MD, Thomas J. Lynch, MD, David C. Christiani, MD and Matthew R. Smith, MD, Phd.

The main research questions proposed by the authors in this paper is what hospital-based exercise program is feasible in dealing with patients who have non-small cell lung cancer. The reasoning behind the research conducted in this paper is what exercise can be proven to improve symptoms for certain cancer populations, but it is unknown whether or not these exercises are feasible within lung cancer. For the methodology technique used to accurately conduct this study, patients that were within 12 weeks of diagnosis were eligible to take part in it, while patients who had any type of cardiac disease including congestive heart failure were deemed ineligible. The tests were carried out with a senior physical therapist and vital signs of the patients were monitored/ These tests included a 6 minute walk test, in which the distance a patient could walk within a 6 minute time period, along with a muscle strength exercise which included upper body exercises such as flexing of the shoulders, elbows and extending the elbows too. Lower body exercises were also carried out these included hip extensions, hip abductions and knee extensions. These sessions would be carried out twice every week over the course of two months.

According to the studies conducted in this paper, it was recorded that 25 of the patients who took part in this research, 80% completed the initial evaluation. But overall only 44% of patients completed the exercise program due to progressions of the disease. But those who were able to complete the study in full showed a significant reduction in lung cancer symptoms and had no record of deterioration in the 6 minute walk test and and muscle strength test. Overall the research conducted was a success for the most part. While less than half of the patients who initially signed up for the research were able to complete it in full, which could be argued that this brings into question the full validity of the paper as these results could have potentially altered the final results of the paper as these patients showed the worst symptoms if the disease. But on the other hand it could be argued that the research was successful as the other half of the patients showed no signs of deterioration due to the disease, and actually showed an improvement in lung cancer symptoms. While an argument can be made for both sides that people showed signs of deterioration had no recognizable improvements to their health, meaning this was a failed study, while the other side could argue that without this study being done, similar deterioration could have took place in the other 44% of patients.

2.2.2 Rheumatoid Arthritis Exercise Methods

Rheumatoid Arthritis is a condition in which the joints in a persons body become inflamed. It is an unpredictable disease as it can occur in any person of any age group. More than 44,000 people in Ireland have this affliction with 70% of them being women[14]. The piece of literature which was researched in how to effectively deal with Rheumatoid Arthritis using exercise methods, is Evidence For the Benefit of Aerobic and Strengthening Exercise in Rheumatoid Arthritis by Christina H. Stenström and Marian A. Minor[15].

The main research question put forward by the authors of this paper are to investigate the evidence regarding the benefits of both aerobic and strengthening exercises in Rheumatoid Arthritis and is there any truth to these benefits? The methodology technique used to accurately conduct this study were randomized controlled trials used to investigate the effects of exercise to improve aerobic capacity and muscle strength in people diagnosed with Rheumatoid Arthritis were searched. The search results produced 208 articles. Of which 30 of these articles reporting on 26 randomised controlled trials remained. These articles were broken down further into 17 papers with 15 randomized controlled trials remaining. It was from these articles where the final results would be determined.

The exercises were performed twice weekly, with a maximum of twice daily in one particular study. Aerobic exercises had an intensity level of being set from moderate to hard, meaning that the patients heart rate varied from 50-85%[16] of the maximum measured heart rate. While the strengthening exercise programs were adjusted, starting with an intensity level of 30-50% and increasing to a maximum of 80%[17]. The studies concluded that in 5 cases there were improvements to aerobic capacity, while in 3 cases there was not, while 8 studies reported muscle function increase after the strengthening exercises, whereas in 6 cases there was not. From the studies conducted there is more than enough evidence to support the theory that aerobic and strengthening exercises benefit those with Rheumatoid Arthritis. As the majority of patents who took part in research did show signs of improvement in both aerobic and muscle function capacities, specifically 62.5% in aerobic and 57.14% in strengthening exercises. A negative of this paper is that most of the research conducted by the author is drawn from other third party sources and not studies done by the authors themselves, meaning they could not validate themselves the questions being asked.

2.2.3 Dealing With Depression

Studies show that many people who suffer with depression don't seek professional help. But why is this the case? The following paper; Belief in Dealing with Depression Alone: Results From Community Surveys of Adolescents and Adults by Anthony F. Jorm, Clair M. Kelly, Annemarie

Wright, Ruth A. Parslow, Meredith G. Harris and Patrick D. McGorr[18] delves into this question on why is their a lack of treatment and why is it better to deal with depression with outside help. The methodology used to conduct this research was a series of surveys sent which were sent out to 3998 Australian adults, with 1001 receiving a vignette of a person with major depression, these vignettes were randomised so respondents would receive a male or a female version. 1115 surveys were sent out to Australian adolescents, with 564 containing a vignette. Questions were also asked regarding the vignette to the respondents of this survey. A second survey was also sent out to adolescents in the Melbourne region of Australia. There were 1207 of these surveys sent out. The respondents to these surveys were asked a series of questions regarding the vignette, such as assessing the particular disorder shown, their beliefs about treatment and whether they felt if it was better to deal with depression alone. From the data collected, responses regarding the question about dealing with depression alone were marked as helpful, harmful and other.

The results obtained from the adult survey indicated that 13.2% believed that it would be easier to deal with depression alone, while 63.4% would believe it to be more harmful. Men believed that it would be more helpful to deal with it alone at 14.8% compared to women at 11.7% and less likely to be harmful at 56.8% compared to 69.6%. Also when asked what was wrong with the individual in the vignette, the group that believed in dealing with depression alone was significantly less likely to correctly recognize depression than the group which thought this would have been more harmful according to the studies done, with the results of these studies standing at 68% for harmful, and 53.6% for helpful. The results obtained from the adolescents survey were similar to the results obtained from the adults survey in regard to the question whether dealing with depression alone was harmful or helpful. In the first adolescent survey, 12.7% of people believed that dealing with depression alone was more helpful, while in the second survey 9.5% of people believed it was more helpful too deal with depression alone. Also similar to the adult survey was that in both adolescent surveys there was a difference between the genders opinions, with males believing that it was better to deal with depression alone at 16.1% compared to females at 9.1% for survey one, and 11.7% compared to 7.7% in survey 2. Males were also less likely to believe that it is harmful with 63.3% compared to females at 70.1% in survey 1, and 55.8% compared to 68.3% in survey 2. When asked what was wrong with the individual in the vignette, the group that believed it was more helpful to deal with depression were less likely to be able to recognize depression. With the results of survey 1 indicating; helpful at 41.4% and harmful at 61.6%, and a similar pattern appearing in survey 2 with helpful at 38.2%, compared to harmful at 47.6%.

Judging from the results of these surveys it indicates that the majority of people do believe that dealing with depression alone does more harm than good. The strong points of these studies is that they got their results from a large group of people, not only different genders, but of different age groups too. This led to a more accurate answer whether or not people believed getting outside help is a better way of dealing with depression therefore answering the question which in which the paper originally set out to answer. One of the weak points of this study is mentioned in the paper itself, which is that the surveys did not ask why did the participants in the survey hold the beliefs that they did in regard to dealing with depression alone, or not alone? So people citing this paper would not have an indication as to the reason why is dealing with depression alone, or not alone do more harm than good and vice versa.

2.2.4 Exercise Training in Alzheimer's

Alzheimer's disease is a chronic neurodegenerative disease that can cause a decline in a persons mental functions. It is a disease which progressively gets worse over time the older someone who is diagnosed with the disease gets. It can affect a persons memory, thinking, behaviour and language. Nearly 40,000 people in Ireland suffer from Alzheimer's disease[19]. A number of studies have also contributed Alzheimer's to a physical deterioration, reduced muscle mass, resulting in a higher risk of falls and severe injuries[20]. The following journal; Exercise Training is Beneficial for Alzheimer's Patients by E. Santana-Sosa, M. I. Barriopedro, L. M. López-Mojares, M. Pérez, A. Lucia[21] researches into how can resistance exercise and training prevent these problems and lead to increased muscle mass and strength, along with higher endurance and fitness levels in Alzheimer patients, therefore making it easier for people diagnosed with this disease to perform "activities of daily living". To conduct this study; 16 patients were chosen, 10 female and 6 male. They were assigned to either a training or a control group.

A series of tests were done on the patients including a senior fitness test, this test was used to evaluate the functional capacity of the patients testing their muscle dynamic strength for the endurance of their legs. This was accomplished by doing a 30 second chair stand test. The patients upper body was also tested, an arm curl test using 5 pound weights for women and 8 pound weights for men was used to accomplish this. Flexibility of the upper and lower body was also tested using a chair sit-and-reach test and a back scratch test. Also tested were speed, agility, endurance, balance while moving and aerobic endurance, this was done through an 8-foot-up-and-go exercise and a 2-minute-step-test. The intervention included 36 programmed training sessions, each 75 minutes, and done over a period of 12 weeks. Each session started with a 15 minute warm up period and a 15 minute cool down period. Patients assigned to the control group did not perform in any physical activities.

No significant difference was found between either group at the start if the tests. By the end of the twelve weeks a significant difference was found for post intervention values in both chair stand and arm curl tests, whereas no difference was found in the control group. No significant difference was found between either group at the start of the lower and upper body flexibility test, and while no significant difference was found between the groups after the tests, there was a significant time and interaction effect that existed. In the training group for the 8-foot-up-and-go and 2 minute step tests revealed that post intervention values were significantly lower and higher. The final results also showed that post intervention training values were much improved in the training group as opposed to the control group when it came to performing activities of daily living.

Overall it is no surprise that daily exercise can help reduce the effects of muscle deterioration as associated with Alzheimer's disease, and this study did a perfect job showing the effects of exercise over an extended period of time as shown by the significant improvements to the patients performance of activities of daily living according to the authors. This can lead to reduced injuries of patients with Alzheimer's when it comes to performing mundane tasks around their home, especially if unsupervised. The authors also answered their original question on whether these daily exercises can help Alzheimer patients regain muscle mass, while increasing the endurance and fitness levels, and the studies done proved this.

