GOLF' N HOME SWAP

By Will Hogan & Andrej Lavrinovics

Golf' n Home Swap

Will Hogan

Andrej Lavrinovics

B.Sc.(Hons) in Software Development

Submission Date 17th April 2017

Final Year Project

GitHub link:

https://github.com/willhogan11/Golf_n_HomeSwap/

Advised by: Dr. John Healy & Maria Murphy,
Department of Computer Science and Applied Physics,
Galway-Mayo Institute of Technology (GMIT).

TABLE OF CONTENTS

TABLE OF FIGURES

Abstract - We live in an era where sharing has evolved from something between a small group of people to a global phenomenon thanks to the new Sharing Economy that's sweeping across the world. It's no secret that throughout history, certain economies thrive or are born out of times of financial crisis, especially when there is something of value to offer. The idea behind this Software Development project, Golf'n Home Swap is to capitalise on previous Home Swap models ie Airbnb, Love Home Swap, seen globally. With this idea, a user who is a member of a golf club can swap homes with someone in a certain locality or somewhere abroad and both will have visitor access to the other person's Golf Club and vice versa, for a specific period. Initially, the application will be limited to a certain number of countries, with a view to expansion in the future. The report will detail this project's software development journey and focus on the various parts such as the technology used, the methodology or approach that was taken, the way in which the project was managed and finally an evaluation of the working components of the applications, which will demonstrate a multitude of different learning outcomes.

Authors

Will Hogan & Andrej Lavrinovic

4th year Honours Degree Software Development students in GMIT

CHAPTER 1: INTRODUCTION

The Idea

At the beginning of year 4 in Software Development, this software development team were given the opportunity to work on the Golf'n Home Swap project for external client Maria Murphy from the Marketing & Tourism department in GMIT. Upon listening to the initial idea, there was a level of excitement and it was thought that this would be a great way to learn and incorporate several new technologies into a full-scale Web Application. The idea is focused in the direction of the golfing fraternity, that simply allows a user to swap their home with another user. Each user will have a golf membership with their club and the idea is that the members will not only swap houses but can play at the other person's golf club as a guest for the duration of the swap. Initially, the application will be limited to users from Within Ireland and the UK, with a few to global expansion as the application gains traction.

When the initial meetings commenced with Maria, a project handbook was handed to the development team in the form of a Storyboard. This pdf file contained a visual representation of what Maria had envisioned to be the user's interface. This clearly illustrated how a user would interact with the system and from a development perspective, made a lot more sense as it created a strong Idea about how to structure the application. Here's a screenshot of the main home page of the Storyboard;

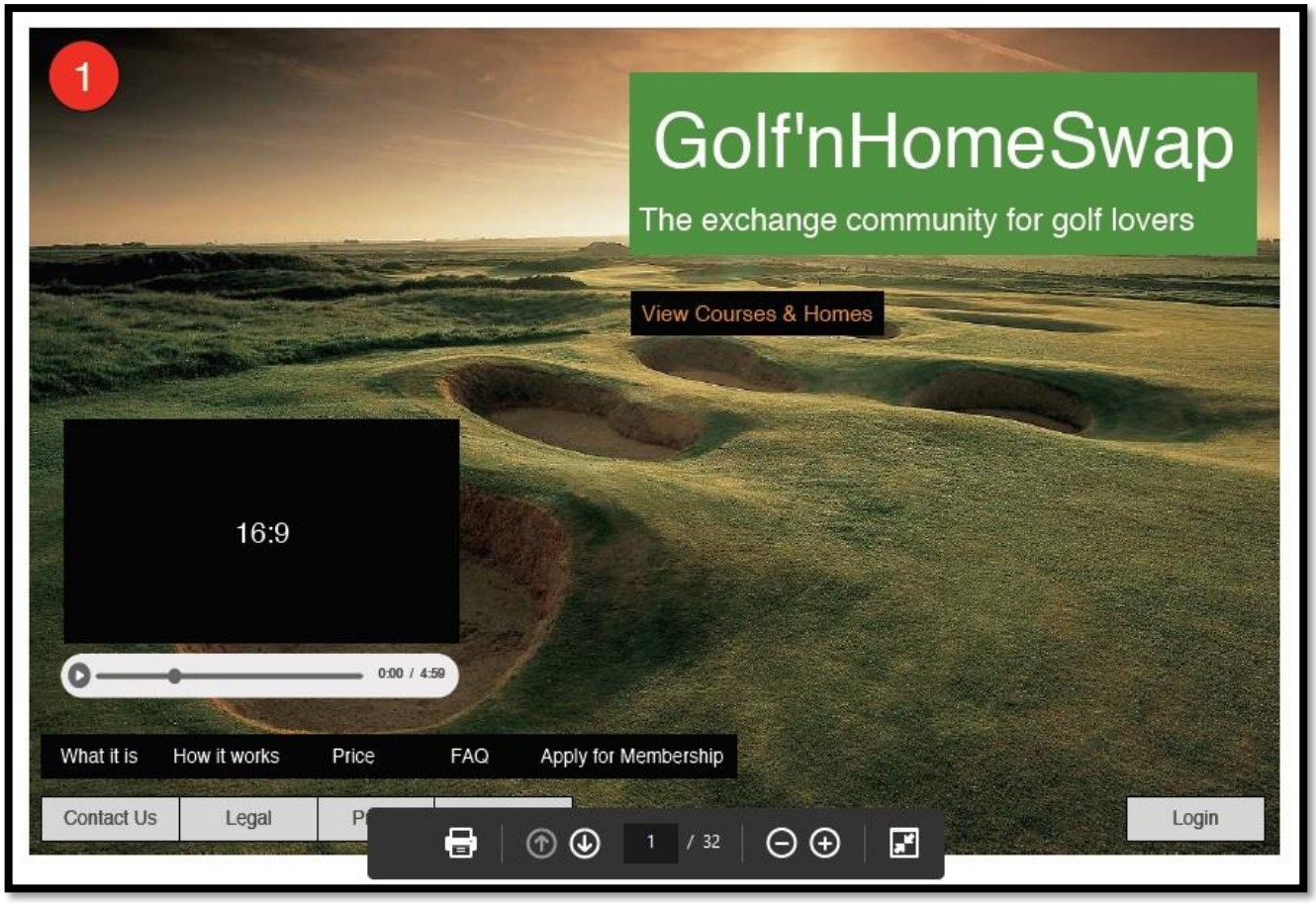


Figure 1: Storyboard Screenshot

The Application

Golf'n Home Swap is a full scale, multi-faceted, Web Application that allows users to view the Website, apply for membership and once approved by the Administrator, become a fully registered member of the System, which will enable users to view, add homes and golf club information to their account. The information that the user provides can be searched by other users in the system and in the future, a registered user will be able to message other users who wish to swap homes and golf club membership for a duration. Because of the general privacy issues associated with user's information being available on the internet, the system has been created from the ground up with a keen focus on security. A user's privileges may be rescinded or reinstated at any time, should the Administrator deem it prudent.

Project Scope

As this is a 4th and final year software development project, it was felt that this project would demonstrate many learning outcomes associated with the required scope at this level. To put this into perspective, when the project specifications were received and because of the varying amount of moving parts in the application, it was decided to choose several different software products to both aid the development process and to compliment the many modular parts and components that make up the System. These ranged from software tools such as Slack that allowed efficient group project collaboration and communication, to using Frameworks within the core development environment that allowed for solid project scaffolding and made the development process much easier. This will be discussed in more detail in the Technology Review Chapter.

Chapter Summary

Throughout this report, a number of different topics are going to be discussed, ranging from how the project was started and how requirements were gathered, to the actual final product that is being delivered. For clarity, here's a synopsis of what each chapter is about.

Methodology

The next chapter is about the methodologies used in the project. Essentially, this is the method that was used to manage and develop this software application. The project incorporates different methodology structures, which follow patterns displayed in Agile, however, it was felt that it would be extremely useful to try and incorporate as many aspects of different methodologies as possible, in order to fully understand how they operate at every level. This allowed the team to learn, but it also offered the opportunity to experiment and find customised and more efficient ways of working together. This was achieved by arranging regular development meetings, where the team would plan, analyse and effectively isolate specific sections of work to be targeted in what are called weekly sprints. Once this piece of work was completed, it was possible to reiterate the process and add or improve on the existing prototype section. Figure 2 shows an Agile Methodology illustration which will be detailed further in the Methodology chapter.

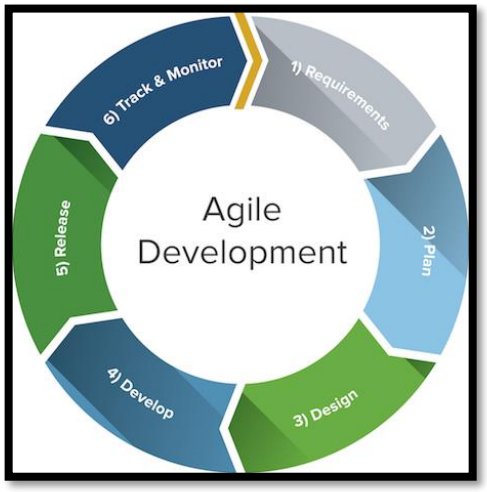


Figure 2: Agile Structure

Technology Review

In this chapter, a complete and comprehensive guide into which Technologies were used during the development of this project is offered. In summary, there were several new technologies that were used that hadn't been used before, prior to commencing software development. For the project management side of things, GitHub was used to manage Milestones and Issues. For group communication, Slack was used to collaborate, and the team also set up an integration on this project's GitHub repository that allowed notifications to be sent when commits or changes were pushed to the repo. Figure 3 shows an illustration that explains how the development flow was communicated to the group;

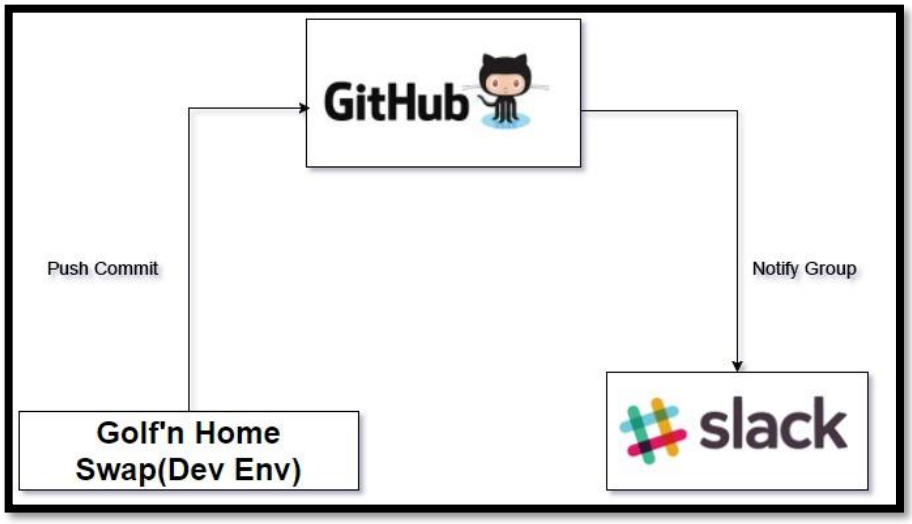


Figure 3: Group notifications

For the development of the project, the Eclipse IDE was used. This allowed the team to incorporate various plugins and libraries into the development environment, associated with a project of this scale. MongoDB was used for the database layer, which is a compact, yet robust document database and uses key-value pairs as its way of storing information.

Implementation

This section deals with how the system was designed, all the way from the top down. Initially, the focus is on a less refined level of granularity, which shows a complete, abstract view of the System in its entirety. This is the overall system architecture and shows how all the various layers and components, connect and communicate with each other. See Figure 4 for a visual representation of how this was accomplished.

The System Design focuses on a more refined level of granularity and looks at various parts of the system, such as the Object-oriented programming approach that was used and Design Patterns sprinkled throughout this applications Java code, ie MVC (Model View Controller), Factory Patterns and Template Patterns. As the system requires user authentication, it became apparent that some major security measures would need to be incorporated into the application. Spring Security was used, which is an inbuilt feature of the Java Spring Framework.

System Evaluation

This chapter details how the application performs in terms of both Robustness and efficiency. An in-depth evaluation of how the system works and responds to various inputs, loads, multiple concurrent users and security features, will be detailed in this chapter. This, in turn, will open a discussion about any limitations that the system might have. Code snippets from this project will be selected and measured in terms of Space and Time complexity. Figure 4 illustrates an example of this;

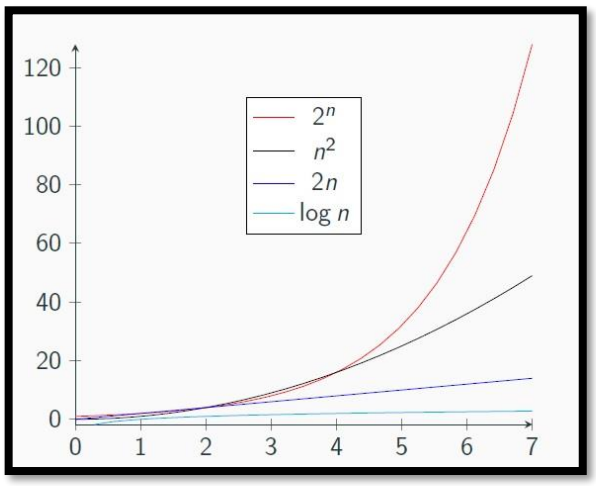


Figure 4: Space and Time Complexity example

Conclusion

Finally, a summary of this project will be discussed along with some of the learning outcomes that have been achieved by creating this system. A reflection will also be offered, on what could have been done differently, which will demonstrate an awareness of certain pitfalls that may have been encountered on during the journey.

CHAPTER 2: METHODOLOGY

Planning

Before any project commences, its quintessential that the proper planning and organisation is done to ensure as smooth a development process as possible. As a team, there was a keen push to start coding and sink teeth into the core of the system as soon as possible, however, this would have been folly and soon realised the pitfalls associated with starting on the wrong foot.

From the get-go, regular meetings were organised with both the project client and project supervisor. From a group development perspective, fortnightly meetings were organised. The first port of call before starting was to write down the various high-level components that are involved in a software development environment.