2.2.5 Type 1 Diabetes Treatments

Type 1 diabetes is a metabolic disorder which tends to occur in childhood or early adult life. In this form of diabetes little insulin is produced, which results in high blood sugar levels and it is required that it is treated with insulin injections. There are an approximate 14-16,000 people in Ireland with type 1 diabetes which accounts for 10-15% of the total diabetes population in Ireland. On top of this it is estimated that 2,750 of these people with this condition are under the age of 16[22]. The following paper; Intensive Diabetes Treatment and Cardiovascular Disease in Patients with Type 1 Diabetes by the Diabetes Control and Complications Trial along with Epidemiology of Diabetes Interventions and Complications (DCCT/EDIC) Study Research Groups ask the question, can intensive therapy aimed at achieving near normal levels of blood sugar reduce the risk of micro-vascular and neurological complications that come with with type 1 diabetes. To see if this was possible 1441 patients ages 13-40 were chosen to take part in this study. These patients were randomly assigned to intensive or conventional therapy. Intensive therapy consisted of three or more daily injections of insulin with dose adjustments based on the patients glucose levels throughout the day. The conventional therapy group did not have to worry about glucose levels and therefor only had to inject two doses of insulin a day.

The studies showed that there was no significant difference between both the intensive treatment group and the conventional treatment group in terms of risk when it came to contracting cardiovascular disease at the baseline of the experiment, but by the end of the tests the conventional treatment group came at a much higher risk of cardiovascular disease with 98 cardiovascular disease events in 52 patients occurring compared to the 46 events in 31 patients for the intensive treatment. Intensive treatment also reduced the risk of cardiovascular disease by 42%. The results concluded that the intensive treatment was beneficial for the group in the reduction of cardiovascular disease in type 1 diabetes.

2.3 Related Work

But how does all this relate to the work that will be done in the project? In this section of the literature review/research conducted it will be discussed how these papers are relevant to the research in regards to the project, their success and effectiveness in answering the questions put forth, the similarities between each paper, the advantages and disadvantages once compared, and a summary of all results.

2.3.1 Relevance

Each research question deals with how these diseases can be dealt with using effective methods through exercise or self therapy that can be done with the aids of research carried out by medical professionals or proven studies. This is relevant to the main application because the purpose of the application is to create a hub where people suffering from these diseases can do their own research and set reminders on useful exercises and therapy methods which they can carry out in a similar fashion to the studies shown above. Instead of ineffectively doing research, the research already conducted can be made available for the application, meaning this information can be accessed in an easier way.

2.3.2 Success/Effectiveness

Every single paper which has been reviewed posed a question which was then researched and studied by the authors and this proved to be the basis for the papers. Each paper follows a formula to try and be as effective as possible in obtaining answers for the research questions. Each researcher followed their own methods for obtaining results, such as conducting surveys, or performing exercises within a specified group of people in an allocated amount of time. Overall each question which was asked at the start of every paper was solved through a series of studies and further research conducted by the authors.

2.3.3 Similarities

Each work does have certain similarities about the way in which the research was conducted, for example they all follow the same formula as previously mentioned. They open up with a research question posed about an illness, a disease or a disorder. They go onto to then use a methodology to try and get the most accurate results possible for their specified research. While each paper is similar in that they carried out their research by using a specified methodology, not all papers used the same methodology when conducting their research. When compared, each paper has their own advantages, and disadvantages. Some papers are much more in depth and cover their specific topic in much more detail than others. Papers which include a lack of depth for example when compared are the research papers on depression which conducted surveys in an effort to answer the questions posed. The problem here is while the surveys can give you the opinions of a large group of people, they often do not give the answer as to why these people feel the way in which they do. Another paper which could be said to lack depth of the others, was the research done into arthritis, as the answers provided were done by tests not carried out by the authors themselves, but instead was researched and cited by the authors.

2.3.4 Summary of Results

Overall each paper did give a result which benefited the research that was conducted and led to valuable information being found when it comes to designing an application based around fighting specific pains which a disease or an illness may cause. This research will lead to the application being more effective and knowing exactly what exercises lead to benefits in their specific disease, this includes small walking exercises in patients with lung cancer can help to fight the deterioration and improve the symptoms of the disease, aerobic and strengthening exercises in helping with muscle function activities in fighting Rheumatoid Arthritis, talking to people about depression can be more helpful than fighting it alone, exercises in Alzheimer's disease can lead to improved daily activity and lessening the chance of injuries related to the disease and intensive diabetes treatments can lead to a lesser chance of cardiovascular diseases. This information can lead to the application being more useful to those in need of a hub where they can find out specific treatments for their disease and use the applications features to help them follow through with these treatments as much as possible.

3
System Analysis

Before creating anything, whether it be an artist painting a picture, or a carpenter designing a table, or a computer scientist developing a web application, the creator must be able to visualise what they expect the finished product to look like. For a web application such as this, this will include the likes of; how will the users be able to interact with each web-page? Will the design of the application make it easy for users to understand? Will the application provide its overall objective, in this case informing, and helping people with a disease or illness? This is why it's so important for a developer to be be able to distinguish and establish the functional, and non-functional requirements of their application

3.1 Functional Requirements

There are many functional requirements to consider when developing a web application. For this particular web application which is designed in Symfony 3 it is important that;

- The web applications routes should all be fully functional and return a HTTP Status code of 200.
- A user should be taken to a specific route depending on the illness they have registered with.
- When a user registers their details such as username, password, email, and illness should be stored in a database.
- Certain routes on the application should be restricted to users who do not have the right credentials.

- A user should be able to create a reminder in a scheduler/calendar for a specific date and time.
- When this is done, the reminders details are stored in a database
- Once a reminder is created, the user can edit or delete the reminder.
- A user should be able to post a message to a forum.
- The messages details should be stored in a database
- The user should be able edit or delete the message posted.
- Any updates or deletions should be recognized by the database

3.2 Non Functional Requirements

There are also a number of non-functional requirements that are necessary when designing a web application. These usually relate to concerns regarding the user when browsing the application, such as;

- All users should be able to navigate the application with relative ease.
- The web application should have an appealing look to anyone intending to use it.
- The web application should be easily understandable in terms of what it provides to the user.

3.3 Use Case Diagrams

This section contains a use case diagram of the system. This is what the user can expect in terms of interaction with the web application.

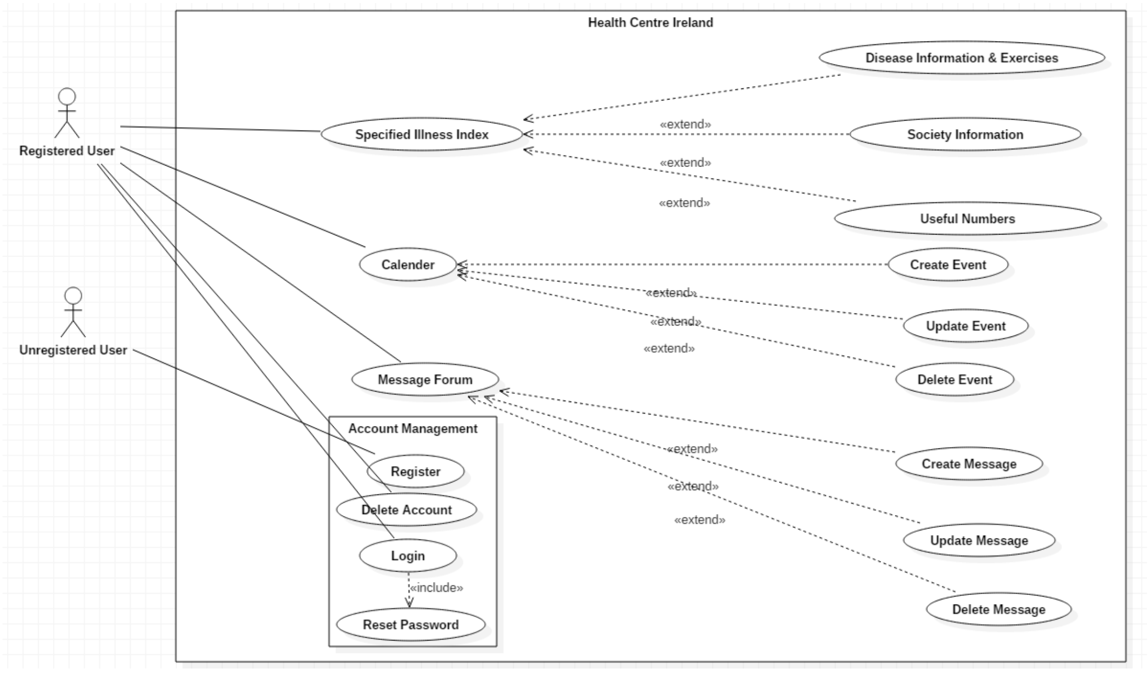


Figure 3.1: System Use Case

3.3.1 Registration

Summary: The user register for the web application with their specified illness, they are then assigned a role, and the database stores their information.

Precondition: The user must not be previously registered to the application.

Triggers: The users is signed up to the application and their data is stored in a database.

Course of Events:

1. The user navigates to the web application.
2. The user is unregistered.
3. The user clicks the registration button and is redirected to a registration form.
4. The user enters their details and registers.
5. The users details are all stored.
6. The user can now successfully sign in and navigate the web application fully.

Postcondition: The user can now use the web application and find out information about their illness, along with posting messages and setting reminders.

3.3.2 Log in

Summary: The users signs in to their account.

Precondition: The user must be already registered for the application.

Triggers: The users details are searched in the database and if they match the user signs in successfully.

Course of Events:

1. The user navigates to the web application.
2. The user clicks the sign in button and is redirected to the sign in page.
3. The user enters their username and password.
4. If the username and password combination are correct the user is successfully signed in.

Postcondition: The user signs in and the features of the web application are available to them.

3.3.3 Delete Account

Summary: The users deletes their account and can no longer access the web application with that specific account.

Precondition: The user must have a registered account, and signed in to the web application.

Triggers: The users account is deleted and their information is removed from the database.

Course of Events:

1. The user is registered for the application
2. The user is signed in to the application
3. The user clicks the option to delete their account.
4. The users account is deleted and their details are removed from the database.