Text2mindmap **[2]** which is an online mind map creator, was used and allowed a visualisation of how the various modules interacted with each other. Figure 5 provides a screenshot of the Mind Map;

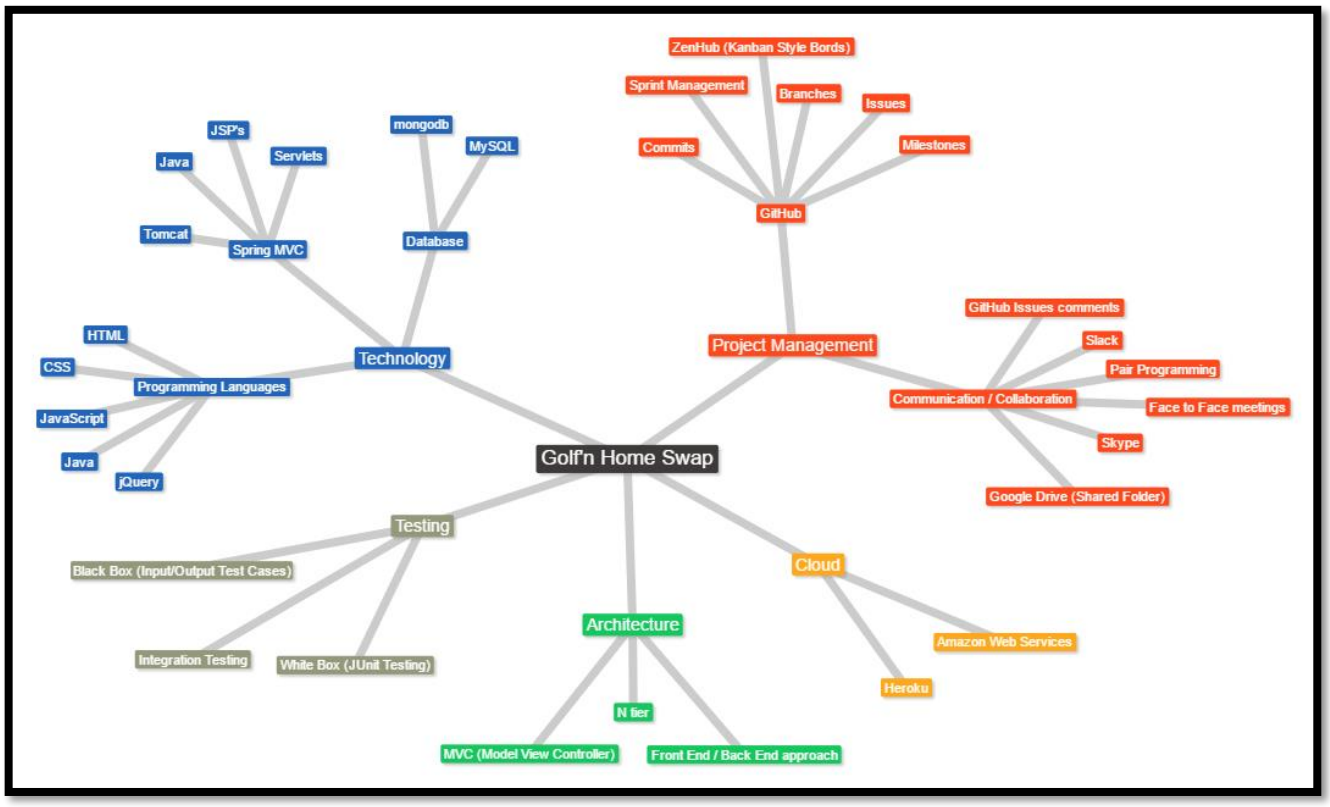


Figure 5: Golf'n Home Swap Mind Map

In the above diagram, there are several nested levels, similar to a tree structure, with each branch representing a section that will be worked on as part of the development of the software, or as part of the general organisation side of things. These are the key parts of the Software Development process, that were focused on.

- **Project Management** – Pivotal in any project and allowed the team to manage time and efforts in the most efficient manner.
- **Architecture** – Which will be discussed in chapter 4, relates to the overall abstract view of the system.
- **Technology** – Looked at the different technologies that were used as part of the development process, for example, programming languages and IDE's (Integrated Development Environments).

- **Testing** – Looked at various testing techniques associated with the Application.
- **Cloud** – Looked at how the system was going to be deployed.

Methodology

As highlighted in the Introduction section, the team chose to implement a cocktail of various methodologies into the development of the project, that allowed for experimenting, which helped find the best possible way of achieving specific tasks. As students, the feeling was that the best way to do this was by using flavours of the Agile methodology and its associated subsets such as RAD (Rapid Application Development), Kanban and XP Programming.

This route was chosen as fundamentally, Agile offers a prototype based, iterative approach which not only allowed for rapid development but introduced a loose structure that suited the team's development skillsets. These are the key components of Agile that were used;

Requirements

This involved the gathering and solidifying of the client's requirements. Questions were asked and feedback was documented, which gave a clear picture of what had to be done. Figure 6 shows a typical development meeting, where requirements were looked at and work was delegated within the group;

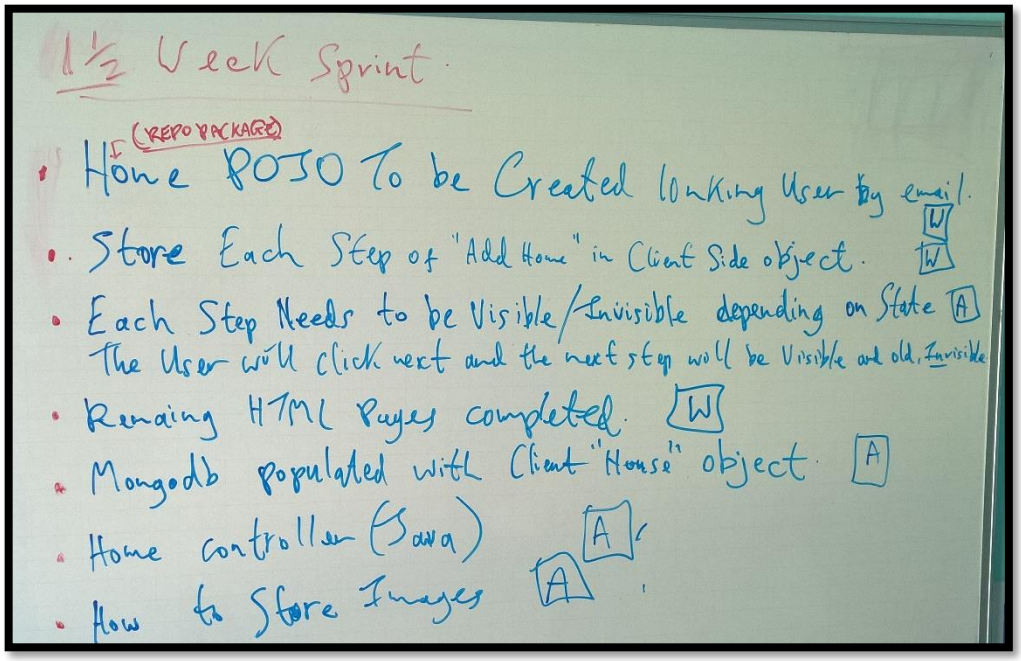


Figure 6: Requirements / Task Delegation

Analysis

This included a comprehensive breakdown of the various pieces of the client's requirements and how they can be implemented into this system. For example, during regular meetings, the team created diagrams to illustrate what should happen. Figure 7 shows how a security feature was implemented, based on requirements;

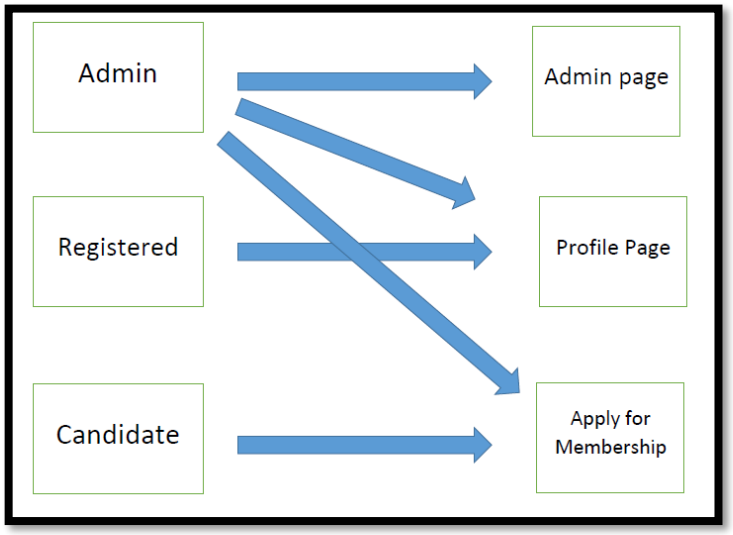


Figure 7: Analysis Diagram Example

Once the team was able to visualise how this security feature would work, it became much easier to implement.

Design

In this section, design specific components based on requirements were analysed, what tech was going to be used, visual sketches etc. Basic HTML pages and the bootstrap CSS framework **[3]** were used, to create basic mock pages that would eventually evolve into the visual representation of the application.

Develop

At this point, the team was now able to not only create some form of prototype or visual representation of how a user will interact with the System but also program the underlying functionality of the application.

To properly organise this, tasks were compartmentalised and work was delegated into Sprints. These sprints typically lasted for a week and at the end of the sprint, what had been completed was reviewed and analysed and what was missed, was added into the next sprint.

The below Figure is a screenshot that illustrates how GitHub was used to manage sprints:

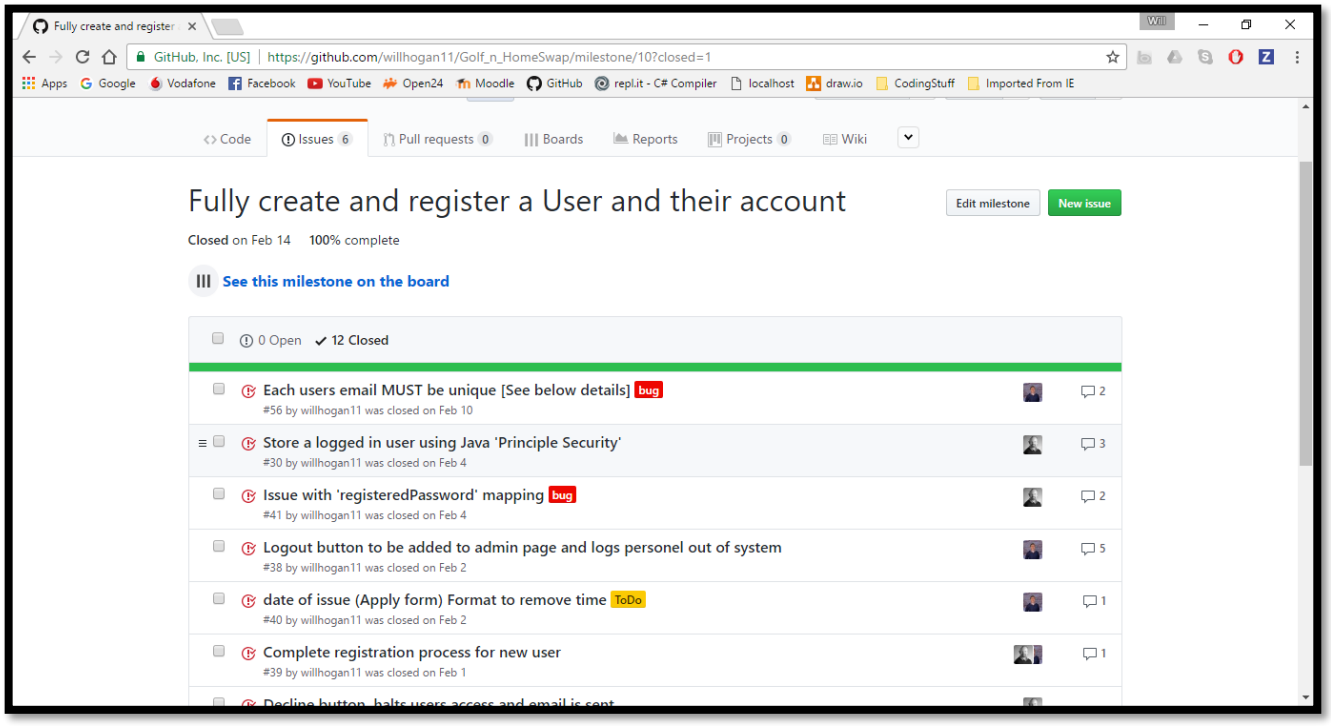


Figure 8: GitHub sprint management example

Other flavours of Agile

One of the ways that a person can learn more about methodologies is by experimenting with them. Although there wasn't any one subset religiously used during this project, there was a focus on incorporating various flavours from each, which resulted in a versatile, customised version that suited the project and was tailored to its development needs. For example, pair programming was used which is part of agile subset XP Programming this allowed for shared learning which sped up the development process. Kanban was also used for issue tracking on GitHub and with this could visualise specific tasks, issues, and milestones, which could be categorised and moved into various sections, such as in progress, complete, testing and deployed. As briefly mentioned in the Design section of this chapter, prototypes were created for the front end of the application, this borrows heavily from the RAD or Rapid Application Development subset of agile, that allowed for quick and relatively easy design prototype creation.

Test / Testing

As this 4th-year project will eventually become a fully functional web application, there remained a keen focus on quality, whilst still maintaining a firm grip on weekly deliverables. To do this, testing was done in two ways.

Black box testing – This is a type of testing that allowed for scrutinisation of the system without testing the code. Essentially what this means is that prior knowledge of the inner workings of the system is not a prerequisite.

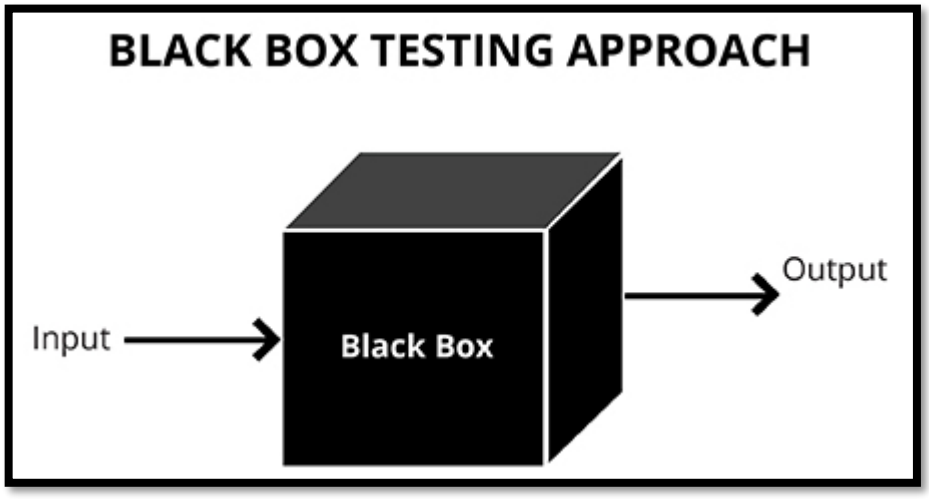


Figure 9: Black box testing

With this example, a user inputs effect the desired output. To explain this in basic terms, let's say there is a system, that when the user inputs 1+1, then the desired output should be 2. An example of this from the perspective of this application might be when a user applies for Golf'n Home Swap membership. The following would be inputs and outputs;

- **Inputs** – All required text fields associated with the apply for membership form are filled in and submitted as a membership application.
- **Output** – Confirmation that the user's application has been successfully submitted and that the application is visible within the Administration page of the application.

Integration Testing

This is a type of self-explanatory testing where certain sections or components that were created, are analysed to ensure that there is a seamless transition into the system. To do this, each section was tested on an individual basis first, then added as a newly added part of the system.

Regression Testing

As the application grew and several new components were incorporated into the system, it became more apparent that checks would need to be put in place, to ensure that existing parts were not affected by the new addition. To do this, Regression testing was performed. An example of this, was when a new section was integrated into the application that allowed users to log in. There are the two sections where this is relevant;

1. Where a newly approved user could change their temporary password for a permanent one.

2. Where an existing user logs on as normal.

It was important that creating the second part didn't affect the first.

Deployment

This refers to how the project was deployed. The application will be hosted using Amazon web services and explained in greater detail in the System Design chapter.

Project Management

Project management is a vital part of any development process and it provides a structure, that allows for efficient time planning, robust work delivery and a general smooth journey through the projects development lifespan. In relation to this project, several methods and software products were used to manage daily and weekly tasks. One of the most important parts of any project is communication. Throughout the development of this project, weekly meetings were organised with both the client and project supervisor to discuss what had been done and what were the next steps or tasks to be completed in the coming week. From the perspective of team communication, several ways were used to communicate.

Slack

Slack is a group collaboration/communication tool that is very useful. It allows teams to discuss projects, insert screenshots and write code snippets and style text using Markdown, which is essentially shorthand CSS. The below Figure is a screenshot of the Golf'n Home Swap slack channel;

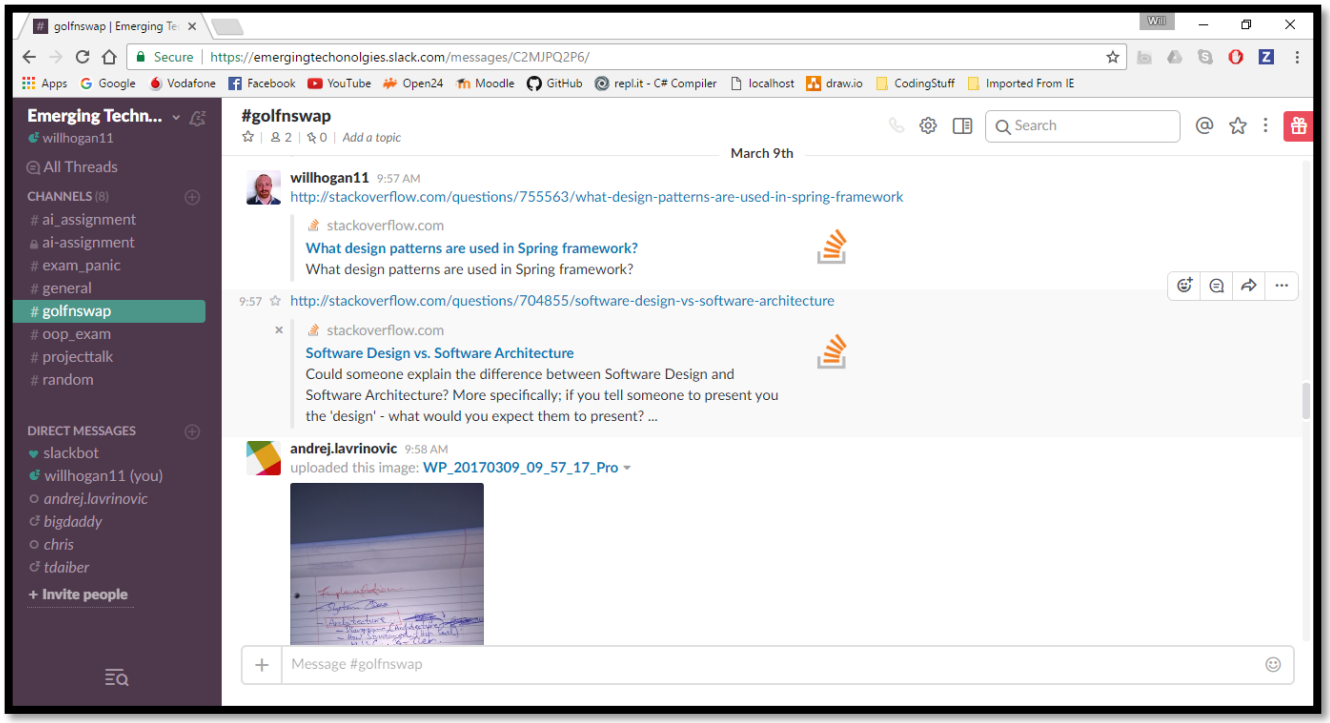


Figure 10: Slack Example

Google Drive

On a larger scale, Google Drive was used to collaborate on group documents, share larger files and work together in a cloud environment.

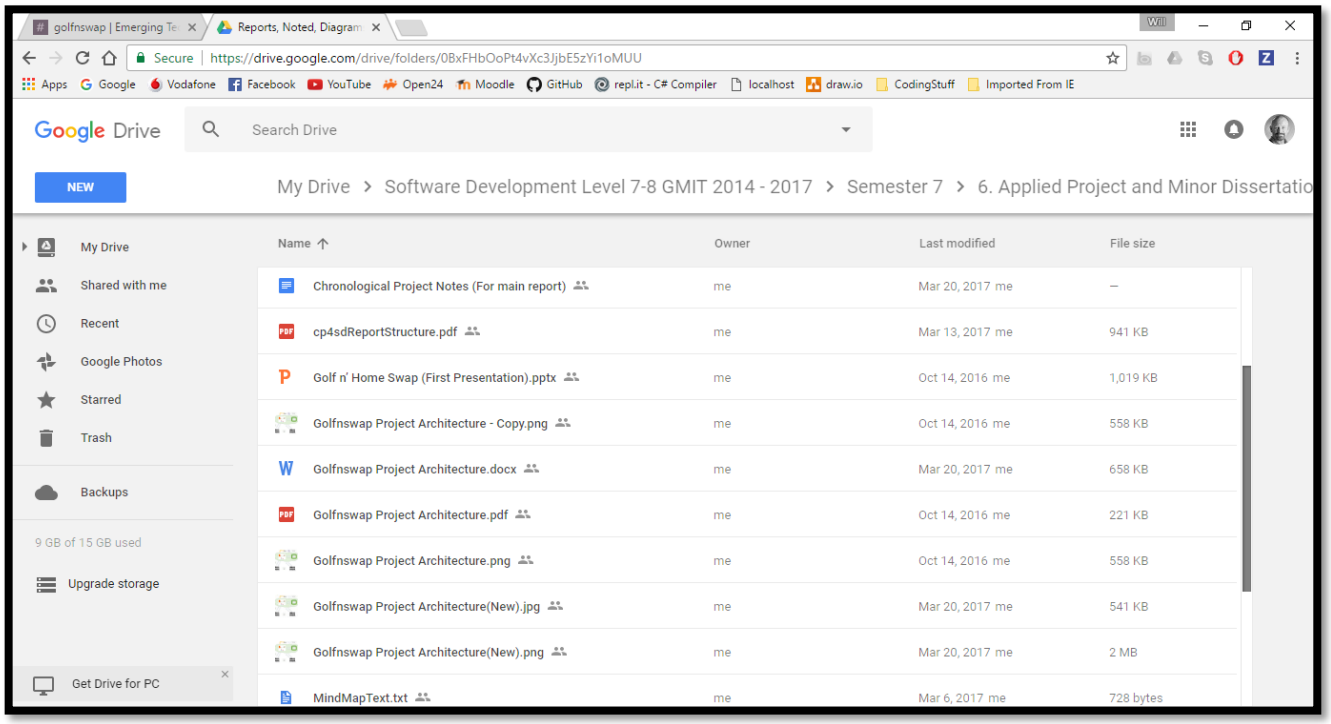


Figure 11: Google Drive

GitHub

For the project and version control management, GitHub was used. There are a plethora of different project management tools available on the software market and they all do specific things. GitHub was chosen because it offers a simple yet effective approach to project management. The team was able to use GitHub to track project tasks by creating milestones, attaching deadlines and creating issues associated with each milestone. This allowed deadlines to be added and assigned to different team members, which also allowed for bugs associated with each issue to be tracked. ZenHub was also used, which is a GitHub integration tool that allowed for Kanban style drag and drop issues and milestone management.

Here are some screenshots of how the project management features of GitHub were used;

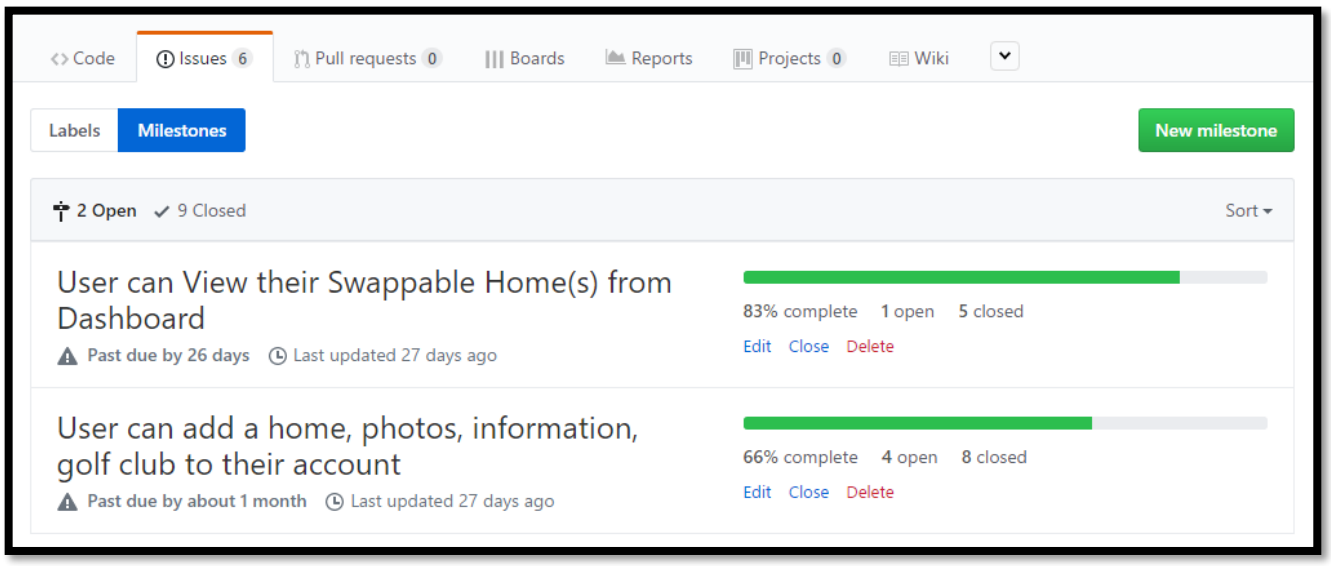


Figure 12: Milestones

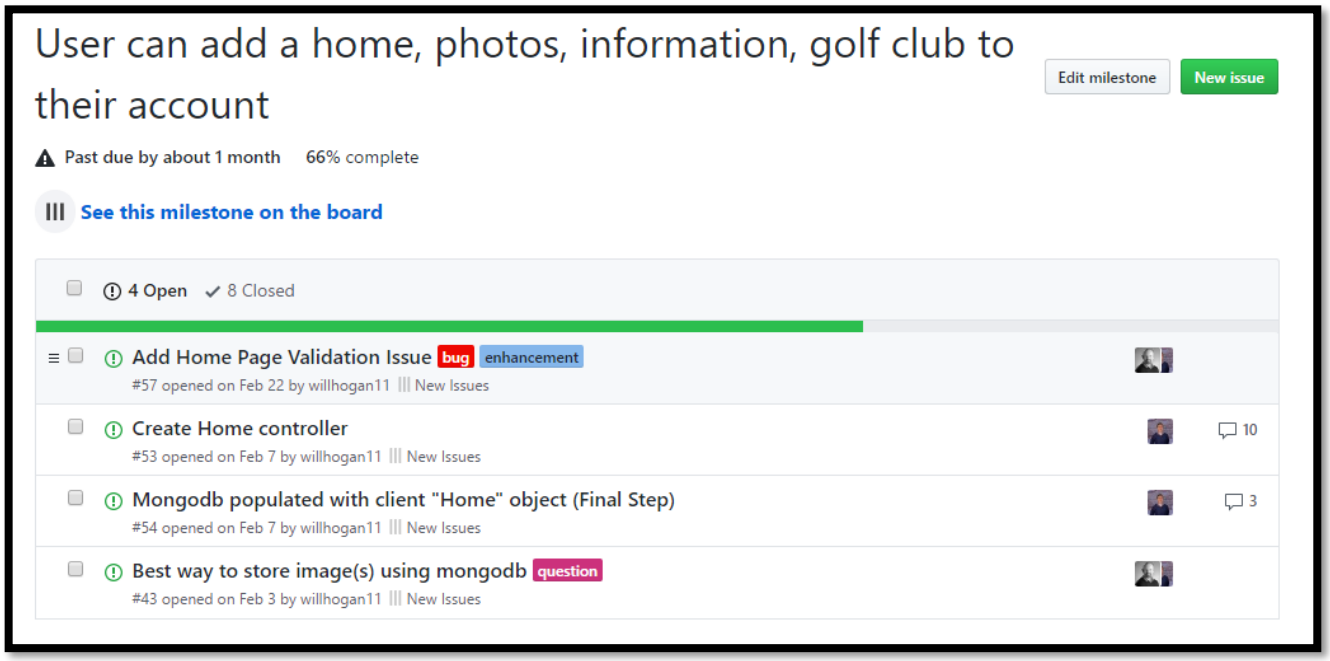


Figure 13: Issues

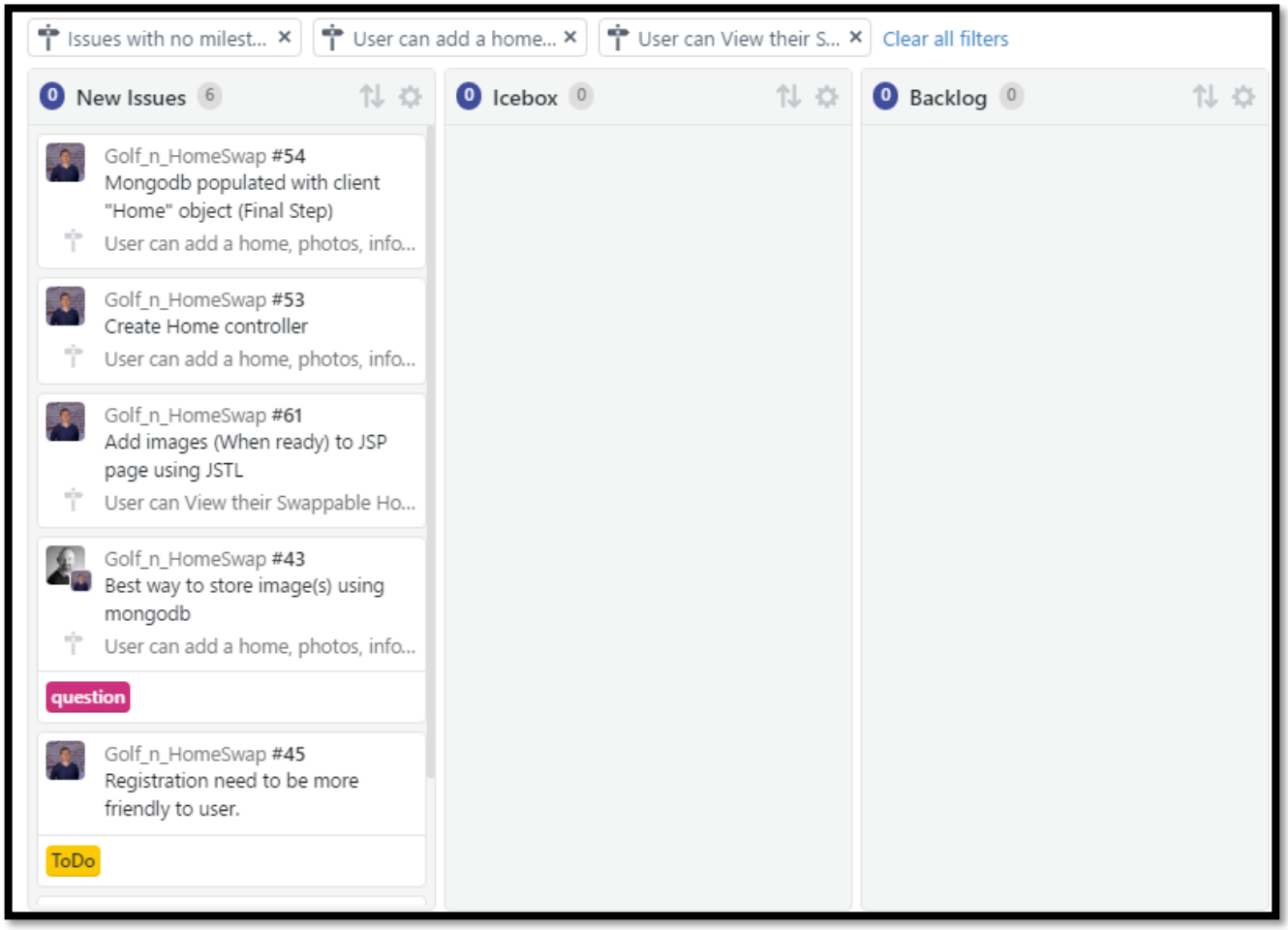


Figure 14: ZenHub Example

Collaborating on a project of this scale can be tricky, especially when more than one person is involved in the development process. GitHub was used to manage commits and branches. Each commit was a sizeable piece of work and then Slack which is a GitHub integration tool was used to inform the group slack channel whenever there were commits being pushed, branches being created or issues being opened or closed. Two branches were used for the project development, one was a prototype branch, which was a work in progress branch. Once this prototype section had been fully integrated and tested, it was pushed to the main Master branch.