Postcondition: The user can no longer sign in on that account, and must create a new account if they wish to use the application.

3.3.4 Reset Password

Summary: The user forgets their password and cannot sign in, so they must reset their password.

Precondition: The user must be registered for the application, but not signed in.

Triggers: An email is sent to the user containing a link, and from here they can reset their password

Course of Events:

1. The user navigates the application to the forgot password route.
2. The user enters their email address and a link containing the password reset link is sent to that email.
3. The user enters their new password.
4. The users new password replaces the old password in the database.

Postcondition: The users password is changed in the database and the user can sign in with their new password.

3.3.5 Specified Illness Index

Summary: When the user signs in they are redirected to a route about their specific illness in which they signed up with.

Precondition: The user must be registered and signed in.

Triggers: Once the user specifies their illness on registration they are redirected to an index page about this specific illness.

Course of Events:

1. The user registers for the application.
2. The users signs in to the application.
3. The user is redirected to the illness specification index page.

Postcondition: The user can now look at information relating to the illness they have signed up with.

3.3.6 Disease Information & Exercises

Summary: The user can view a series of informational articles and papers about their illness' and information on exercises and treatments they can do to relive the illness.

Precondition: The user must be signed in to the application.

Triggers: The user can now view information regarding their specified illness.

Course of Events:

1. The user signs in to the web application.
2. The user is redirected to the illness specification index page.
3. The user can now access the page which displays information about their illness.

Postcondition: The user has access to a page which displays information about their illness and how they can combat it through exercise. It will also include a link to a paper which gives information about the exercises and treatments the user can do regarding the illness.

3.3.7 Society Information

Summary: The user can view information about societies which help people deal with their illness. This page will have information such as the society's number, a link to thier website, an RSS Feed of their social media pages and a link in which the user can donate to the society if they want to.

Precondition: The user must be signed in to the application.

Triggers: The user can view information about the charity/society regarding their illness.

Course of Events:

1. The user signs in to the web application.
2. The user is redirected to the illness specification index page.
3. The user can now access the page which displays information about the society.

Postcondition: The user now has access to the page which displays information on the specific charity/society about their illness.

3.3.8 Useful Numbers

Summary: The user can view useful numbers of helplines which they can call to talk about their illness.

Precondition: The user must be signed in to the application

Triggers: The user can view useful phone numbers for helplines in which they can talk to professionals about their illness.

Course of Events:

1. The user signs in to the web application.
2. The user is redirected to the illness specification index page.
3. The user can now access the page which displays all the useful helpine numbers.

Postcondition: The user can view information about the helplines regarding their specific illness.

3.3.9 Calendar

Summary: The user can check their reminders in their own calendar.

Precondition: The user must be signed in.

Triggers: The user is taken to the calendar page in the application

Course of Events:

1. The user signs in to the web application.
2. The user then navigates to the calendar page of the web application.

Postcondition: The user views their calendar and can set reminders in the calendar.

3.3.10 Create Event

Summary: The user can set a reminder or an event in their calendar. Whether it be to take part in an exercise, or take a prescribed medication, or if they have a medical appointment.

Precondition: The user must be signed in and on the calendar page.

Triggers: A reminder is then set for the user and it is stored in the database.

Course of Events:

1. The user signs in to the application.
2. The user navigates to the calendar page of the web application.
3. The user sets a reminder in their calendar.
4. The reminder appears in the users calendar and is stored in the database.

Postcondition: A reminder is then set for the user which they can see on their calendars view.

3.3.11 Update Event

Summary: Once a reminder is set in the users calendar the user can then update the reminder as they see fit.

Precondition: The users reminder must already be set in the calendar.

Triggers: The users reminder is updated and the details in the database are then changed to the new reminder.

Course of Events:

1. The user signs in to the application.
2. The user navigates to the calendar page of the application.
3. The user sets a reminder in their calendar.
4. The reminder appears in the users calendar and its information is stored in the database.

5. The user then updates the reminder to be different.
6. The reminders information is then changed on the calendars view and in the database.

Postcondition: The event is then updated in the calendars view and the information which has been stored in the database is updated.

3.3.12 Delete Event

Summary: Once a reminder is set in the users calendar the user can then delete the reminder if they want to.

Precondition: The users reminder must already be set in the calendar.

Triggers: The reminder is deleted and removed from the view and the database.

Course of Events:

1. The user signs in to the application.
2. The user navigates to the calendar page of the application.
3. The user sets a reminder in their calendar.
4. The reminder appears in the users calendar and its information is stored in the database.
5. The user can then delete the reminder.
6. The reminder is then removed from the calendars view and its information is removed from the database.

Postcondition: The reminder is removed and can no longer be seen in the view.

3.3.13 Message Forum

Summary: The user views the messages which have been written and posted by other users.

Precondition: The user must be signed in.

Triggers: The user views a series of messages which they can post and view other users messages.

Course of Events:

1. The user signs in to the web application
2. The user navigates to the message forum section of the web application.

Postcondition: The user can view and post messages and communicate with other people on the web application.

3.3.14 Create Message

Summary: The user posts a message to the forum.

Precondition: The user must be signed in and on the message forum web page of the application.

Triggers: A new message is posted to the message board. The messages information is then stored in a database.

Course of Events:

1. The user signs in to the web application.
2. The user navigates to the message forum page of the application.
3. The user posts a message in the forum.
4. A new message appears in the message forum which other users can then see and its information is stored in the database.

Postcondition: The user posts their own personalised message in the message board which can then be seen by other users. ### Update Message **Summary:** The user updates their previously posted message.

Precondition: The user must be signed in and already have previously posted a message.

Triggers: The message is changed and updated, and the messages information in the database is also updated.

Course of Events:

1. The user signs in to the web application.
2. The user navigates to the message forum of the application.
3. The user posts a message in the forum.
4. The message appears in the forum and its information is stored in the database.
5. The user updates the message and the changed message appears in the forum. The message information in the database is also updated.

Postcondition: The message is changed and updated in the forum.

3.3.15 Delete Message

Summary: The user deletes the message which was previously posted.

Precondition: The user must be signed in and have already previously posted a message.

Triggers: The message is deleted from the message forums view and deleted from the database.

Course of Events:

1. The user signs in to the web application.
2. The user navigates to the message forum of the application.
3. The user posts a message in the forum.
4. The message appears in the forum and its information is stored in the database.
5. The user deletes the message from the forum and its information is removed from the database.

Postcondition: The message is deleted from the forum and can no longer be seen.

3.4 Class Diagrams

4 Implementation and Design of System

4.1 Overview of Implementations

The system was implemented in Symfony 3, a web framework designed to speed up the creation of web applications and replace certain repetitive coding tasks. It eliminates these coding tasks because the developer can reuse components and it can make development less daunting for the developer by not having to redevelop basic, generic features, such as making user forms. Symfony also certain tools which can improve productivity for the user. The Web Toolbar Debugger being the main feature which does this. This tells the developer detailed technical information about each page and request made on the application. This makes it easier to track errors, such as the HTTP request code. If an error is detected Symfony will return a detailed error page outlining what the error is and this will be shown in the toolbar also. The toolbar also contains the users current credentials, the time it took the page to execute and render and the current page which is being displayed.

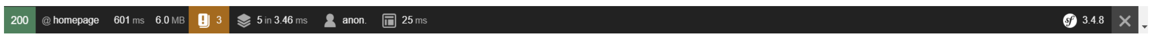


Figure 4.1: Symfony Toolbar

Symfony was installed using “Composer”[23], a tool used for dependency management in PHP. It allows the developer to declare libraries which the developers project will depend on. Symfony was than created by issuing the command

```
composer create-project symfony/website-skeleton:^3.4 HealthCentreIreland
```

The project also used databases made in MySQL Workbench 6.3[24]. These databases were used to store various bits of information, be it the users info, the reminders made in the scheduler, and the messages posted. As Symfony does not provide a component which works directly with databases, the third party library Doctrine was used.

Doctrine was installed using the command

```
composer require-doctrine
```

From here all the necessary components were installed and the project could start being worked on.

4.2 Implementation Components

There are several components which needed to be implemented in order to provide a fully functional web application which provides its purpose. These included;

- Database Entities.
- User Registration
- User Login
- Delete User
- User Calendar
- User Blog/Messaging System
- Implanting Routes based on user credentials.

4.2.1 Database Entities

First, databases needed to be set up which dealt with storing the users information, along with the calendar information and the posted messages information. To first set up the databases the parameters.yml file in the app/config folder with the database settings which are specific to the developers environment. The parameters.yml file in this specific project looked like this.

```
parameters:
    database_host: 127.0.0.1
    database_port: 3306
    database_name: member_form
    database_user: root
    database_password: dbpassword
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: ThisTokenIsNotSoSecretChangeIt
```

Once these settings were in place the database was created using the command

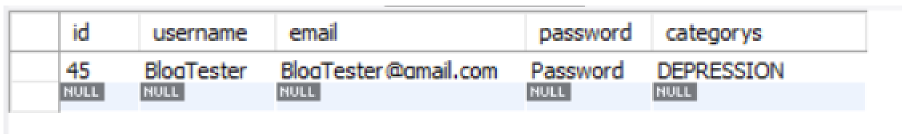
```
php bin/console doctrine:database:create
```

Once this is done the table must be created. This table was created with the title of member and the columns being the details which the user must input at a later stage, these being; id, username, email, password and categorys.