CHAPTER 3: TECHNOLOGY REVIEW

Golf'n Home Swap is an enterprise web application and several technologies were used to build the applications;

- Eclipse IDE
- Java language
- Bootstrap
- Spring MVC framework
- Git version control
- Apache maven project manager
- Apache Tomcat web server
- Slack
- Mongo database
- Amazon AWS

Spring MVC

J2EE (Java to platform Enterprise Edition) is a platform that is used to develop Enterprise web applications **[10]**. Technically J2EE is a combination of JSP's (Java Server Pages), Java servlets and EJB (Enterprise Java Beans) modules. Those components offer developers the ability to build large-scale distributable applications. To build the Golf'n Home Swap system, the Spring MVC framework which is an implementation of the J2EE main framework was used.

Spring MVC is a lightweight framework. It consists of several modules such as IOC (Inversion of Control), AOP (Aspect Oriented Programming), DAO (Data Access Objects), Context, ORM (Object Relational Mapping), WEB MVC (Model View Controller) etc.

The advantages of the Spring framework are:

- **Predefined templates** – The Spring framework provides templates for Hibernate, JPA and other technologies. So, there is no need to write too much code. It hides the basic steps of these technologies.
- **Loose Coupling** - The Spring applications are loosely coupled because of dependency injection.
- **Easy to Test** - The Dependency Injection makes the application easier to test and it doesn't require a server.
- **Lightweight** – The Spring framework is lightweight because of its POJO (Plain Old Java Objects) implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface and is said to be non-invasive.
- **Fast developing** - The Dependency Injection is a feature of the Spring Framework and it supports various frameworks, which makes the development of a JavaEE application easier.
- **Powerful abstraction** - It provides powerful abstraction to JavaEE specifications
- **Declarative support** - It provides declarative support for caching, validation, transactions, and formatting.

Eclipse IDE

Eclipse is an integrated development environment whose projects are focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle. It is open source and very popular among developers. Its infrastructure allows downloading necessary modules and plugins in no time. It's very easy to install and use.

One of the advantages of eclipse is the effortless way that it integrates and adapts any java project into the editor. It can be achieved by having the configuration tools for runtime environment and management tools for jar libraries.

The Eclipse environment for the Golf'n Home Swap project consists of the runtime environment with Tomcat web server, Java Development Kit, Maven project manager, Spring MVC modules, and Web EE modules.

Java language

The Golf'n Home Swap application's code is written mostly in the java programming language **[8]**. Java is designed to have as few implementation dependencies as possible. It is class-based, and object-oriented. It allows developers to implement object-oriented principles such as inheritance, composition, polymorphism, abstraction, encapsulation etc. The Java platform consists of the Java application programming interfaces (APIs) and the Java virtual machine (JVM). Java APIs are libraries of compiled code that you can use in your programs. The Java virtual machine is used to run java applications.

Bootstrap

Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

Most of the elements on this applications web pages are implemented using the bootstrap framework. One of the significant advantages of using bootstrap is high capability with various browsers. It is a frequent problem when some web page elements work for one browser and don't for another or work in a different manner. Bootstrap reduces this risk to a minimum. It provides templates that help increase productivity and help reduce the developing time, which is very important.

Git

Version controlling for The Golf'n Home Swap project was performed by Git **[11]**. Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals. It allows for developers to build the system independently and safely. Git provides the remote service for storing the files – github.com.

Here are some key statistics collated from the Golf'n Home Swap repository;

- 170+ commits were made to the repository
- 3 remote branches and many local branches.
- 64 issues logged
- 11 Milestones created

Apache Maven

Maven **[9]** is a software project management tool and it was used during the Golf'n Home Swap system building. The general component of maven is a POM.xml file (Project Object Model). It is one of the projects configuration files and contains a list of dependencies or libraries that's used in the project. The integrated development environment such as Eclipse, that was used to develop the Golf'n Home Swap application contains the maven plugin that helps to manage this tool. By clicking on the POM file, eclipse provides an interactive user interface where configuration can be performing in a very comfortable manner using add, edit, remove buttons and other fields.

```
<dependencies>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>4.3.3.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>4.3.3.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>4.3.3.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>4.3.3.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>4.3.3.RELEASE</version>
  </dependency>
```

Maven also performs some project development activities such as;

- Build project
- Clean project
- Test
- Deploy
- Run

One of maven's feature is project deployment, which was used to deploy Golf'n Swap to the Amazon cloud service.

Apache Tomcat

Apache Tomcat is a web server that is used to run this system. It is an open-source implementation of the Java Servlet, Java Server Pages, Java Expression Language and Java Web Socket technologies. Basically, Tomcat is a container with all necessary environments required for web applications. The web app directory in the tomcat's infrastructure contains the projects that the server is running in. So, project deployment is a process of importing project's package into that directory. The project must be built and packed into war file and when it is done it's ready for deployment. The Tomcat web server has some security levels. The *tomcat-users.xml* and *manager.XML* files must be configured to have permission for project deployment.

Tomcat's default port is 8080. So, there is need to add ':8080' port to the IP address and include project name which is the same as the war file, for example, *192.168.0.25:8080/golfnswap*

Mongo DB

Mongo database is used in the Golf'n Home Swap to store and keep the system's data. MongoDB is an open-source, document database designed for ease of development and scaling. A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents. Key features of using Mongo DB are;

- High performance
- Rich query language
- High availability
- Horizontal scalability
- Support for multiple storage engines

Other technologies

Some other technologies and features were used to make the Golf'n Home Swap application, which helped to make the development process more productive and effective.

Those are;

- collaboration tools such as slack mobile and desktop version, Google drive, Microsoft outlook.
- Code editors other than eclipse, such as Microsoft Visual Studio Code, Notepad++, plain Notepad and Brackets.
- Utilities for different purpose such as wget, nano, git bash, ssh terminal and other.
- Graphics editors such as GIMP and Paint.net and Microsoft Word.
- Graph builders – Gliffy, draw.io and text2mindmap
- Operating systems – Windows, Linux Mint, RedHat.
- Amazon AWS cloud.
- Browsers: Google chrome, Microsoft edge, Firefox, Opera, Yandex.

CHAPTER 4: IMPLEMENTATION

System Architecture

The Golf'n Home Swap application has been completely created from the ground up using a methodical and structured approach. This was achieved by analysing the requirements that were received as part of the project specifications. As work began and the various pieces of how the System should be structured were pulled apart, it became more evident that large amounts of time were required to get the overall system architecture right from the get go, in order to facilitate a maintainable, robust and scalable Web application.

To begin with, analysis began by looking at what sort of users would be using the system and how they would interact with it. At the most basic level, several ways to be able to send and receive information were looked at, that allowed information to be sent from the user's side to the servers at the back end of the application. To put this into some sort of context and to create a visual perspective, the below Figure 1 diagram shows the System Architecture at a high level, which will be dissected and explained thoroughly throughout this chapter.

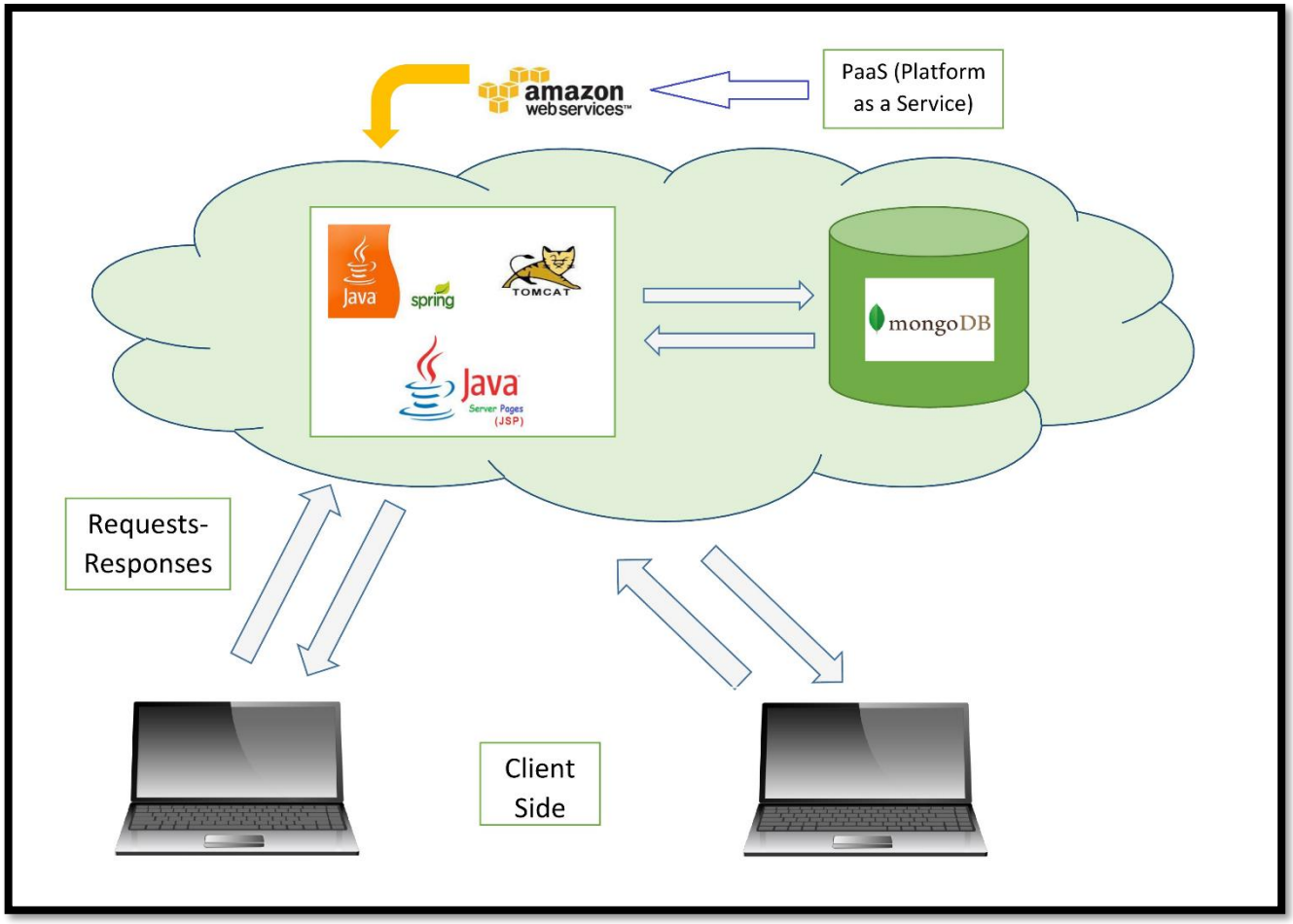


Figure 15: System Architecture

To explain what's going on in this diagram, it's important to highlight each part in a Layman's terms type of way. For example, let's suppose that a user wants to interact in some way with the Application, how would they go about it?

There were several core questions that needed to be asked from the outset to properly progress further. They were;

- Who are the users going to be?
- How will they be using/interacting with the Application?

- What devices will they be accessing the application from?
- How many users can the System hold at any one time?
- Will the system be storing information about a user?
- If so, how much information will be stored about the user and where?

By analysing the needs of the user, it became clear as to what type of infrastructure would be needed to make this work. Let's break these questions down by answering them one at a time.

Who are the users going to be?

The users are from a Golfing fraternity, and possible only have some or little knowledge about how an application like this would work, so the system has been designed in a straight forward and simple manner, so that a user can have an enjoyable experience when using the application.

How will they be using/interacting with the Application?

The user will be using the system, to initially apply for membership with the organisation. When they have been fully approved by the administration, the user will be able to create a full profile and add swappable homes to their account. In the future when the application is fully completed, the user will be able to message other users and setup house and Golf Club membership swaps from within the application.

What devices will they be accessing the application from?

The mindset was that there should be no restrictions about what devices or web browsers for that matter that a user can use to access the application. With these things in mind, it became clear that the team needed to focus on how the visual part of the application would be rendered in the various browsers, but also as mentioned using different devices, such as laptops, tablets, and phones.

How many users can the System hold at any one time?

Before starting to develop this application, it was imperative for the team to abandon the single-user mindset, indicative of the way in which a much smaller application operates, in favour of something that works on a larger scale. Concurrency was researched, which with respect to the application simply means that the system can hold many users simultaneously without crashing.

Will the system be storing information about a user?

It's an era where lots of information about people, in general, is stored and in several ways and so the amount of data being stored was minimised. However, there are certain prerequisites such as Name, Address information and password data that must be stored, in order for the application to be set up properly.

If so, how much information will be stored about the user and where?

Because this application is storing personal information about a user, it's important to keep this as far away as is possible from the main part of the system, to avoid anyone gaining unlawful access to it. To do this, users information was stored in a document database called MongoDB, which is itself completely separated from where the development work was done. This information is stored remotely in the cloud using Amazon Web services, which is a *PaaS* or Platform as a Service.

Detailed breakdown of Architecture

The system has several different levels that deal with various parts of each operation and so there is a lot more to this system than what a user sees on the home page of the web application. Information is being passed by the user when they click the Submit or login buttons, or when a different page of the application loads. This information is being handled by another tier or level.

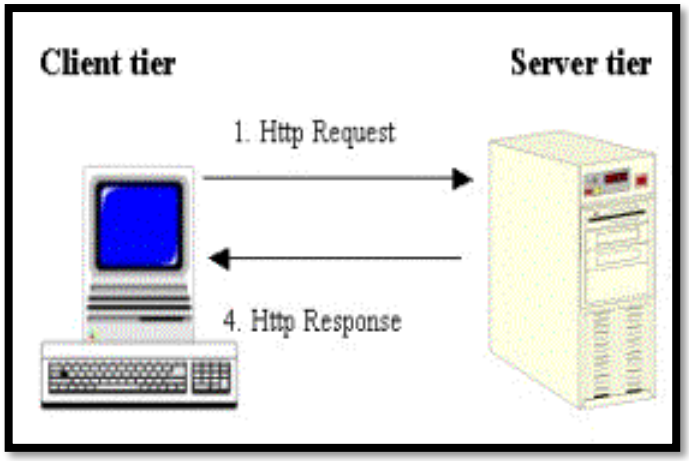


Figure 16: Request/Response

In Figure 2, there is a line of communication between what the users interacts with (The Client Tier) and the server (The Server Tier). This communication is made using an *HTTP (Hypertext Transfer Protocol)* request. This level of communication is specific to Web pages and allows for information to be sent with a request, for example, get a specific web page from the server and display it on the client's screen. This request is processed by the Server and if the page exists, the server will return the page in what's called a response.

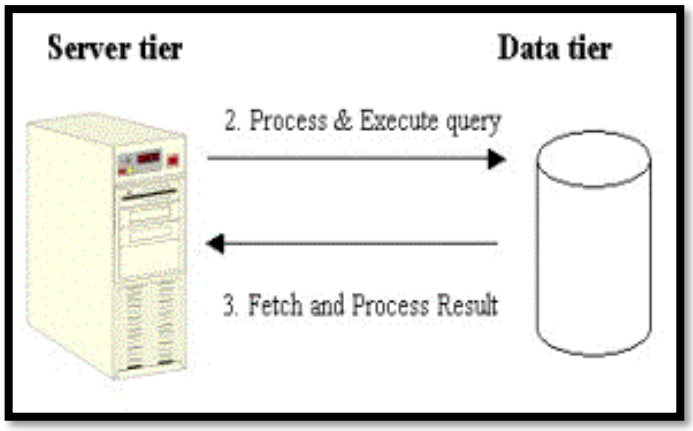


Figure 17: Server/Data Tiers

In Figure 3, the server processes the client's request and sends the request to the Database Tier, which returns the result from the database. This result is then processed by the server and returned to the client side and displayed on the web page.

HTTP Requests

As briefly mentioned, HTTP is a way of communicating with various parts of a Web Application using various commands. There are several of them but the main ones that are relevant to the application are GET and POST requests.

GET

As its name might suggest, a GET request is trying to retrieve some information. This information could be a user trying to navigate to a specific page, but the info would firstly need to be retrieved in order for it to be displayed for the user. This is what's called a GET request.

POST

A POST request, on the other hand, is a request that sends information associated with a user when they click a button, this information is processed elsewhere in the system. An example of a POST request might be a user who is trying to apply for Golf'n Home Swap membership. When they have filled out the required application information fields and clicked the Apply button, then this is what's known as a POST request.

But let's say that a user wants to retrieve information, specific to a user. For example, they want to view someone else's Home and Golf Club that they may wish to message and swap with. The topology in the above Figure won't suffice as the server doesn't hold that sort of information. What is needed is another Tier or level to deal with this sort of request. The following section discusses how these issues were dealt with.

Model, View, Controller (M.V.C)

The type of Architecture that was chosen for this project is called M.V.C, which stands for Model, View, Controller. Essentially what this means is that the information takes a journey from A to B by following a certain pattern, something similar to what has been seen in previous figures. The whole idea behind the M.V.C model is to provide several layers of abstraction between what the user does and how the information is processed. This means that a user will have no knowledge of what's happening once they click on a button displayed on their side, knowledge of how a system separates and where the information is stored should not concern the user and with the MVC model, there is a clear separation of concerns that must be adhered to.

To shed some light on this, let's have a look at the below figure;

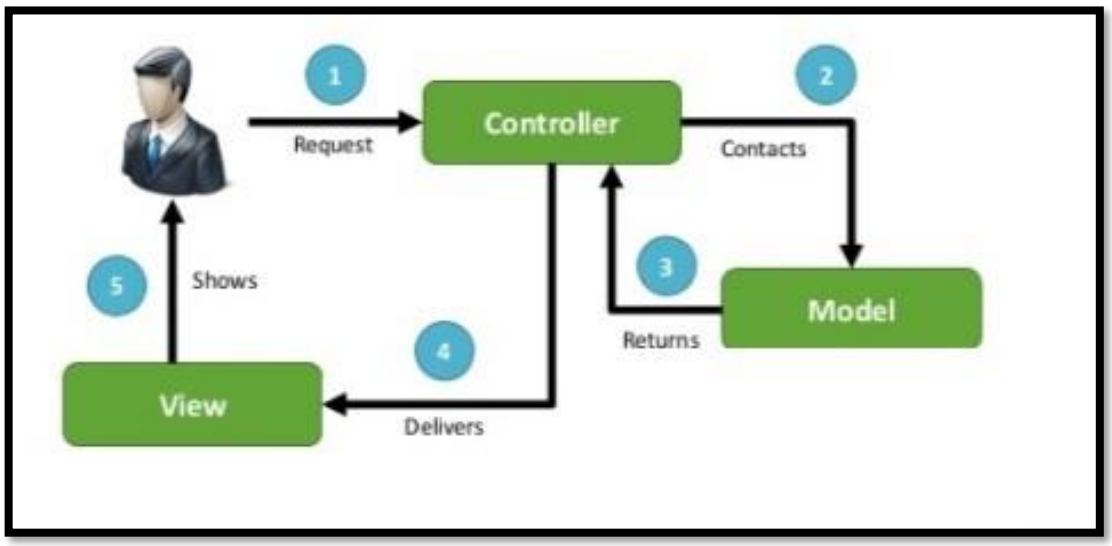


Figure 18: Model, View, Controller

View

The View is essentially what the user of the system sees. Like the Client Tier in Figure 2, it is the visual perspective, the web page that displays the parts of the application that a user can interact with. These parts could be controls such as text boxes that allow a user to input text or buttons that submit user information.

In the case of this application, here's what the homepage view looks like;

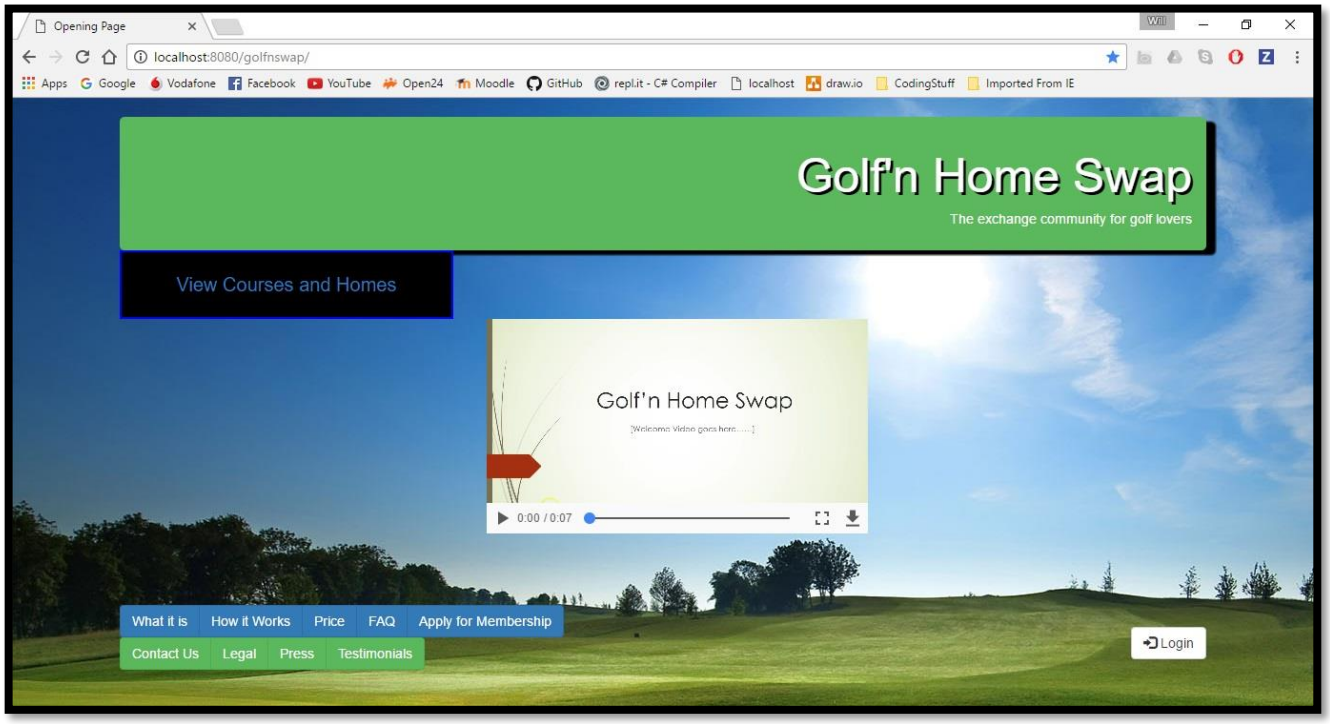


Figure 19: View Example

Model

The model refers to the actual data that's being used and stored in the system. Like the Data Tier in Figure 3, this data could be a Golf'n Home Swap applicant, whose data consists of Name, Address, and Golf Club information.

Here's an example of how Model information is stored using MongoDB;

Figure 20: Model Example

Controller

The Controller is the section of the system that deals with communication between the View and the Model and will typically be responsible for reading requests, making database calls and updating the View.

Here's some of the Java code associated with the controller section of the application;

```
@RequestMapping(value="/docreate", method=RequestMethod.POST)
public String addUser(@ModelAttribute("user") User user, Model model){

    User u = userRepo.findByEmail(user.getEmail());

    if(u != null){
        model.addAttribute("userEmailIsNotUnique", "User with provided email is allready exists.");
        log.info("User " + u.getEmail() + " is allready exists.");
        return "apply";
    }

    user.setPassword();
    log.info("Browser: " + user.getFirstname());
    log.info("Date of membership issued -- " + user.getDateofissue().toString() + "\n");
    log.info("UserController--addUser => " + user);
    userRepo.addUser(user);

    return "successapply";
}
```

Figure 21: Example Controller Code

Let's have a look at a use case situation just for clarity. Suppose a user wants to submit an application to Golf'N Home Swap, here are the various steps and the part that each section plays in the process.

- **The View**
    - A user enters the website clicks on the Apply for membership button.

- They proceed to enter in the relevant information and click the Apply button.

- **The Controller**
  - Receives the request from the View.
  - Parses through the information and sees that it's a POST request.
- **The Model**
  - The post request code makes a call to open communication with the database.
  - The user's information is inserted into the database.
- **The Controller**
  - Receives information that the insert was entered successfully.
  - The controller sends word to the View that the Application was successful.
- **The View**
  - Informs the user on the screen that their application has been entered successfully.

Spring MVC

Spring is a Java Framework **[1]** that was used to create this application and it builds on the MVC architecture previously detailed in the last number of pages and is extremely useful and easy to work with. The whole idea of this framework is to scaffold a project in terms of managing the various application dependencies, such as Tomcat Web servers and general application components required to help a more robust and efficient development environment.

The below Figure helps to illustrate how a typical Spring MVC architecture operates. This will be detailed more in the System Design section of this Chapter.

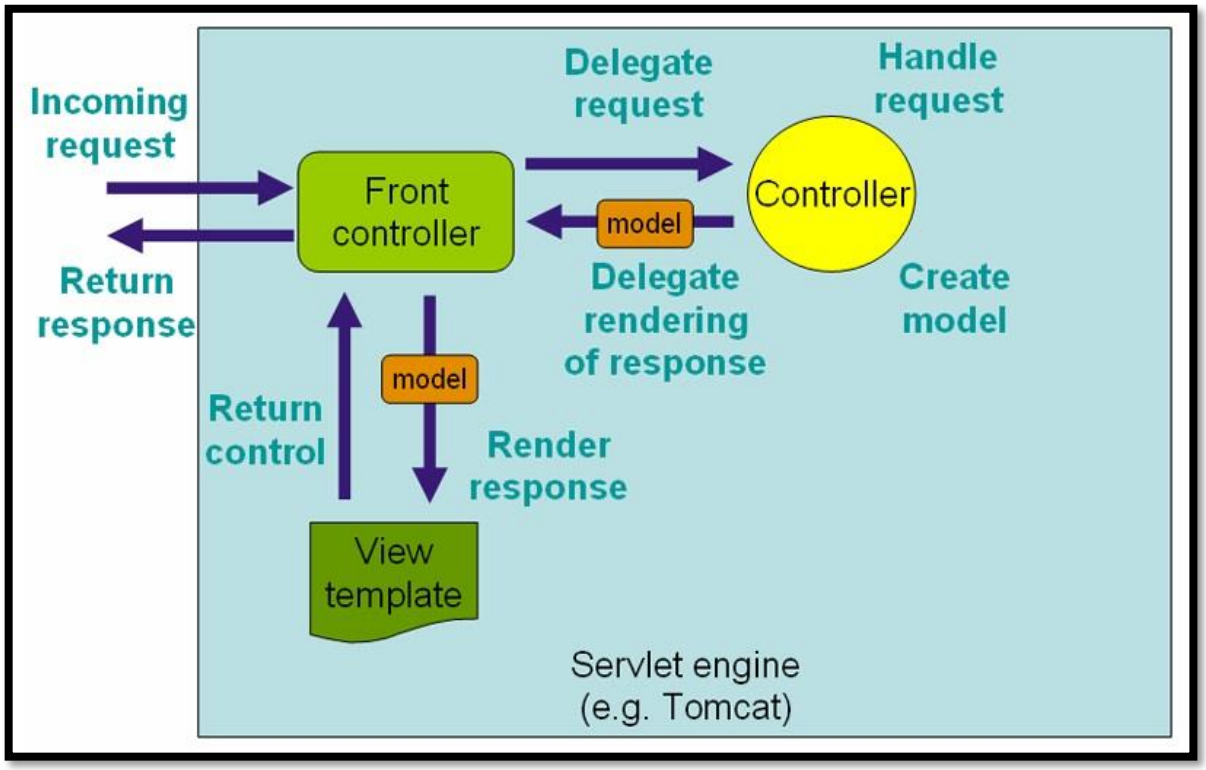


Figure 22: Spring MVC Example

Whilst it's important to look at the greater picture as seen in the System Architecture diagram earlier in this Chapter, it's equally important to acknowledge the structure of the java classes that have been created by the developers during the project development stage. The below figure illustrates how the classes of this application communicate with each;

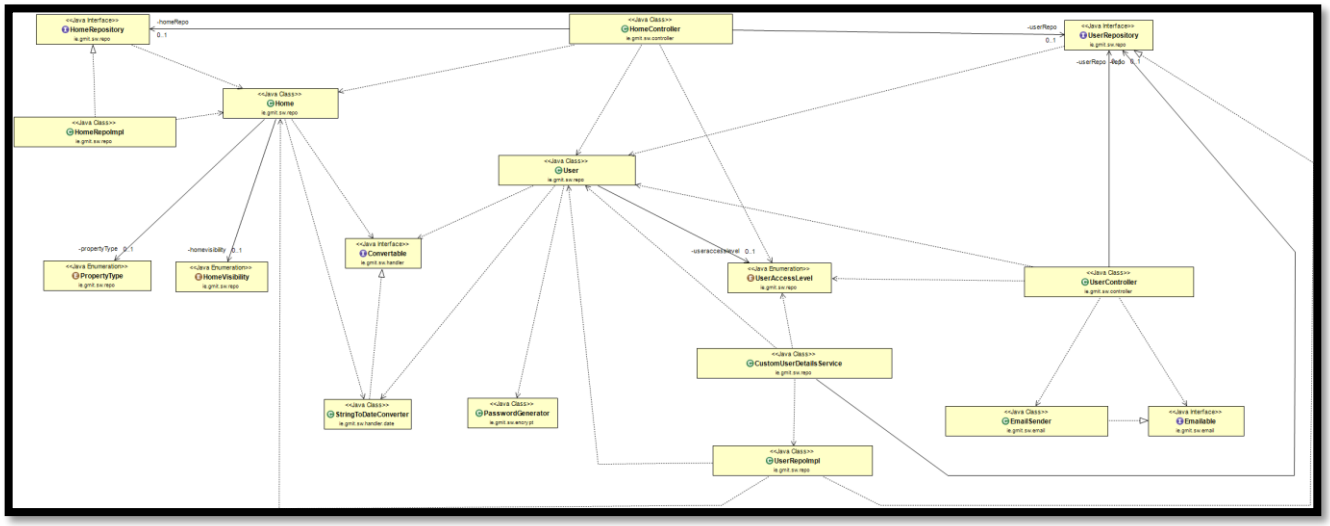


Figure 23: Project Class Diagram

System Design

Systems are not built in a vacuum but are created in order to meet the needs of the users and aspects such as architecture, components, modules, interfaces and data that must be defined for the system. This process of defining things about what the system is going to be built on is called System design. Normally this phase commences after the system analysis part of the development cycle has been completed. It is important to understand what the system is for, what functionality it needs to implement and what types of data will be used. After deciding with the system's owner about the general high-level structure of the applications, applications requirements are then organised.