The DDL(Data Definition Language) looked like this for the setup of this table:

```
CREATE TABLE `member` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `email` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `password` varchar(64) COLLATE utf8_unicode_ci NOT NULL,
  `categorys` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=45 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

Once the table was created it looked like this:



id	username	email	password	categorys
45	BlogTester	BlogTester@gmail.com	Password	DEPRESSION
NULL	NULL	NULL	NULL	NULL

Figure 4.2: Database Member Table

Each time a database is created an ORM file for the particular database will be setup.

The command to generate an ORM file is

```
php bin/console doctrine:mapping:import --force AppBundle yml
```

This will generate an ORM file for the Member table which will look like this:

```
AppBundle\Entity\Member:
    type: entity
    table: member
    indexes:
        memberType_idx:
            columns:
                - categorys
    id:
        id:
            type: integer
            nullable: false
            options:
                unsigned: false
            id: true
            generator:
                strategy: IDENTITY
    fields:
        username:
            type: string
            nullable: false
            length: 255
            options:
                fixed: false
        email:
            type: string
            nullable: false
            length: 255
            options:
                fixed: false
        password:
            type: string
            nullable: false
            length: 64
            options:
                fixed: false
        categorys:
            type: string
            nullable: false
            length: 255
            options:
                fixed: false
    lifecycleCallbacks: {  }
```

Now that the database has been connected and setup and an ORM file has been generated, a Doctrine entity can be created. This is an object with an $id property in the database.

The member entity is then generated using the command:

```
php bin/console doctrine:generate:entities AppBundle
```

Once generated the entity class looks like this:

Member.php

```
<?php
```

```
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Member
 *
 * @ORM\Table(name="member")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\MemberRepository")
 */
class Member implements UserInterface, \Serializable
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     *
     * @var string
     * @ORM\Column(name="username", type="string", length=255, unique=true)
     *
     */
    protected $username;

    /**
     *
     * @var string
     * @ORM\Column(name="email", type="string", length=255, unique=true)
     *
     */
    protected $email;

    /**
     * @var string
     * @ORM\Column(name="categorys", type="string", length=255)
     */
    protected $categorys;


    protected $plainPassword;


    /**
     * @var string
     *
     * @ORM\Column(name="password", type="string", length=64)
     */
    protected $password;


    /**
     * Get id
     *
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }


    /**
     * Set username
     *
     * @param string $username
     *
     * @return Member
     */
    public function setUsername($username)
    {
        $this->username = $username;

        return $this;
    }

    /**
```

```
     * Get username
     *
     * @return string
     */
    public function getUsername()
    {
        return $this->username;
    }

    /**
     * Set email
     *
     * @param string $email
     *
     * @return Member
     */
    public function setEmail($email)
    {
        $this->email = $email;

        return $this;
    }

    /**
     * Get email
     *
     * @return string
     */
    public function getEmail()
    {
        return $this->email;
    }

    /**
     * Set password
     *
     * @param string $password
     *
     * @return Member
     */
    public function setPassword($password)
    {
        $this->password = $password;

        return $this;
    }

    /**
     * Get password
     *
     * @return string
     */
    public function getPassword()
    {
        return $this->password;
    }
```

To transform this into a symfony user object on the application two interfaces must first be implemented. These two interfaces being:

```
implements UserInterface, \Serializable
```

and:

```
use Symfony\Component\Security\Core\User\UserInterface;
```

By implementing these two interfaces more methods must be added to the Member entity. These methods being the "getRoles()", getSalt(), and eraseCredentials() method. The getRoles() method will return an arrays of strings, these strings being the users role within the application. For example in this web application the roles given to users are based on their registered illness, such as; "ROLE_CANCER", "ROLE_DEPRESSION", "ROLE_DIABETES", "ROLE_ALZHEIMERS" and "ROLE_ARTHRITIS".

To assign the user the correct role, the function .strtoupper() is used to read in the category which the users signs up with and converts the string to uppercase. This is preceded by the string 'ROLE_' meaning that the users category in the database is added to the end of this, returning the role of 'ROLE_ARTHRITIS' for example.

```
    public function getRoles()
    {
            return[
                'ROLE_' . strtoupper($this->categorys)
            ];
    }
```

The getSalt() method will hash and encrypt passwords. But if a password is being hashed using bcrypt as it will be in this project there is no need to use the salt method, and it can return null. The eraseCredentials() method will remove any sensitive data from the Member object. The purpose of implementing the "Serializable" method is so that the user only has to log in once for a pre-determined length of time, as opposed to having to log in every time a new request is made on the application. Creating a user and how their information is added to the database will be further explained in section 4.2.2.

Once the member table is created it is necessary to also create a table for both both the scheduler and the schedulers category. For the scheduler a table with all the necessary information regarding the users set reminder must be taken into account. The table was called appointments and was given 5 fields all relating to the reminders set by the user, these being; id, title, description, start_date, end_date and category.

The DDL for the following table was;

```
CREATE TABLE `appointments` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `description` text,
  `start_date` datetime NOT NULL,
  `end_date` datetime NOT NULL,
  `category` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_6A41727A64C19C1` (`category`),
  CONSTRAINT `FK_6A41727A64C19C1` FOREIGN KEY (`category`) REFERENCES `categories` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=38 DEFAULT CHARSET=utf8
```

Once this table has been created it looks like:

	id	title	description	start_date	end_date	category
	NULL	NULL	NULL	NULL	NULL	NULL

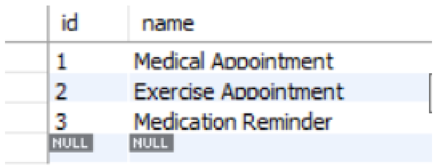
Figure 4.3: Database Appointment Table

The category field is a foreign key which references another table called category. This will further be explained in section 4.2.4. It is used to display a category in which the user can set their reminder for, being "Medical Reminder", "Exercise Reminder" and "Medication Reminder"".

The DDL for this looks like;

```
CREATE TABLE `categories` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8
```

Once the table is created it looks like:



id	name
1	Medical Appointment
2	Exercise Appointment
3	Medication Reminder
NULL	NULL

Figure 4.4: Database Category Table

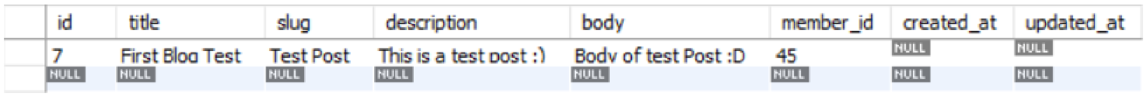
Once both of these tables have been created both ORM files and entities were generated for each of them using the same command as before. Both the ORM files and the entities can be viewed in the appendix of this paper.

The final table which needed to be created was one which dealt with posting messages within a blog. For this table it was necessary to store all the information regarding the posted message. There are 8 field names in total. These being; id, title, slug, description, body, member_id, created_at and updated_at. The member_id field is a foreign key which references the member table and the column of id. The reason why will be further explained in section 4.2.5.

The DDL for the following table was:

```
CREATE TABLE `blog_post` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `slug` varchar(255) NOT NULL,
  `description` varchar(2000) NOT NULL,
  `body` text,
  `member_id` int(11) DEFAULT NULL,
  `created_at` datetime DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `slug_UNIQUE` (`slug`),
  KEY `blogForeignKey_idx` (`member_id`),
  CONSTRAINT `blogForeignKey` FOREIGN KEY (`member_id`) REFERENCES `member` (`id`) ON DELETE CAS
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8
```

Once the table is created it will look like.



id	title	slug	description	body	member_id	created_at	updated_at
7	First Blog Test	Test Post	This is a test post :)	Body of test Post :D	45	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.5: Database Blog Post Table

An ORM file and an entity is also generated for this table which can be viewed in the appendix of this paper.

4.2.2 User Registration

Once all the necessary databases are set up and the entities are generated it is time to move on to creating a registration controller for the user which will store users information into the database. The first thing to do is to change the entity class slightly to allow for a bcrypt encrypted password in the password field so the users plain password is not stored here. Instead of this, all future user requests to log in to the application will involve running the bcrypt process and comparing that value to the users submitted plain password. If the two match, this means that the right password has been submitted. This means that a temporary holding area to store the users plain password must be created. A new class property is added to the Memeber.php entity class to hold this plain password

```
private $plainPassword;

/**
* @return String
*/
public function getPlainPassword()
{
    return $thid->plainPassowrd;
}

/**
*@param string $plainPassowrd
*@return Member
*/
public function setPlainPassword($plainPassword)
{
    $this->plainPassword = $plainPassword;

    return $this;
}
```

After this is done an encoder must be configured in security.yml for users passwords, for which the bcrypt algorithm will be used. The member entity is then added to encoders using the bcrypt algorithm and now all passwords in this entity will be encrypted.

```
    encoders:
        Symfony\Component\Security\Core\User\User:
            algorithm: bcrypt
        AppBundle\Entity\Member:
            algorithm: bcrypt
```

```
providers:
    in_memory:
        memory:
            users:
                admin:
                    password: $2y$13$sBG1nMvGxgY.AprC/tYl5ecuDvxT66bYNRrdX0zAyQCqQiUX./fqS
                    roles: 'ROLE_ADMIN'
```

Once this is done, the next step is to generate a registration form. This will be talked about further in section 4.3 of this paper. Once the registration form is generated, a controller must be made which handles all routes in dealing with registration and form submission. First a function which renders the registration form page must be created. What this function does it will redirect any user who has the credentials of "ROLE_ARTHRITIS", "ROLE_CANCER", "ROLE_DIABETES", "ROLE_DEPRESSION" OR "ROLE_ALZHEIMER" back to their specified index route and any anonymous user will be redirected to to the twig template page "register.html.twig". This page creates the view for the form which is passed through the key 'registration_form'.