System Modules

The "Golf'n swap" system assumes that the users of the application are home owners and have preferences to play golf. They must be a member of any golf club. This system needs to be controlled and managed by the administration side (system owner or privileged users). New users must be allowed to apply for membership. The administration must have a flexible way to respond to an application (accept, reject, postponed, restore etc.). The administration must have a way to notify users of the application outcome and users must be able to contact administration or other users. The available information must be displayed if requested.

Based on information about the system and requirements, the team were able to ascertain that the system must contain the following;

- Account management system.
- Administration system.
- Data management system.
- Messaging system.
- Security system

Information provided by the user is handling by the user controller - "*UserController.java*". The method responsible for adding a user to the system *(addUser(...) {...})* performs several steps of the process.

The following Figure depicts java code from the *addUser* controller section of the system;

```
49
50    @RequestMapping(value="/docreate", method=RequestMethod.POST)
51    public String addUser(@ModelAttribute("user") User user, Model model){
52
53        // 1) Validate email: must be unique
54
55        User u = userRepo.findByEmail(user.getEmail());
56
57        if(u != null){
58            model.addAttribute("userEmailIsNotUnique", "User with provided email is allready exists.");
59            log.info("User " + u.getEmail() + " is allready exists.");
60            return "apply";
61        }
62
63        user.setPassword();
64        log.info("Browser: " + user.getFirstname());
65        log.info("Date of membership issued -- " + user.getDateofissue().toString() + "\n");
66        log.info("UserController--addUser => " + user);
67        userRepo.addUser(user);
68
69        return "successapply";
70    }
71
```

Figure 24: addUser controller

1. The first part is validating each distinct email address. The system tries to retrieve the document (record) from the database, based on the email provided by the user. The email is considering valid if the requested result is null. That means, there is no such email previously stored in the database and the email provided by the user is unique. If the document is found (email already exists), then the system response is to redirect the user back to the *apply* form and with a relevant error message: *"User with provided email already exists."*

2. Upon successful email validation, the system generates a temporary password by using a custom made password generator tool (*ie.gmit.sw.encrypt.PasswordGenerator.java*).

3. In the end, all the data provided by the user is stored in the database.

Approved users will have access to the account dashboard page and be allowed to create and link a property to their account. After successful authorisation, there is an interactive button that appears on the home page. By clicking on it, a user is redirecting to the add a home wizard. The process of completing this part of the account configuration is divided into the following sections;

- Home Type form,
- Home Details form,
- Location map,
- Availability schedule,
- Add Photos uploader

Before the start of adding a property to the system, the user is asked to watch the guide video about how to complete the add a home forms. There is an option to skip the tutorial. After the video instruction, the user is redirecting to the wizard page where the first section is the Home Type form. After completing the first section and clicking the next button page is swapped with the second form and so on up to the last section which is Add Photos uploader. By submitting "*Add Home*", the frontend validation process is invoked and if there is no missing or incorrect information, then the information is transported to the server and is handled by another controller method which is called *"createHome(...){...}"*:

The following Figure depicts java code from the *createHome* controller section of the system;

```
131
132    @RequestMapping("/docreatehome")
133    public String createHome(@ModelAttribute("home") Home home, Model model, Principal principal){
134
135        log.info("Testing for controller >>> success");
136        log.info("Home >>> \n" + home.toString());
137
138        // 1.1) Find user email
139        String userEmail = getUsername(principal);
140
141        // 1.2) Get user info
142        // retrieving user infoattributeValue
143        User user = userRepo.findByEmail(userEmail);
144        log.info("User: " + user.toString());
145        // bind user information
146        String username = user.getFirstname() + " " + user.getSurname();
147        model.addAttribute("username", username);
148        model.addAttribute("email", userEmail);
149
150        // 2) Link home to the user
151        home.setUserEmail(userEmail);
152
153        // 3) Add home to the db:
154        homeRepo.addHome(home);
155
156        return "dashboard";
157    }
158
```

Figure 25: createHome controller

The first part in the above Figure in the method is to retrieve the user's name and email and set those for "Home" object as each home must be associated with that user (belong to the user). If everything is fine, then the home is stored in the database. Users with membership can list and see his/her house and add another one if they wish. A user is also able set their home as private or public. Depending on this state, the property details are either shared among users or not.

Account Management System (AMS)

The account management system provides a user-friendly module to create and manage an existing account within the system. To create the account, a user must first apply for membership by completing the application form. Requests are then considered and approved by administration personnel. The user is notified about any outcome made by administration through email. Upon application success, the user will use their temporary password, which he/she must change for a permanent one. Approved users will then have access to the account page.

To implement AMS in the system, several interactive JSP's (Java Server Pages) were created such as;

- **apply.jsp** – contains the application form.
- **dashboard.jsp** – Main user page (When logged in).
- **firsttimelogin.jsp** – Contains a form that allows a user to change their temporary password for a permanent password.
- **addhome.jsp** – A page where a user can add information about their property.
- **addhomewizard.jsp** – Guide to adding home, using a wizard type video.
- **successapply.jsp** – Confirms that the application form completed by the user is valid and sent.

Information provided by the user is handling by the user controller - "UserController.java". The method responsible for adding a user to the system (addUser(...) {...}) performs several steps of the process.

```
49
50    @RequestMapping(value="/docreate", method=RequestMethod.POST)
51    public String addUser(@ModelAttribute("user") User user, Model model){
52
53        // 1) Validate email: must be unique
54
55        User u = userRepo.findByEmail(user.getEmail());
56
57        if(u != null){
58            model.addAttribute("userEmailIsNotUnique", "User with provided email is allready exists.");
59            log.info("User " + u.getEmail() + " is allready exists.");
60            return "apply";
61        }
62
63        user.setPassword();
64        log.info("Browser: " + user.getFirstname());
65        log.info("Date of membership issued -- " + user.getDateofissue().toString() + "\n");
66        log.info("UserController--addUser => " + user);
67        userRepo.addUser(user);
68
69        return "successapply";
70    }
71
```

Figure 26: AMS Example

1. The first part is validation each distinct email address. The system tries to retrieve the document (record) from the database, based on the email provided by the user. The email is considering valid if the requested result is null. That means, there is no such email previously stored in the database and the email provided by the user is unique. If the document is found (email already exists), then the system response is to redirect the user back to the *apply* form and with a relevant error message: *"User with provided email already exists."*

2. Upon successful email validation, the system generates a temporary password by using a custom made password generator tool (*ie.gmit.sw.encrypt.PasswordGenerator.java*).

3. In the end, all the data provided by the user is stored in the database.

Approved users will have access to the account dashboard page and be allowed to create and link a property to their account. After successful authorisation there is an interactive button that appears on the home page. By clicking on it, a user is redirecting to the add a home wizard. The process of completing this part of the account configuration is divided into the following sections;

- Home Type form,
- Home Details form,
- Location map,
- Availability scheduler,
- Add Photos uploader

Before the start of adding a property to the system, the user is asked to watch the guide video about how to complete the add a home forms. There is an option to skip the tutorial. After the video instruction, the user is redirecting to the wizard page where the first section is the Home Type form. After completing the first section and clicking the next button page is swapped with the second form and so on up to the last section which is Add Photos uploader. By submitting "*Add Home*", the frontend validation process is invoked and if there is no missing or incorrect information, then the information is transporting to the server and is handled by another controller method which is called *"createHome(...){...}"*:

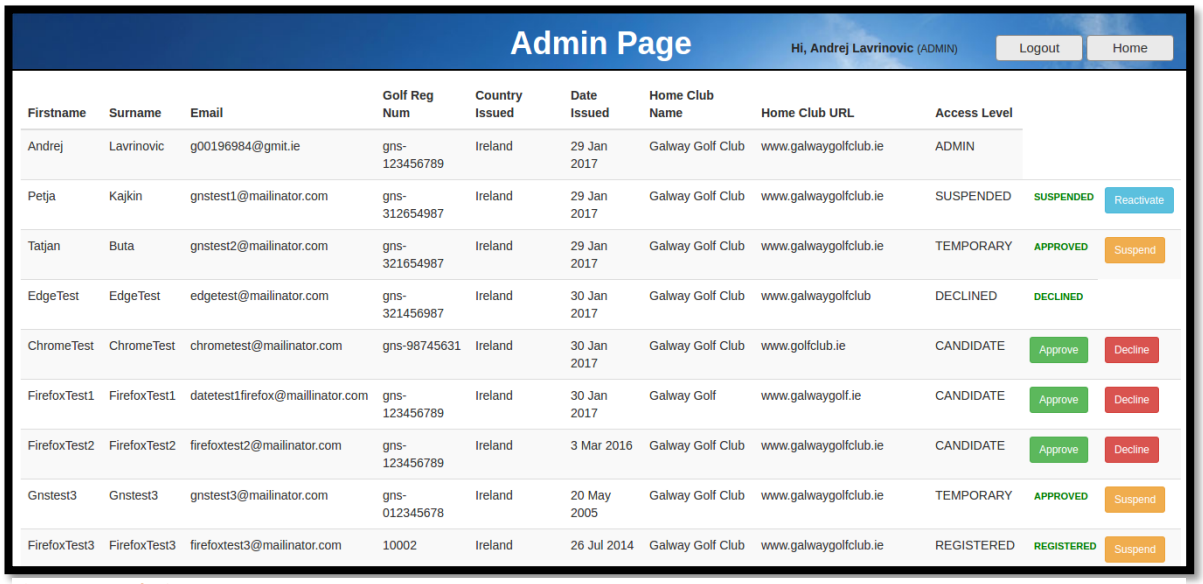
```
131
132     @RequestMapping("/docreatehome")
133     public String createHome(@ModelAttribute("home") Home home, Model model, Principal principal){
134
135         log.info("Testing for controller >>> success");
136         log.info("Home >>> \n" + home.toString());
137
138         // 1.1) Find user email
139         String userEmail = getUsername(principal);
140
141         // 1.2) Get user info
142         // retrieving user infoattributeValue
143         User user = userRepo.findByEmail(userEmail);
144         log.info("User: " + user.toString());
145         // bind user information
146         String username = user.getFirstname() + " " + user.getSurname();
147         model.addAttribute("username", username);
148         model.addAttribute("email", userEmail);
149
150         // 2) Link home to the user
151         home.setUserEmail(userEmail);
152
153         // 3) Add home to the db:
154         homeRepo.addHome(home);
155
156         return "dashboard";
157     }
158
```

Figure 27: Adding a user's home

The first thing in the method is to retrieve the user's name and email and set those for "*Home*" object as each home must be associated with that user (belong to the user). If everything is fine, then the home is stored in the database. Users with membership can list and see his/her house and add another one if they wish. A user is also able set their home as private or public. Depending on this state, the property details are either shared among users or not.

Administration System

When the user has submitted their application form for membership, the system administrator or system owner needs to approve it. The administration system allows the admin personnel to manage this process. The system administrator has the same details as the user. If "*useraccesslevel*: ADMIN", then the user has super privileges, and the user will have access to the admin page. There is an "*Admin*" button, that is visible at the home but only for admin personnel. The admin page contains the list with details of all members and applicants. The very right interactive panel contains status information and buttons for changing this status;



Firstname	Surname	Email	Golf Reg Num	Country Issued	Date Issued	Home Club Name	Home Club URL	Access Level		
Andrej	Lavrinovic	g00196984@gmit.ie	gns-123456789	Ireland	29 Jan 2017	Galway Golf Club	www.galwaygolfclub.ie	ADMIN		
Petja	Kajkin	gnstest1@mailinator.com	gns-312654987	Ireland	29 Jan 2017	Galway Golf Club	www.galwaygolfclub.ie	SUSPENDED	SUSPENDED	Reactivate
Tatjan	Buta	gnstest2@mailinator.com	gns-321654987	Ireland	29 Jan 2017	Galway Golf Club	www.galwaygolfclub.ie	TEMPORARY	APPROVED	Suspend
EdgeTest	EdgeTest	edgetest@mailinator.com	gns-321456987	Ireland	30 Jan 2017	Galway Golf Club	www.galwaygolfclub	DECLINED	DECLINED	
ChromeTest	ChromeTest	chrometest@mailinator.com	gns-98745631	Ireland	30 Jan 2017	Galway Golf Club	www.golfclub.ie	CANDIDATE	Approve	Decline
FirefoxTest1	FirefoxTest1	datetest1firefox@maillinator.com	gns-123456789	Ireland	30 Jan 2017	Galway Golf	www.galwaygolf.ie	CANDIDATE	Approve	Decline
FirefoxTest2	FirefoxTest2	firefoxtest2@mailinator.com	gns-123456789	Ireland	3 Mar 2016	Galway Golf Club	www.galwaygolfclub.ie	CANDIDATE	Approve	Decline
Gnstest3	Gnstest3	gnstest3@mailinator.com	gns-012345678	Ireland	20 May 2005	Galway Golf Club	www.galwaygolfclub.ie	TEMPORARY	APPROVED	Suspend
FirefoxTest3	FirefoxTest3	firefoxtest3@mailinator.com	10002	Ireland	26 Jul 2014	Galway Golf Club	www.galwaygolfclub.ie	REGISTERED	REGISTERED	Suspend

Figure 28: Admin page

Records of users who applied for membership provide two options for the administrator - "*Approve*" and "*Decline*". Users who have been declined have no access to the system, but their details remain stored in the database with status "*DECLINED*". Approved users must set a permanent password and then their "*useraccesslevel*" is changed from "*APPROVED*" to "*REGISTERED*".

The system administrator can suspend registered or approved users if necessary and if issues are cleared up, that user can have their privileges reinstated. Suspended users also have no access to the AMS (Account Management System).

Data Management System

The Golf'n Swap system makes it possible for end users with privileges to communicate with data. By using this application, potential candidates can create data and store it (write) in the database. Members of the system can retrieve information (read). Administration or the system owner can update or remove data. From a Database perspective, the application uses mongo database. The reason for choosing this program is in its definition. It is free, open-source, cross-platform, document-oriented, classified as NoSQL database. MongoDB uses BSON (JSON-like) documents with schemas and can run over multiple servers, which makes it very flexible. This becomes a very important feature of MongoDB, as this system matures. One of the significant areas were MongoDB was used, was in Spring and using a POJO (Plain Old Java Object), centric model. It allows interaction with database collections and easily, that can write a Repository style data access layer (DAL), which is *ie.gmit.sw.repo* in this application.

Here's a screenshot of the java repository code folder;

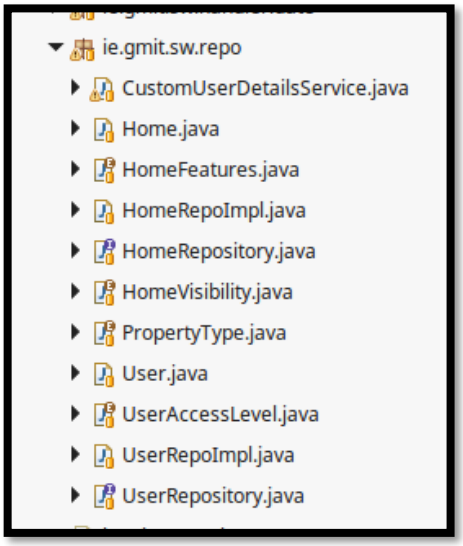


Figure 29: java repo view

Dependencies

The *spring-data-mongdb* and mongo-java-driver repositories are used for MongoDB integration into the system. It follows the Spring Data-centric approach and comes with flexible and complex API operations, based on well-known data access patterns in all Spring Data projects. Here's an example;

```
81
82      <!-- mongodb java driver -->
83      <dependency>
84          <groupId>org.mongodb</groupId>
85          <artifactId>mongo-java-driver</artifactId>
86          <version>2.11.0</version>
87      </dependency>
88
89
90      <!-- Spring data mongodb -->
91      <dependency>
92          <groupId>org.springframework.data</groupId>
93          <artifactId>spring-data-mongodb</artifactId>
94          <version>1.7.2.RELEASE</version>
95      </dependency>
96
```

Figure 30: mongodb dependency example

Configuration

With *context-annotation* namespace, the MongoDB API is connected to the *ie.gmit.sw.repo*, which is a representation of the document access layer. Also, the configuration file contains the mongo access control and database configuration, with parameters that are encapsulated in the *mdb.properties* file. The *MongoTemplate* class contains all useful mongo operations and it is also injected using a bean configuration.