```
/**
  * @Route("/register", name="registration")
  * @return \Symfony\Component\HttpFoundation\Response
  * @throws \LogicException
  */
 public function registerAction(Request $request)
 {
     $member = new Member();

     $form = $this->createMemberRegistrationForm($member);


     if($this->container->get('security.authorization_checker')->isGranted('ROLE_ARTHRITIS'))
     {
         return $this->redirectToRoute('arthritis');
     }

     else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_CANCER
     {
         return $this->redirectToRoute('cancer');
     }

     else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DIABET
     {
         return $this->redirectToRoute('diabetes');
```

```
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DEPRESSION'))
        {
            return $this->redirectToRoute('depression');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_ALZHEIMER'))
        {
            return $this->redirectToRoute('alzheimers');
        }

        return $this->render('registration/register.html.twig',[
            'registration_form' => $form->createView(),

        ]);
    }
```

The next function used in this class is the handleFormSubmission(). This function deals with the submission of the form. If the form is not successfully submitted the user is redirected back to the registration page. Once the request is handled the password is then encoded, and the value of the $member entity is set. Then the Doctrine entity manager is used to save this to the database. An instance of UsernamePasswordToken is then used, this takes 4 arguments 3 of which are mandatory. These arguments are an object which represents the user, the users credentials, a provider key which is called after the firewall, in this case the firewall is called main, so the value used for the provider key is 'main'. Then the role of the user is returned. The final lines of code then check the users role and will redirect the user accordingly. The user is now successfully registered

```
    /**
     * @param Request $request
     * @Route("/registration-form-submission", name="handle_registration_form_submission")
     * @Method("POST")
     * @return \Symfony\Component\HttpFoundation\RedirectResponse|\Symfony\Component\HttpFoundation\Res
     * @throws \LogicException
     * @throws \InvalidArgumentException
     */
    public function handleFormSubmissionAction(Request $request)
    {
        $member = new Member();

        $form = $this->createMemberRegistrationForm($member);
```

```
$form->handleRequest($request);

if (! $form->isSubmitted() || ! $form->isValid()) {
    return $this->render('registration/register.html.twig',[
        'registration_form' => $form->createView(),

    ]);
}

$password = $this->get('security.password_encoder')
    ->encodePassword($member, $member->getPlainPassword());

$member->setPassword($password);

$em = $this->getDoctrine()->getManager();


$em->persist($member);
$em->flush();

$token = new UsernamePasswordToken(
    $member,
    $password,
    'main',
    $member->getRoles()
);

$this->get('security.token_storage')->setToken($token);
$this->get('session')->set('_security_main', serialize($token));

$this->addFlash('success', 'You are now registered');

if($this->container->get('security.authorization_checker')->isGranted('ROLE_ARTHRITIS'))
{
    return $this->redirectToRoute('arthritis');
}

else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_CANCER
{
    return $this->redirectToRoute('cancer');
}
```

```
        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DIABETES'))
        {
            return $this->redirectToRoute('diabetes');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DEPRESSION'))
        {
            return $this->redirectToRoute('depression');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_ALZHEIMER'))
        {
            return $this->redirectToRoute('alzheimers');
        }
    }
```

4.2.3 User Login

Once a user is registered and wants to login the user needs to be loaded from the database. This is done in the security.yml file. First what must be done in the providers section of the security.yml file a db_provider must be added which then loads in the users from an entity. A chain provider must then be built which will look through both the db_provider and the in_memory provider until it finds the matching user. The property which is being looked for in the provider in this case is the username, in the class AppBundle:Member. Once this is found and the password is correct the user will be logged in.

```
    providers:
      chain_provider:
        chain:
          providers:
            - in_memory
            - db_provider
      in_memory:
            memory:
                users:
                    admin:
                        password: $2y$13$sBG1nMvGxgY.AprC/tYl5ecuDvxT66bYNRrdX0zAyQCqQiUX./fqS
                        roles: 'ROLE_ADMIN'
      db_provider:
        entity:
```

```
        class: AppBundle:Member
        property: username
```

When a user can be loaded from the database a login form must be created so that the user can log in with their correct credentials. This will be talked about further in section 3.4

The next function which must be coded is the route which renders the login page. If a user is granted any role they will not be able to access the login page as they are already logged in. So they will be redirected to their specified illness page, but otherwise, the route /login will render the twig template of 'security/login.html.twig'.

```
    /**
     * @Route("/login", name="login")
     */
    public function loginAction()
    {
        if($this->container->get('security.authorization_checker')->isGranted('ROLE_ARTHRITIS'))
        {
            return $this->redirectToRoute('arthritis');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_CANCER
        {
            return $this->redirectToRoute('cancer');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DIABET
        {
            return $this->redirectToRoute('diabetes');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DEPRES
        {
            return $this->redirectToRoute('depression');
        }

        else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_ALZHEI
        {
            return $this->redirectToRoute('alzheimers');
        }

            return $this->render('security/login.html.twig',[
            ]);
```

```
}
```

The next challenge is where will the user be redirected once they login? The user will be redirected the route /signin once they are logged in. This route will render a page based on the users role within the application.

```
/**
 * @Route("/signin", name="signin")
 */
public function signinAction()
{
    if($this->container->get('security.authorization_checker')->isGranted('ROLE_ARTHRITIS'))
    {
        return $this->redirectToRoute('arthritis');
    }

    else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_CANCER'))
    {
        return $this->redirectToRoute('cancer');
    }

    else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DIABETES'))
    {
        return $this->redirectToRoute('diabetes');
    }

    else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_DEPRESSION'))
    {
        return $this->redirectToRoute('depression');
    }

    else if ($this->container->get('security.authorization_checker')->isGranted('ROLE_ALZHEIMER'))
    {
        return $this->redirectToRoute('alzheimers');
    }
}
```

4.2.4 Log Out

If a user wants to log out of the web application first a route for logout must be configured.

```
/**
```

```
     * @Route("/logout", name="logout")
     * @throws \RuntimeException
     */
    public function logoutAction()
    {
        throw new \RuntimeException('Logout');
    }
```

Then inside the firewalls main section of security.yml a logout parameter must be added and set to true.

```
    firewalls:
        main:
            pattern: ^/
            provider: chain_provider
            form_login:
                login_path: login
                failure_path: login
                check_path: login
                success_handler: crv.authentication.success_handlers
                success_handler: crv.authentication.failure_handlers
            logout: true
```

4.2.5 Delete User

Deleting a user is similar to registering a user. The function gets the current user that is signed in and instead of the user being added to the database using '$em->persist$(member);' the user is removed instead using 'em->remove($user);'. The users role is then defined, the users token is set to null and the users details are removed from the database. The user is then redirected to the login page.

```
    /**
     * @Route("/deleteuser", name="deleteuser")
     */
    public function deleteUser(Request $request)
    {

        $user = $this->getUser();
        $member = new Member();

        $password = $this->get('security.password_encoder')
            ->encodePassword($member, $member->getPlainPassword());
```

```
if($this->container->get('security.authorization_checker')->isGranted('ROLE_ARTHRITIS'))
{

    $token = new UsernamePasswordToken(
        $member,
        $password,
        'main',
        $member->getRoles()
    );

    $this->get('security.token_storage')->setToken(null);

    $em = $this->getDoctrine()->getManager();
    $em->remove($user);
    $em->flush();

    return $this->redirectToRoute('login');
}
```

4.2.6 User Calendar/Scheduler

There were a number of libraries which were needed to create the scheduler for this application. The first of these libraries being the dhtmlx[25] scheduler library. This library came as a zipped file which was downloaded and then imported into the project. The moment.js[26] library was used to format the date. The final library which was used was the jQuery AJAX library to submit reminders into the view. All the databases, entities and ORM files were already previously implemented for the scheduler. So the next was to implement the schedulers controllers and its routes. There are only 4 routes for the scheduler, all of these routes can be accessed at the route /scheduler. The routing.yml file was modified to register a routing file which will then handle the routes for the calendar.

```
app_scheduler:
    resource: "@AppBundle/Resources/config/routing/scheduler.yml"
    prefix:   /scheduler
```

The new routing file is located in AppBundle/Resources/config/routing/scheduler.yml, with each route will be handled by the SchedulerController.php class. The following routes in this yml file will create, delete and modify the reminders via jQuery AJAX.

```
# app/config/routing.yml
```

```
scheduler_create:
    path:      /appointment-create
    defaults:  { _controller: AppBundle:Scheduler:create }
    methods:  [POST]

scheduler_update:
    path:      /appointment-update
    defaults:  { _controller: AppBundle:Scheduler:update }
    methods:  [POST]

scheduler_delete:
    path:      /appointment-delete
    defaults:  { _controller: AppBundle:Scheduler:delete }
    methods:  [DELETE]
```

Next a controller must be created to handle the routes for the scheduler. The first function renders the scheduler, along with retrieving the entity manager and getting the repository for both the appointments and the categories tables in the database. A JSON structure must also be generated from the appointments to render at the start of the scheduler. A JSON structure must also be generated the categories repository so they categories that the user can select will be rendered inside a select option on the lightbox. The scheduler is then rendered.

```
    /**
     * @Route("/scheduler", name="scheduler")
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();

        $repositoryAppointments = $em->getRepository("AppBundle:Appointments");

        $repositoryCategories = $em->getRepository("AppBundle:Categories");

        $appointments = $repositoryAppointments->findAll();

        $formatedAppointments = $this->formatAppointmentsToJson($appointments);

        $categories = $repositoryCategories->findAll();

        $formatedCategories = $this->formatCategoriesToJson($categories);

        return $this->render("scheduler/scheduler.html.twig", [
            'appointments' => $formatedAppointments,
            'categories' => $formatedCategories,
        ]);
    }
```

The next function after this handles the creation of the appointment. It uses the date format use by the Moment.js in the view. The appointment entity is then created and the fields values are set. The appointment is then created and saved to the database.

```
   /**
     * Creates reminder in the database.
     */
    public function createAction(Request $request){
        $em = $this->getDoctrine()->getManager();
        $repositoryAppointments = $em->getRepository("AppBundle:Appointments");

        $format = "d-m-Y H:i:s";

        $appointment = new Appointment();
        $appointment->setTitle($request->request->get("title"));
        $appointment->setDescription($request->request->get("description"));
        $appointment->setStartDate(
            \DateTime::createFromFormat($format, $request->request->get("start_date"))
        );
        $appointment->setEndDate(
            \DateTime::createFromFormat($format, $request->request->get("end_date"))
        );

        $repositoryCategories = $em->getRepository("AppBundle:Categories");

        $appointment->setCategory(
            $repositoryCategories->find(
                $request->request->get("category")
            )
        );

        // Create appointment
        $em->persist($appointment);
        $em->flush();

        return new JsonResponse(array(
            "status" => "success"
        ));
    }
```

The next function is used to handle any updates made to the appointment. If the appointment being updated does not exist an error is returned. The rest of the function then carries out like the createAction() function does.

```
   /**
     * Updates reminder in the database.
     */
    public function updateAction(Request $request){
        $em = $this->getDoctrine()->getManager();
        $repositoryAppointments = $em->getRepository("AppBundle:Appointments");

        $appointmentId = $request->request->get("id");

        $appointment = $repositoryAppointments->find($appointmentId);

        if(!$appointment){
            return new JsonResponse(array(
                "status" => "error",
                "message" => "The appointment you tried to update does not exist?"
            ));
        }

        $format = "d-m-Y H:i:s";

        $appointment->setTitle($request->request->get("title"));
        $appointment->setDescription($request->request->get("description"));
        $appointment->setStartDate(
            \DateTime::createFromFormat($format, $request->request->get("start_date"))
        );
        $appointment->setEndDate(
            \DateTime::createFromFormat($format, $request->request->get("end_date"))
        );

        $repositoryCategories = $em->getRepository("AppBundle:Categories");

        $appointment->setCategory(
            $repositoryCategories->find(
                $request->request->get("category")
            )
        );

        // Update appointment
        $em->persist($appointment);
        $em->flush();

        return new JsonResponse(array(
            "status" => "success"
        ));
    }
```

A function must also be created to delete appointments from the database. Like before if the appointment does not exist an error message is returned. The appointment is then removed from the database.

```
    /**
     * Deletes reminder from the database
     */
    public function deleteAction(Request $request){
        $em = $this->getDoctrine()->getManager();
        $repositoryAppointments = $em->getRepository("AppBundle:Appointments");

        $appointmentId = $request->request->get("id");

        $appointment = $repositoryAppointments->find($appointmentId);

        if(!$appointment){
            return new JsonResponse(array(
                "status" => "error",
                "message" => "The appointment you tried to delete does not exist?"
            ));
        }

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

        return new JsonResponse(array(
            "status" => "success"
        ));
    }
```

The final function which is then created returns a JSON string of the reminder which is then rendered on the calendar. The dates must follow the format already set by the Moment.js library.

```
    private function formatAppointmentsToJson($appointments){
        $formatedAppointments = array();

        foreach($appointments as $appointment){
            array_push($formatedAppointments, array(
                "id" => $appointment->getId(),
                "description" => $appointment->getDescription(),
                "text" => $appointment->getTitle(),
                "start_date" => $appointment->getStartDate()->format("Y-m-d H:i"),
                "end_date" => $appointment->getEndDate()->format("Y-m-d H:i"),
            ));
        }

        return json_encode($formatedAppointments);
    }
```

The same must be done then for the category which will return a json string based on the data in the repository.

```
    private function formatCategoriesToJson($categories){
        $formatedCategories = array();

        foreach ($categories as $category){
            array_push($formatedCategories, array(
                "key" => $category->getId(),
                "label" => $category->getName()
            ));
        }

        return json_encode($formatedCategories);
    }
```

Client side logic must the be written. The user will be allowe to create their appointments through the default Lightbox of the dhtmlx scheduler. What must be written is the way in which the scheduler will behave. The xml_date format must be provided, while the rest is optional. Once the date is set the scheduler is set so that it starts at midnight, and ends at 24:00 hours. Meaning the time in which the user can set reminders is between 00:00 and 23:55. The details of the event are also shown on creation

```
scheduler.config.xml_date="%Y-%m-%d %H:%i";
scheduler.config.first_hour = 0;
scheduler.config.last_hour = 24;
scheduler.config.limit_time_select = true;
scheduler.config.details_on_create = true;
```

In the next section event creation is disabled on a single click and if a user wants to create and event they must double click. The user can also only set a maximum of 5 events each month before they must delete one.

```
scheduler.config.select = false;
scheduler.config.details_on_dblclick = true;
scheduler.config.max_month_events = 5;
scheduler.config.resize_month_events = true;
```

The Lightbox form sections are created next. With the names of “Title”, “Description”, “Type of Reminder” and “Time”. They are also given heights of 30, 50, 40 and 30 respectively. The Type Of Reminder options are called from the Category table.

```
scheduler.config.lightbox.sections = [
    {name:"Title", height:30, map_to:"text", type:"textarea"},
    {name:"Description", height:50, map_to:"description", type:"textarea"},
    {name: "Type of Reminder", options: window.GLOBAL_CATEGORIES, height:40, map_to: "category", type:
    {name:"Time", height:30, type:"time", map_to:"auto"}
];
```

Next custom settings are setup for the calendar, such as the element where the scheduler will be started. The date where the scheduler should be started and which mode it should start on. In this case the scheduler will start on the current day, and will start on the mode week, meaning it will show the current week by default.

```
var initSettings = {
    elementId: "scheduler_element",
    startDate: new Date(),
    mode: "week"
};
```

Next a funtion must be created which formats the events to the expected format in the server side.

```
/*
 * @param {*} id
 * @param {*} useJavascriptDate
 */
function getExpectedFormat(id, useJavascriptDate){
    var event;

    // If the ID already exists on an event don't search for it and use it
    if(typeof(id) == "object"){
        event = id;
    }else{
        event = scheduler.getEvent(parseInt(id));
    }

    if(!event){
        console.error("This ID does not exist " + id);
        return false;
    }

    var start , end;

    if(useJavascriptDate){
        start = event.start_date;
        end = event.end_date;
    }else{
        start = moment(event.start_date).format('DD-MM-YYYY HH:mm:ss');
        end = moment(event.end_date).format('DD-MM-YYYY HH:mm:ss');
    }

    return {
        id: event.id,
        start_date : start,
        end_date : end,
        description : event.description,
        title : event.text,
        category: event.category
    };
}
```

Next the handlers for the events must be attached. These being the create, update and delete handlers. If the appointment is successfully created an alert will tell the user their appointment has been created, otherwise they will get an error message. It is important to update the id of the schedulers appointment with the ID of the database, so the appointment can be edited.

```
/**
 * Calendar Create Event
 */
scheduler.attachEvent("onEventAdded", function(id,ev){
    var schedulerState = scheduler.getState();

    $.ajax({
        url:  window.GLOBAL_SCHEDULER_ROUTES.create,
        data: getExpectedFormat(ev),
        dataType: "json",
        type: "POST",
        success: function(response){
            scheduler.changeEventId(ev.id , response.id);

            alert('Appointment '+ev.text+ " has been created! Good Luck!");
        },
        error:function(error){
            alert('Error: The following appointment '+ev.text+ 'refresh and try again or report to an a
            console.log(error);
        }
    });
});
```

If a user successfully updates an event they will get alert saying that the appointment has been successfully updated, otherwise they will get an error message.

```
/**
 * Calendar Update Event
 */
scheduler.attachEvent("onEventChanged", function(id,ev){
    $.ajax({
        url:  window.GLOBAL_SCHEDULER_ROUTES.update,
        data: getExpectedFormat(ev),
        dataType: "json",
        type: "POST",
        success: function(response){
            if(response.status == "success"){
                alert("Appointment Updated!");
            }
        },
        error: function(error){
            alert("Error: Cannot save changes, refresh and try again or report to Admin");
            console.error(error);
        }
    });

    return true;
});
```

If a user successfully deletes an event they will get alert saying that the appointment has been successfully deleted, otherwise they will get an error message.

```
/**
 * Calendar Delete Event
 */
scheduler.attachEvent("onConfirmedBeforeEventDelete",function(id,ev){
    $.ajax({
        url: window.GLOBAL_SCHEDULER_ROUTES.delete,
        data:{
            id: id
        },
        dataType: "json",
        type: "DELETE",
        success: function(response){
            if(response.status == "success"){
                if(!ev.willDeleted){
                    alert("Appointment succesfully deleted");
                }
            }else if(response.status == "error"){
                alert("Error: Cannot delete appointment refresh and try again or report to an Ad
            }
        },
        error:function(error){
            alert("Error: Cannot delete appointment: " + ev.text + "refresh and try again or rep
            console.log(error);
        }
    });
    return true;
});
```

4.2.7 User Blog/Messaging System

Next, the blog system should be implemented so users can communicate with other users on the web application. As shown earlier in the paper the table in the database has been created to store information about the blog posts, along with the entities for the blog posts have been generated along with the ORM files. But the field member_id has been created as a foreign key, referencing the member table, and the column of id. This has been done so that when a specific user creates a blog the id of that user is stored so the system knows the user which that blog post belongs too. First the blog controller must be created, rendering the twig template of blog/BlogPosts.html.twig.

```
    /**
     * @Route("/blogs", name="blogs")
     */
```

```
public function blogAction()
{
    return $this->render('blog/BlogPosts.html.twig',[
    ]);
}
```

Next the entities must be injected into this class to retrieve the data from the database. At the top of the class the following services were injected.

```
/**
 * @var EntityManagerInterface
 */
private $entityManager;

/**
 * @var \Doctrine\Common\Persistence\ObjectRepository
 */
private $memberRepository;

/**
 * @var \Doctrine\Common\Persistence\ObjectRepository
 */
private $blogPostRepository;


public function __construct(EntityManagerInterface $entityManager)
{
    $this->entityManager = $entityManager;
    $this->blogPostRepository = $entityManager->getRepository('AppBundle:BlogPost');
    $this->memberRepository = $entityManager->getRepository('AppBundle:Member');
}
```

A new controller then has to be implemented in the BlogController which deals with creating blog entries. In this it renders an entry form page which is generated through a form. Generation of forms will be explained further in the Graphical Components and User Interface section. What the createBlogEntryAction does is check if the form submitted for the blog entry is valid, if the blog entry is valid then the form is successfully submitted and the blog will be posted.