Another bean *PersistenceExceptionTranslationPostProcessor* was implemented in configuration. This element is necessary to automatically apply persistence exception translation to any bean marked with *@Repository* annotation. It is also responsible for translating the native resource exceptions DataAccessException hierarchy.

```
    <context:annotation-config></context:annotation-config>
    <context:component-scan base-package="ie.gmit.sw.repo"></context:component-scan>

    <context:property-placeholder location="classpath:/ie/gmit/sw/config/resources/mdb.properties" />




    <mongo:mongo host="${mdb.host}" port="${mdb.port}">
        <mongo:options
            connections-per-host="${mdb.connectionsPerHost}"
            threads-allowed-to-block-for-connection-multiplier="${mdb.threadsAllowedToBlockForConnectionMultiplier}"
            connect-timeout="${mdb.connectTimeout}"
            max-wait-time="${mdb.maxWaitTime}"
            auto-connect-retry="${mdb.autoConnectRetry}"
            socket-keep-alive="${mdb.socketKeepAlive}"
            socket-timeout="${mdb.socketTimeout}"
            slave-ok="${mdb.slaveOk}"
            write-number="1"
            write-timeout="0"
            write-fsync="${mdb.slaveOk}"/>
    </mongo:mongo>

    <mongo:db-factory dbname="${mdb.dbname}" mongo-ref="mongo" />

    <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
        <constructor-arg name="mongo" ref="mongo" />
        <constructor-arg name="databaseName" value="${mdb.dbname}" />
    </bean>

    <bean
        class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor">
    </bean>
</beans>
```

Figure 31: Mongo configuration file

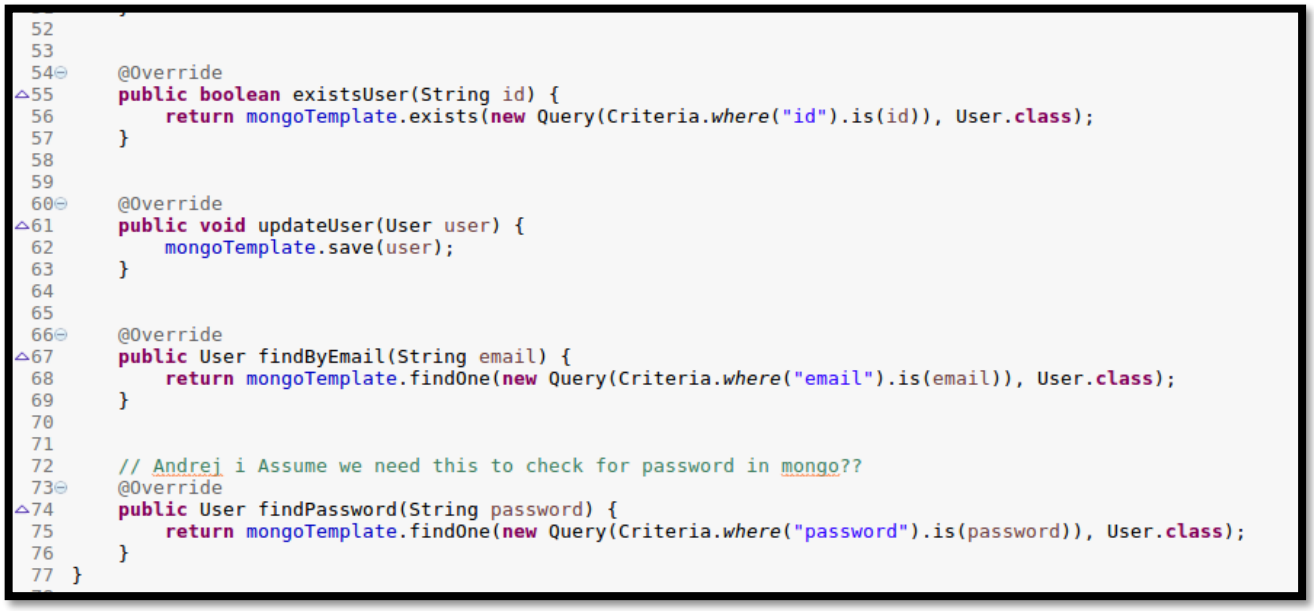
Repositories

Repositories in the Golf'n Swap project are beans that carry all functionality associated with data access. Spring-data-MongoDB provides a very effective API that implements CRUD (create-read-update-delete) functionality. The advantage of this approach is to avoid writing a class that implements a customer repository with plain queries. In most cases the *SpringDataMongoDB* creates it on the fly when the application is running.

```
1  package ie.gmit.sw.repo;
2
3  import java.util.List;
6
7  public interface UserRepository extends Repository<User, Long>{
8
9      public List<User> findAllUsers();
10     public void addUser(User user);
11     public User findUserById(String id);
12     public User findByEmail(String email);
13     public Long numberOfUsers();
14     public void delUser(User user);
15     public boolean existsUser(String id);
16     public void updateUser(User user);
17     public User findPassword(String password);
18 }
19
```

Figure 32: Repository example

The system uses *MongoTemplate* to execute queries. As it can be seen from the figure above, there are other queries that are defined by simply declaring their method signature. In *UserRepository* there are findUserByEmail(String email), existsUser(String id) and findPassword(String password). The implementation of customer queries needs its definition which exists in the UserRepoImpl class as shown in the below figure;



```
52
53
54      @Override
55      public boolean existsUser(String id) {
56          return mongoTemplate.exists(new Query(Criteria.where("id").is(id)), User.class);
57      }
58
59
60      @Override
61      public void updateUser(User user) {
62          mongoTemplate.save(user);
63      }
64
65
66      @Override
67      public User findByEmail(String email) {
68          return mongoTemplate.findOne(new Query(Criteria.where("email").is(email)), User.class);
69      }
70
71
72      // Andrej i Assume we need this to check for password in mongo??
73      @Override
74      public User findPassword(String password) {
75          return mongoTemplate.findOne(new Query(Criteria.where("password").is(password)), User.class);
76      }
77  }
```

Figure 33: Custom Mongo Template

Messaging System

The Spring Framework provides a helpful utility library for sending emails, that shields the user from the specifics of the underlying mailing system and is responsible for low-level resource handling on behalf of the client. The *org.springframework.mail* package is the root level package for the Spring Framework's email support. The central interface for sending emails is the *MailSender* interface; a simple value object encapsulating the properties of a simple mail such as from and to (plus many others) is the *SimpleMailMessage* class. This package also contains a hierarchy of checked exceptions which provide a higher level of abstraction over the lower level mail system exceptions with the root exception being *MailException*.

The org.springframework.mail.javamail.JavaMailSender interface adds specialised *JavaMail* features such as MIME message support to the MailSender interface (from which it inherits). *JavaMailSender* also provides a call back interface for preparation of *JavaMail* MIME messages, called org.springframework.mail.javamail.MimeMessagePreparator.

MS Components

As *MailSender* is a very well encapsulated *JavaMail*, the **javax.mail.jar** library needs to be added to the pom file. Also, there is a need to use org.springframework.mail package to use *MailSender*.

The application requires the following dependencies handled by the pom file:

Library Name	**Version**	**Description**
Spring-Context-Support	4.3.5	Classes supporting the org.springframework.context package, such as abstract base classes for *ApplicationContext* implementations and a *MessageSource* implementation.
Javax-mail-api	1.5.6	Set of abstract APIs that model a mail system

Mail	1.4.7	The JavaMail reference implementation jar file, including the SMTP, IMAP, and POP3 protocol providers

Java-mail-api is a general component that provides abstract API's that model a mail system. It uses a platform and protocol independent framework that builds Java technology based on the email client applications and is also a required part of the Java Platform, Enterprise edition (Java EE).

Implementation

There are three general components required for *JavaEmail* to manage sending emails:

- User Interface
- Configuration
- Executable mechanism

UI (User Interface)

In the Golf'n Swap system, the *JavaMail* is used as a notification mechanism to inform users, such as system administrators or regular users with information about the state of their account. When the candidate applies for membership, he/she must provide a valid email, which is then used as the destination address used by the Email Management System. The sending mechanism is fully automated and the UI is nothing more than as interactive panel with buttons. For example, taking the administrator page that lists all users. If a user with higher priority (e.g. system administrator) decides to change their status for another user, it can be done by clicking the associated button, and appropriate message with notification will be sent to that user.

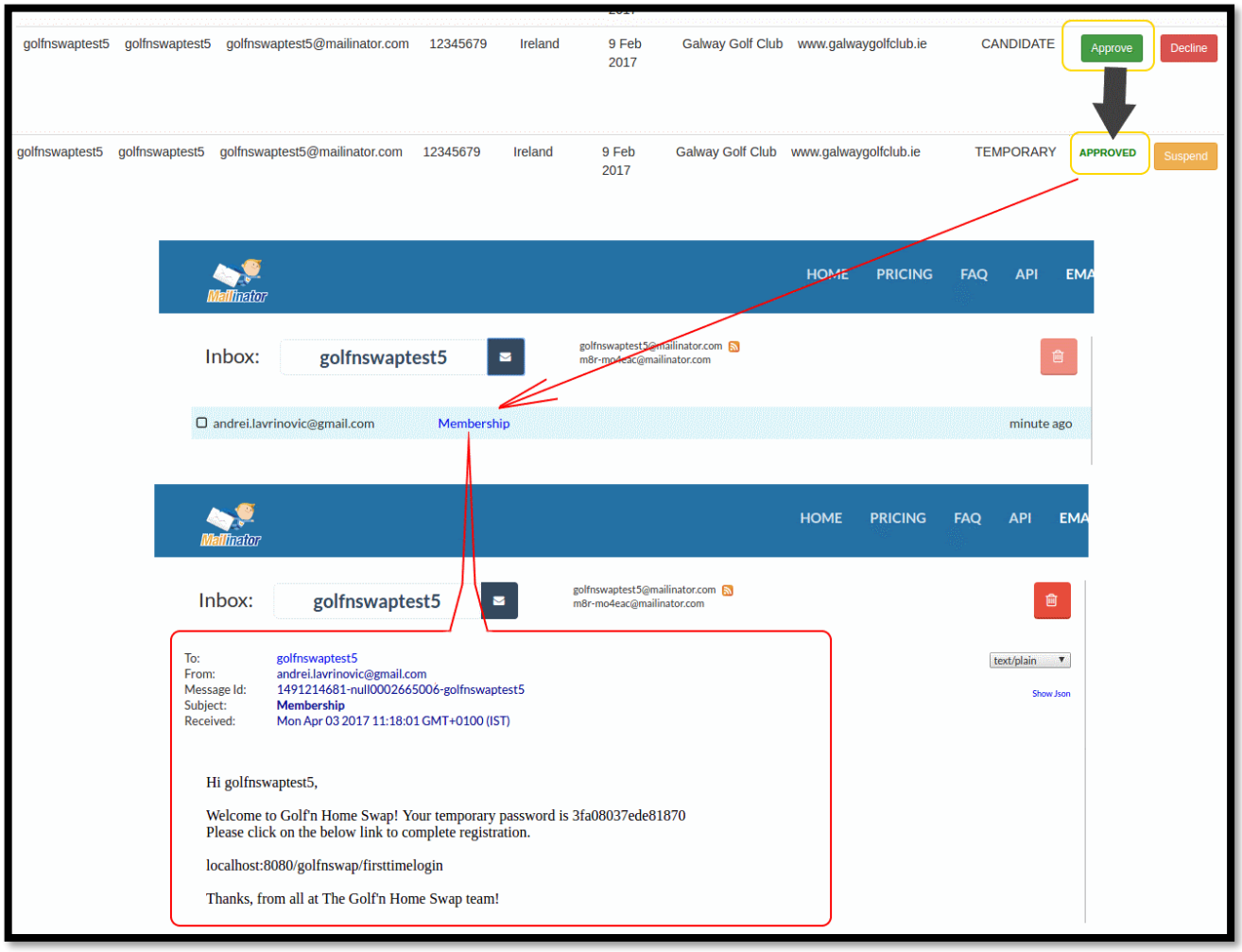


Figure 34: Automated Email Sending

Configuration

The configuration for the Email Management System is implemented in the bean that belongs to the email-context.xml file. The *mailSender* instance in the controller has a reference by id to that bean. The bean contains the properties for *JavaMailSenderImpl* class that is handling the email service at that time.

Figure 35: Java email config bean

Executable mechanism

As mention previously, the main interface for the Java mail system is MailSender. It instantiates within the system controller file and performs the sending of the emails. To send an email, there must be at least three general properties set: message, destination address, and subject. These properties are wrapped into *EmailSender* that implements the *Emailable* interface. Then using the *MailSender*, an email is sent to the user's email address that's declared in the below figure;

Figure 36: Email Sending Execution

Authentication and Authorisation

Security is one of the most significant areas in software development. I.T. companies spend large amounts of money on research and implementation. A priceless aspect of a company is how secure their products and systems are.

Authentication is one of the components of security and one of the outer layers that performs restrictions and disables unwanted users from getting access to the system. Generally, it is the process of determining if someone is who they say they are. The user provides credentials and if they match, the process is completed and the user is granted authorisation and allowed access to the system.

Authorisation is a process of giving permission to the user. Many systems contain such areas that only users with specific privileges are permitted. For example, take a user's Facebook account or any administration page that allows a user to change some system configuration or resources that contain some sensitive data. All these software elements need to be managed by someone, but must not be available for everyone.

Golf'n Swap is an interactive system, where users can share information, but at the same time, some data is private and not for public view. The administration system must be responsible and be managed by users with high privileges only. The user management system is private as well, where users with certain permissions can manage their accounts and of course, the system must restrict non-authorised users from accessing the certain sections, such as other software systems. Golf'n Home Swap assures users that information will be kept in a private manner. To

achieve this goal the authentication and authorisation system was implemented and the Spring framework was efficiently used in this scenario also.

Spring Security

Spring security focuses on providing both authentication and authorisation in an application. At the same time, this framework provides the protection against attacks like session fixation, clickjacking and cross-site request forgery.

Spring Security is an open source platform **[13]** and it is quite simple to write your own authentication mechanism. Irrespective of the authentication mechanism, Spring Security provides a deep set of authorisation capabilities. There are three main areas of interest: authorising web requests, authorising whether methods can be invoked and authorising access to individual domain object instances.