```
/**
 * @Route("/blogs/create-entry", name="create_entry")
 * @param Request $request
 * @return \Symfony\Component\HttpFoundation\Response
 */
```

```
public function createBlogEntryAction(Request $request)
{
    $blogPost = new BlogPost();

    $member = $this->memberRepository->findOneByUsername($this->getUser()->getUserName());
    $blogPost->setMember($member);

    $form = $this->createForm(EntryFormType::class, $blogPost);
    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){
        $this->entityManager->persist($blogPost);
        $this->entityManager->flush($blogPost);

        $this->addFlash('success', 'Congratulations! You have created a new post!');

        return $this->redirectToRoute('blogs');
    }

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

Next a delete functionality must be made for the users post. In this function the blog is found by its entry id, and so is the user. It checks the table in the database to see if the blog post belongs to that particular member, and if it does that user can delete the blog post, otherwise they cannot and they will be redirected to the route 'create_entry'. This is to prevent people from being able to delete any blog post which they please. Once the blog post is deleted it is then removed from the database and the user is redirected to the 'create_entry' page.

```
/**
 * @Route("/delete-entry/{entryId}", name="delete_entry")
 *
 * @param $entryId
 *
 * @return \Symfony\Component\HttpFoundation\RedirectResponse
 */
public function deleteBlogEntryAction($entryId)
{
    $blogPost = $this->blogPostRepository->findOneById($entryId);
    $member = $this->memberRepository->findOneByUsername($this->getUser()->getUserName());
```

```
        if (!$blogPost || $member !== $blogPost->getMember()){
            $this->addFlash('error', 'Unable to Remove Blog Post');

            return $this->redirectToRoute('create_entry');
        }

        $this->entityManager->remove($blogPost);
        $this->entityManager->flush();

        $this->addFlash('success', 'Entry was Deleted');

        return $this->redirectToRoute('create_entry');
    }
```

The final function for the BlogController is an action which displays the details of a singular blog post. This generates a url based on the slug which the user enters in their blog entry form. If the blog post does not exist then the user is given an error message and redirected back to the blog page. Otherwise they will be redirected to the blog/entry.html.twig template which is rendered in this action.

```
/**
 * @Route("/blogs/userentry/{slug}", name="userentry")
 */
public function slugAction($slug)
{
    $blogPost = $this->blogPostRepository->findOneBySlug($slug);

    if(!$blogPost){
        $this->addFlash('error', 'Unable to find entry');

        return $this->redirectToRoute('blogs');
    }
    return $this->render('blog/entry.html.twig', array(
        'blogPost' => $blogPost
    ));
}
```

4.2.8 Implementing Routes based on user credentials

Depending on the credentials or the roles the user has they will be only be able to access certain routes. This is part of what makes the website personal to specific peoples illness'. User signed up with Alzheimer's will only have the routes with information regarding alzheimer's, while people with cancer will only have routes with information specific to cancer. They will not be able to access certain parts of the website. Certain routes though are open to all users such as, the blog and the calendar. While other routes are only available to users who are not authenticated, such as login and registration. These routes are assigned to users in the access_control parameter of the secuirty.yml file.

Here is a list of routes and the users which can access them:

```
access_control:
  - {path: ^/$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/(login|register)$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/registration-form-submission$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/recover$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/reset/confirm$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/reset/request$, role: IS_AUTHENTICATED_ANONYMOUSLY}
  - {path: ^/arthritis$, role: ROLE_ARTHRITIS}
  - {path: ^/arthritis/arthritisinformation$, role: ROLE_ARTHRITIS}
  - {path: ^/arthritis/arthritisireland$, role: ROLE_ARTHRITIS}
  - {path: ^/arthritis/arthritisusefulinformation$, role: ROLE_ARTHRITIS}
  - {path: ^/cancer$, role: ROLE_CANCER}
  - {path: ^/cancer/cancerinformation$, role: ROLE_CANCER}
  - {path: ^/cancer/cancersocietyinfo$, role: ROLE_CANCER}
  - {path: ^/cancer/cancerusefulinformation$, role: ROLE_CANCER}
  - {path: ^/diabetes$, role: ROLE_DIABETES}
  - {path: ^/diabetes/diabetesinformation$, role: ROLE_DIABETES}
  - {path: ^/diabetes/diabetesireland$, role: ROLE_DIABETES}
  - {path: ^/diabetes/diabetesusefulinformation$, role: ROLE_DIABETES}
  - {path: ^/depression$, role: ROLE_DEPRESSION}
  - {path: ^/depression/depressioninformation$, role: ROLE_DEPRESSION}
  - {path: ^/depression/mentalhealthireland$, role: ROLE_DEPRESSION}
  - {path: ^/depression/depressionusefulinformation$, role: ROLE_DEPRESSION}
  - {path: ^/alzheimers$, role: ROLE_ALZHEIMER}
  - {path: ^/alzheimers/alzheimersinformation$, role: ROLE_ALZHEIMER}
  - {path: ^/alzheimers/alzheimerssociety$, role: ROLE_ALZHEIMER}
  - {path: ^/alzheimers/alzheimersusefulinformation$, role: ROLE_ALZHEIMER}
  - {path: ^/signin$, role: [ROLE_ARTHRITIS, ROLE_CANCER, ROLE_DEPRESSION, ROLE_DIABETES, RO
  - {path: ^/scheduler$, role: [ROLE_ARTHRITIS, ROLE_CANCER, ROLE_DEPRESSION, ROLE_DIABETES,
  - {path: ^/blogs$, role: [ROLE_ARTHRITIS, ROLE_CANCER, ROLE_DEPRESSION, ROLE_DIABETES, ROL
  - {path: ^/blogs/create-entry$, role: [ROLE_ARTHRITIS, ROLE_CANCER, ROLE_DEPRESSION, ROLE_
```

4.3 Graphical Components and User Interface

The graphical components and user interface of a web application are extremely important as it is what attracts a user to the application. If the application is to complicated to use, or does not look appealing to the user they are more likely to not return to the application. The css and general integration of the web application can enhance the user experience when it comes to dealing with a web application.

4.3.1 Bootstrap

A large part of the css was created using the bootstrap library[27]. This is an open source front end library which can be used in designing web sites and web applications. Unlike most libraries and frameworks bootstrap only concerns itself with front end development. The implementation of bootstrap used for this application was 4.0.0.

4.3.2 Base

A base file is included within the html.twig templates that comes with the Symfony Framework. This file is usually extended among all other pages, meaning that each page in the application will contain the elements from this template. For this application the template file was a toolbar which would link to every other part of the web application. It would also have a log out button for the user and a delete account option. The base file was styled using bootstrap and personalised css. It also included scripts for jQuery.

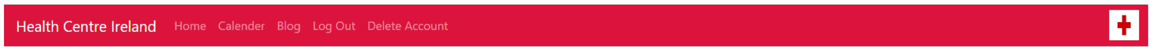


Figure 4.6: Toolbar

```
<body>
<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
    <a class="navbar-brand" href="#">Health Centre Ireland</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
            data-target="#navbarsExampleDDefault" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarsExampleDefault">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('homepage') }}">Home <span class="sr-only"></span> </
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path ('scheduler') }}">Calender</a>
            </li>

            <li class="nav-item">
                <a class="nav-link" href="{{ path('blogs') }}">Blog</a>
            </li>

            <li class="nav-item">
                {% if app.user %}
                    <a class="nav-link" href="
                    {{ logout_path('main') }}">
                        Log Out</a>
                {% else %}
                    <a class="nav-link" href="
                    {{ path ('login') }}">
                        Log In</a>
                {% endif %}

            <li class="nav-item">
                {% if app.user %}
                    <a class="nav-link" href="{{ path('deleteuser') }}"> Delete Account</a>
                {% endif %}
            </li>
        </ul>
        <img class="redcross"
             src="{{ asset('images/RedCross.png') }}" alt="redcross" height="40" width="40">
    </div>

    {% if app.session.flashBag.has('success') %}
        <div class ="alert alert-success">
            {% for msg in app.session.flashBag.get('success') %}
                {{ msg }}
            {% endfor %}
        </div>
    {% endif %}
</nav>
```

4.3.3 Forms

As previously stated forms were generated for both registration and blog submissions. The form for registration was generated using the command

```
php bin/console doctrine:generate:form AppBundle:Member
```

This created the form MemberType.php. The password property in this form was not worked with directly, first the plainPassword property was used to store the plain password and then save the encoded password to the password property. Also included in the form was the username, email and the categories ChoiceType. The choices were “Alzheimer’s”, “Arthritis”, “Lung Cancer”, “Depression” and “Type 1 Diabetes”. Depending on what the user chose the respective choice was stored in the database. The form was also styled using the ‘form-control’ class.

```
/**
 * {@inheritdoc}
 */
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder->add('username', TextType::class, [
        'attr' => ['class' => 'form-control'],
        'label' => 'Username: '
    ])
        ->add('email', EmailType::class, [
            'attr' => ['class' => 'form-control'],
            'label' => 'Email: '
        ])
        ->add('categorys', ChoiceType::class,
                    array('label' => 'Illness type: ',
                    'choices' => array(
                        'Alzheimer\'s' => 'ALZHEIMER',
                        'Arthritis' => 'ARTHRITIS',
                        'Lung Cancer' => 'CANCER',
                        'Depression' => 'DEPRESSION',
                        'Type 1 Diabetes' => 'DIABETES',
                                ),
                        'attr' => ['class' => 'form-control'],
                    ))
        ->add('plainPassword', RepeatedType::class,[
            'attr' => ['class' => 'form-control'],
        'type'=>PasswordType::class,
        'first_options' => [
            'attr' => ['class' => 'form-control'],
            'label' => 'Password: ',
        ],
        'second_options'=>[
            'attr' => ['class' => 'form-control'],
            'label' => 'Repeat Password: '
        ]
    ])
    ->add('register', SubmitType::class,[
        'attr' => ['class' => 'form-control btn-primary pull-right'],
        'label' => 'Register!!'
    ]);
}
```

The form view was then passed in a twig template under the key of registration_form. From this the registration form page is produced.

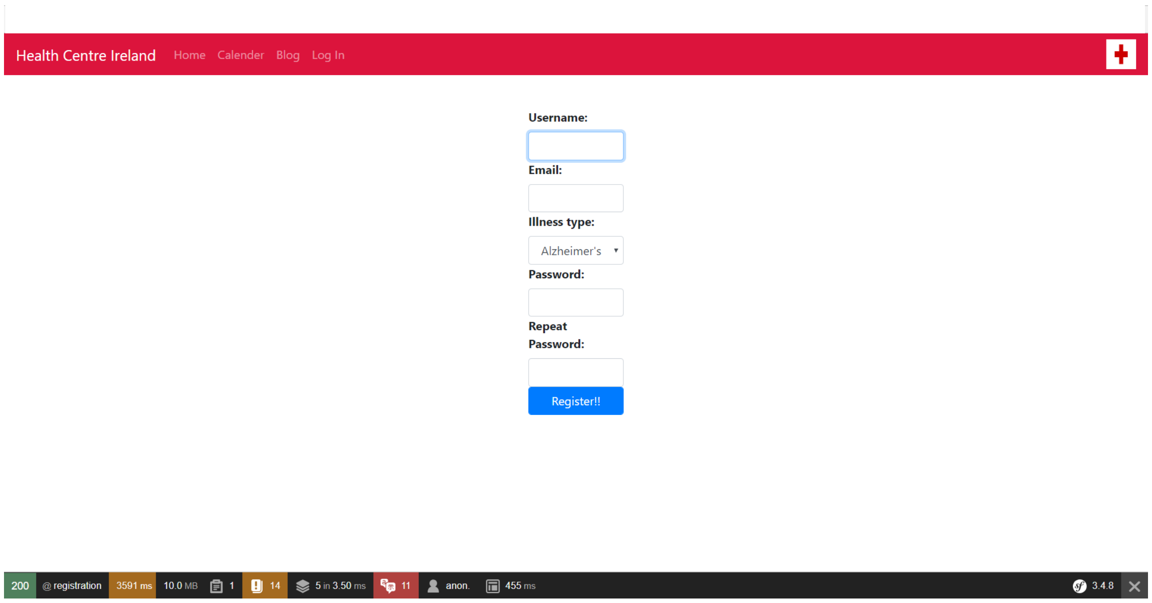


Figure 4.7: Registration Page

```
{% block body %}
    <div class="reg">
        {{ form(registration_form) }}
    </div>
{% endblock %}
```

For the Blog Post form called EntryFormType.php was created. The data class used is the entity of BlogPost. These include the likes of the title of the blog, the slug, which has been given the label of identifier so users know what this field does, the description of the blog, the blog's body and finally a button to create the blog. The form was styled using the class 'form-control'.

```
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title',
                TextType::class, [
                    'constraints' => [new notBlank()],
                    'attr' => ['class' => 'form-control']
                ])
            ->add('slug', TextType::class,[
                'label' => 'Identifier',
                'constraints' => [new NotBlank()],
                'attr' => ['class' => 'form-control']
            ])
            ->add('description' , TextareaType::class,[
                'constraints' => [new NotBlank()],
                'attr' => ['class' => 'form-control']
            ])
            ->add('body', TextareaType::class,[
                'constraints'=>[new NotBlank()],
                'attr' => ['class' => 'form-control']
            ])
            ->add('create', SubmitType::class,[
                'attr' => ['class' => 'form-control btn-primary pull-right'],
                'label' => 'Create!'
            ]);
    }
```

A template called entry_form.html.twig was then created and the following code was inserted into it to display the blog entry form. From this template the blog entry page is then produced.

```
{% block body %}

    <div class="container">
        <div class="blog-header">
            <h2 class="blog-title"></h2>
        </div>

        <div class="row">
```

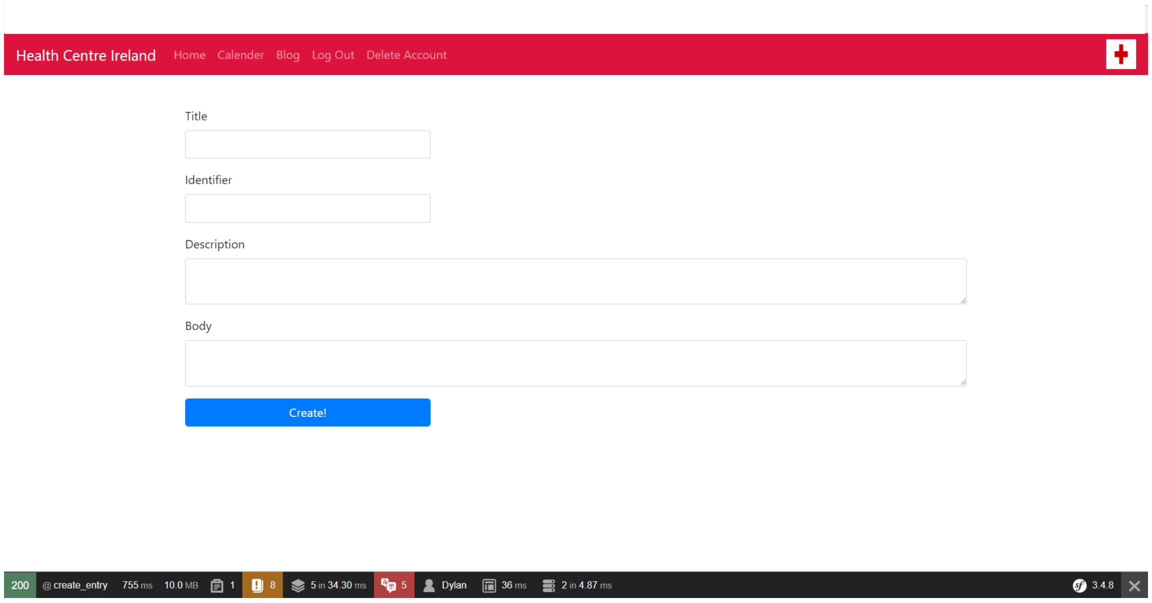


Figure 4.8: Blog Entry Page

```
<div class="col-sm-12 blog-main">
    {% for label, messages in app.flashes %}
        {% for message in messages %}
            <div class="bg-{{ label }}">
                {{ message }}
            </div>
        {% endfor %}
    {% endfor %}

    {{ form_start(form) }}
    <div class="col-md-12">
        <div class="form-group col-md-4">
            {{ form_row(form.title) }}
        </div>
        <div class="form-group col-md-4">
            {{ form_row(form.slug) }}
        </div>
            <div class="form-group col-md-12">
                {{ form_row(form.description) }}
            </div>
            <div class="form-group col-md-12">
                {{ form_row(form.body) }}
            </div>
```

```
                        <div class="form-group col-md-4 pull-right">
                            {{ form_widget(form.create) }}
                        </div>
                    </div>
                    {{ form_end(form) }}
                </div>
            </div>
        </div>
{% endblock %}
```

5
Testing & Evaluation

6
Evaluation & Self Reflection

7 Conclusions & Further Work

7.1 Conclusions

7.2 Further Work

Part I

Appendices

List of References

[1] Potencier Fabien, Symfony 3, June 2017, https://symfony.com

[2] Lvque Bruno, Schlumberger Igor, PrestaShop, 2007, https://www.prestashop.com/en

[3] Sonota Project, https://sonata-project.org/

[4] Pimcore, Pimcore, 2013, https://pimcore.com/en

[5] MediSafe.inc, Medisafe Pill Reminder 5.4.1, September 25 2017, https://itunes.apple.com/us/app/medisafe-pill-reminder/id573916946?mt=8

[6] GreatCall.inc, MedCoach Medication Reminder 1.12, November 11 2016, https://itunes.apple.com/us/app/medcoach-medication-reminder/id443065594?mt=8

[7] Schwartz L Anna, Cancer Fitness: Exercise Programs for Patients and Survivors. Simon and Schuster, 1439103933, 9781439103937, 2008

[8] Dr. Brewer Sarah, Overcoming Arthritis: The Complete Complementary Health Program. Watkins Media, 780282605, 9781780282602, 2012

[9] Dr. Smith Matt, An Introduction to Symfony 3. 2017

[10] Hoffman Markus, COMP H4028, Research Skills, Proposal Lecture

[11] Phillay Srini MD, The Dangers of Self Diagnosis, May 03 2010, {https://www.psychologytoday.com/blog/debunking-myths-the-mind/201005/the-dangers-self-diagnosis

[12] Irish Cancer Society, Fact About Lung Cancer in Ireland, https://www.cancer.ie/reduce-your-risk/health-education/cancer-awareness-campaigns/lung-cancer-awareness/lung-cancer-ireland

[13] Temel S. Jennifer MD, Greer A. Joseph PhD, Goldberg Sarah MD, Vogel Downed Paul PT, MS, Sullivan Michael PT MBA, Pirl F. William MD, Lynch J Thomas MD, Christiani C David MD, Smith R Matthew MD Phd, A Structured Exercise Program for Patients with Advanced Non-small Cell Lung Cancer, May 05 2009

[14] Arthritis Ireland, Frequently Asked Questions, http://www.arthritisireland.ie

[15] Stenström H Christiana, Minor A Marian, Evidence for the benefit of aerobic and strengthening exercise in rheumatoid arthritis, June 03 2003

[16] Minor MA, Hewett JE, Webel RR, Anderson SK, Day DR, Efficacy of physical conditioning exercise in patients with rheumatoid arthritis and osteoarthritis, Arthritis Rheum, 1989

[17] Häkkinen A, Häkkinen K, Hannonen P, Effects of strength training on neuromuscular function and disease activity in patients with recent onset inflammatory arthritis, Scan J Rheumatol, 1994

[18] Jorm F Anthony, Kelly M Claire, Wright Annemarie, Parslow A Ruth, Harris G Meredith, McGorry D Patrick, Belief in dealing with depression alone: Results from community surveys of adolescents and adults, Journal of Affective Disorders, November 2006

[19] Irish Health.com, Alzheimer's Disease, http://www.irishhealth.com

[20] Dvorak RV, Poehlman ET. Appendicular skeletal muscle mass, physical activity, and cognitive status in patients with Alzheimer's disease, Neurology.org, 1998

[21] Santana-Sosa E, Barriopedro M I, Lopez-Mojares M, Perez M, Lucia A, Exercise Training is Beneficial for Alzheimer's Patients, February 08 2008

[22] Nathan M David MD, Cleary A Patricia MS, Backlund C Jye-Yu MS, Genuth M Saul MD, Lachin M John d.Sc, Orchard J Trevor MS, Raskin Phillip MD, Zinam Bernard MD, Intensive Diabetes Treatment and Cardiovascular Disease in Patients with Type 1 Diabetes, Feb 10 2009