To integrate Spring Security into Golf'n Home Swap, there were dependencies required that were included in the pom.xml file;

Library Name	**Version**	**Description**
Spring-security-core	3.1.4	Contains core authentication and access-control classes and interfaces, remoting support and basic provisioning APIs.
Spring-security-web	3.1.4	Contains filters and related web-security infrastructure code. You'll need it if you require Spring Security web authentication services and URL-based access-control.
Spring-security-config	3.1.4	Contains the security namespace parsing code & Java configuration code. You need it if you are using the Spring Security XML namespace for configuration or Spring Security's Java Configuration support.
Spring-security-taglibs	3.1.4	Spring Security has its own taglib which provides basic support for accessing security information and applying security constraints in JSPs

Figure 37: Spring Security table

Spring Security Configuration

As in previous modules, Spring Security **[7]** needs to be configured in the context. Configuration is implemented in security-context.xml file. The authentication manager (interface *AuthenticationManager*) is performing the authentication by calling the authenticate method that takes user details such as credentials (*getCredentials()*), authorities (*getAuthorities()*), username (*getPrincipal()*) and other information. Based on gathered data, the authentication manager can grant or restrict the access to the system. To get the user's data, the class *Custom UserDetailsService.java* was created. It is handling the retrieval of user information when logging in as part of Spring Security. It is used by authentication manager. *CustomUserDataService* generally has access to the database and retrieving the user's data to pass it to the *org.springframework.security.core.userdetails.User* object. The service itself implements *UserDetailsService* interface that is also part of security-core module and it contains one method – *public UserDetails loadUserByUsername(String username)* that is overriden in *CustomUserDetailsService* in such way that it is requesting the database information and getting user's data. So, the authentication manager is getting the user's data using *CustomUserDataService* and performs the authentication. It

is also responsible of creating the session ID that has a timeout counter for nonactive users. The service class and authentication manager configuration is shown in the figure below;

```
23 @Service
24 public class CustomUserDetailsService implements UserDetailsService {
25
26     private Logger log = Logger.getLogger(CustomUserDetailsService.class);
27
28     @Autowired
29     private UserRepository repo = new UserRepoImpl();
30
31     /**
32      * Method loadUserByUsername locates the user based on the email (username). In the actual implementation, the sea
33      * may possibly be case insensitive, or case insensitive depending on how the implementation
34      * instance is configured. In this case the UserDetails object that comes back may have
35      * a email (username) that is of a different than what was actually requested.
36      */
37     @Override
38     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
39         User user = this.repo.findByEmail(username);
40         log.info("CustomUserDetailsService: User email is " + user.getEmail());
41
42         org.springframework.security.core.userdetails.User userDetails =
43                 new org.springframework.security.core.userdetails.User(user.getEmail(),
44                         user.getPassword(), true, true, true, true,
45                         getAuthorities(user.getUseraccesslevel().toString()));
46         return userDetails;
47     }
```

Figure 38: Custom user details service

```
10     <bean id="customUserDetailsService"
11         class="ie.gmit.sw.repo.CustomUserDetailsService">
12     </bean>
13
14     <security:authentication-manager>
15         <security:authentication-provider
16             user-service-ref="customUserDetailsService">
17         </security:authentication-provider>
18     </security:authentication-manager>
19
20
```

Figure 39: Authentication Manager

The Golf'n Swap system has some pages that permit all users and guests, other pages are permitted just for users with higher priorities, and some pages are available just for registered users with normal privileges. To manage access for a different type of users, the configuration file contains an access manager bean that has a list of all resources with access permission associated with that user's authority value. The user authority in the system is represented by *useraccesslevel* state that is stored in the user database with other user's details. There are six user access level states:

- ADMIN
- REGISTERED
- CANDIDATE
- DECLINED
- SUSPENDED
- TEMPORARY

User's with admin state have access to the administration page and can change the states of other users. Registered users can manage their accounts on the account management pages. Candidates have access to common pages that could be visited by anyone including guests. Same for the users with other states – declined, suspended and temporary. Here is an example of the Spring Security access manager bean;

```
20
21      <!-- Access manager -->
22      <security:http use-expressions="true">
23
24          <security:intercept-url pattern="/admin" access="hasAnyRole('ADMIN')" />
25          <security:intercept-url pattern="/approve" access="hasAnyRole('ADMIN')" />
26          <security:intercept-url pattern="/decline" access="hasAnyRole('ADMIN')" />
27          <security:intercept-url pattern="/suspend" access="hasAnyRole('ADMIN')" />
28          <security:intercept-url pattern="/reactivate" access="hasAnyRole('ADMIN')" />
29
30          <!-- FOR TESTING, will need to be changed to 'REGISTERED' only when finished -->
31          <security:intercept-url pattern="/addhomewizard" access="hasAnyRole('ADMIN', 'REGISTERED')" />
32          <security:intercept-url pattern="/addhome" access="hasAnyRole('ADMIN', 'REGISTERED')" />
33          <security:intercept-url pattern="/userhome" access="permitAll" />
34          <security:intercept-url pattern="/docreatehome" access="hasAnyRole('ADMIN', 'REGISTERED')" />
35          <security:intercept-url pattern="/usershomes" access="hasAnyRole('ADMIN', 'REGISTERED')" />
36
37          <!-- ********************************************************************** -->
38
39          <!-- Some kind of security check neede here to see if user is Registered? -->
40          <security:intercept-url pattern="/dashboard" access="hasAnyRole('ADMIN', 'REGISTERED')" />
41
42          <security:intercept-url pattern="/firsttimelogin" access="permitAll" />
43          <security:intercept-url pattern="/successapply" access="permitAll" />
44          <security:intercept-url pattern="/denied" access="permitAll" />
45          <security:intercept-url pattern="/docreate" access="permitAll" />
46          <security:intercept-url pattern="/whatitis" access="permitAll" />
47          <security:intercept-url pattern="/howitworks" access="permitAll" />
48          <security:intercept-url pattern="/price" access="permitAll" />
49          <security:intercept-url pattern="/faq" access="permitAll" />
50          <security:intercept-url pattern="/apply" access="permitAll" />
51          <security:intercept-url pattern="/" access="permitAll" />
52          <security:intercept-url pattern="/login" access="permitAll" />
Source | Namespaces | Overview | beans | context | sec
```

Figure 40: Access Manager bean

Now when Spring Security is integrated and configured the Golf'n Swap system is much more secure. The way of implementation allows for more restrictions to be added and include more access levels very easily by adding them into the *UserAccessLevel* enum, which is loosely coupled and independent.

System components

The Golf'n Home Swap application enables interactive communication between its users and so must handle user's requests and responses respectively with associated information. To achieve these goals, A centralised structure was built, which is a secure mechanism that could deal with information transportation in an efficient manner.

Libraries Used

As mentioned earlier, Spring MVC is a framework that provides Model-View-Controller structure and such design patterns can responsibly and effectively maintain the system behind the scene. In addition, the Spring framework contains a suite of libraries that implement many features that make this application very comfortable to use and is flexible and secure.

This is a list of libraries that is used in the Golf'n Swap application;

Library Name	**Version**	**Description**
Spring-beans	4.3.3	Provides an advanced configuration capable of managing of any type of object
Spring-context	4.3.3	Supper set of BeanFactory and used as a container.
Spring-core	4.3.3	Providing the fundamental parts of the framework including dependency injection features.
Spring-web	4.3.3	Provides basic web-oriented integration features.
Spring-webmvc	4.3.3	Also known as the Web-Servlet module. Contains Spring's model-view-controller (MVC) and REST Web Services implementation for web applications.
Jstl	1.2	Java Server Tag Library encapsulates core functionality common to many JSP applications.
Bootstrap	3.1.1	This class loader contains the basic runtime classes provided by the Java Virtual Machine
Mongo-java-driver	2.11.0	The official MongoDB Java Driver providing both synchronous and asynchronous interaction with MongoDB. Powering the drivers is a new driver core and BSON library.
Spring-data-mongodb	1.7.2	provides integration with the MongoDB document database. Key functional areas of Spring Data MongoDB are a POJO centric model for interacting with a MongoDB DBCollection and easily writing a Repository style data access layer.
Cglib	1.2	is a powerful, high performance and quality Code Generation Library, It is used to extend JAVA classes and implements interfaces at runtime.
Slf4j-api	1.7.5	provides a Java logging API by means of a simple facade pattern
Slf4j-log4j	1.7.5	It helps prevent projects from being dependent on lots of logging APIs just because they use libraries that are dependent on them.
Spring-security-core	3.1.4	Contains core authentication and access-contol classes and interfaces, remoting support and basic provisioning APIs.
Spring-security-web	3.1.4	Contains filters and related web-security infrastructure code. You'll need it if you require Spring Security web authentication services and URL-based access-control
Spring-security-config	3.1.4	Contains the security namespace parsing code & Java configuration code. You need it if you are using the Spring Security XML namespace for configuration or Spring Security's Java Configuration support.

Spring-security-taglibs	3.1.4	Spring Security has its own taglib which provides basic support for accessing security information and applying security constraints in JSPs
Spring-context-support	4.3.5	Classes supporting the org.springframework.context package, such as abstract base classes for *ApplicationContext* implementations and a *MessageSource* implementation
Javax-mail-api	1.5.6	Set of abstract APIs that model a mail system
Mail	1.4.7	The JavaMail reference implementation jar file, including the SMTP, IMAP, and POP3 protocol providers

Figure 41: Table of Libraries used

By using general Spring libraries, the team could build a flexible system with separate modules each of which performs its own function. Get and Post requests are handled by the *"Dispatcher servlet"* that is using "*Handler mapping*" to pick the associated "*Controller*". Controller, which in turn, returns the name of the "*view*" resources with associated bound data. The "*Dispatcher Servlet*" with accompanied "*View Resolver*", looks for the appropriate view and supplies it with the associated data. In the end, the user gets what they initially requested.

Server

The Golf'n Home Swap system is based on a client/server model as is the case with most EE systems, a server program waits and fulfills requests from client programs which are running on other computing devices such as user's desktops, laptops, tablets or mobiles. When data needs to be retrieved from the database, the web server application is playing the role of the client and sends requests to other services such as MongoDB.

Considering the specifications for the system, the "Apache Tomcat" web server was chosen as the runtime environment. The general aspects associated with Java EE and which are implemented in Tomcat are Java Servlet, Java Server Pages (JSP), Java Expression Language (Java EL) and Web Sockets. This is also providing the java HTTP Web Server environment in which java code can run. At the same time, Tomcat is a free and open source with very good and detailed documentation.

Tomcat is using parallelism during request handling. Each incoming request requires a thread for the duration of that request. If more simultaneous requests are received than can be handled by the currently available request processing threads, additional threads will be created up to the configured maximum (the value of the maxThreads attribute). If still more simultaneous requests are received, they are stacked up inside the server socket created by the Connector, up to the configured maximum (the value of the *acceptCount* attribute). Any further simultaneous requests will receive "connection refused" errors until resources are available to process them.

Tomcat connections for this system have been established on Amazon AWS cloud service. It runs on the RedHat platform. The most common Tomcat port that is used to get access to the system is:8080. The server IP address is 52.214.71.85. So to get access to the Golf'n Home Swap application the ip:port/golfnswap need to be typed into the browser (http://52.214.71.85:8080/golfnswap/)

The version of Tomcat is 9.0.0.M15

Database

The system is using MongoDB to store all data that's been provided by users. Mongo is not a relational database and is related to the group of NoSQL databases. It is applying the document-based storage in the collections system. The document is represented by BSON computer data format which is very similar with JSON.

The general advantage of MongoDB is its scalability. The document-based records can be modified and updated very easy. If the Golf'n Home Swap system needs to be altered, then some more user's details will be required or data format will be changed in some way, so with changing data entity (POJO) the mongo database will adapt automatically. There is no relational structure or dependency between rows or tables, which makes the storage system very flexible. Another reason why MongoDB was picked for this system is that its open source and free to use.

The Golf'n Home Swap service has been deployed and is hosted on Amazon Web Services (AWS) along with MongoDB. The version used database is 3.0.14. It also runs on the RedHat platform.

Maven

As it was mentioned above the system has many libraries that the application is depending on. A lot of packages and many classes need some way to be organised. The Apache Maven is a project manager and comprehension tool that is used by this system to organise and manage the project. Maven can perform a lot of tasks and was highly used during the system building. The tasks that maven can perform are;

- **validate**: Validate that the project is correct and all necessary information is available.
- **compile**: Compile the source code of the project.
- **test**: Test the compiled source code using a suitable unit testing framework. These tests should not require the code to be packaged or deployed.
- **package**: Takes the compiled code and packages it in its distributable format, such as a JAR.
- **integration-test**: Processes and deploys the package, if necessary into an environment where integration tests can be run.
- **verify**: Run any checks to verify the package is valid and meets quality criteria.
- **install**: Install the package into the local repository, for use as a dependency in other projects locally.
- **deploy**: Done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

There are two other Maven lifecycles of note beyond the default list above and they are;

- **clean**: Cleans up artifacts created by prior builds.
- **site**: Generates site documentation for this project.

The Golf'n Home Swap applications were deployed on Amazon AWS using maven.

Spring

The form and goals of "Golf'n Swap" system, assumes that it will be built with Java 2 Platform Enterprise Edition. The J2EE platform consists of a set of services, API's and protocols that provide the functionality for deploying

multitiered, Web-based application. The Spring is a J2EE implementation. Here are some of the reasons Spring was used to build this application;

- Spring can effectively organize your middle tier objects. Spring takes care of plumbing that would be left up to you if you use only Structs or other frameworks geared to particular J2EE APIs. And while it is perhaps most valuable in the middle tier, Spring's configuration management services can be used in any architectural layer, in whatever runtime environment.

- Spring is designed so that applications built with it depend on as few of its API's as possible. Most business objects in Spring applications have no dependency on Spring.
- Spring is open source and free.

The spring modules used in the Golf'n Home Swap applications are;

- Spring Core
- Spring Web
- Spring Security
- Spring MongoDB

Object Oriented (OO) Model

The system development approach for the Golf'n Home Swap application in terms of how the software system's object mode was going to be engineered was by using an Object-Oriented Model. The requirements were organised around objects which incorporates both data and functions. The Plain Old Java Object (POJO) such as *User.java* entity can be taken as an example, where the data is represented by object properties and functions are representing with getters and setter's methods;

```
@Document(collection="user")
public class User {

    @Id
    private String id; // Delegate id creation to Mongodb, like we saw earlier today (11th Oct)
    private String firstname;
    private String surname;
    private String email;
    private String golfregnum;
    private String countryissued;
    private Date dateofissue;
    private String homeclubname;
    private String homecluburl;
    private String password;
    // private String permanentPassword;


    /* Created an ENUM class to hold the values for [admin, registered, candidate, declined, suspended, temporary]
       Also set default value to be CANDIDATE until approved */
    private UserAccessLevel useraccesslevel = UserAccessLevel.CANDIDATE;


    // Constructors
    /**
     * Default constructor used for creation User entity
     */
    public User() {}


    // Getters & Setters
    public String getId() { return id; }

    public void setId(String id) { this.id = id; }

    public String getFirstname() { return firstname; }

    public void setFirstname(String firstname) { this.firstname = firstname; }

    public String getSurname() { return surname; }

    public void setSurname(String surname) { this.surname = surname; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getGolfregnum() { return golfregnum; }

    public void setGolfregnum(String golfregnum) { this.golfregnum = golfregnum; }
```

Figure 42: User object (Entity)

The User object in turn is representing the data document that will eventually be sent to the mongo database. When implementing the Object-Oriented approach, the first step after determining the requirements is to define the objects. In the Golf'n Home Swap system which is basically a user-based system, the *User* object is the one of the most significant objects. As mentioned previously, this object is representing the database document, so this object must be used to perform the basic database CRUD operations such as: write, read, update and delete. These commands are implemented in another object *UserRepoImpl*, which implements the *UserRepository* interface. The *UserRepository* in turn implements the Repository interface that is part of the Spring-data-mongodb library. At the same time the *CustomUserDetailsService* object implements the *UserDetailsService*, which are Spring-security components and are fully composed of the *UserRepoImpl* to get an access to the User.

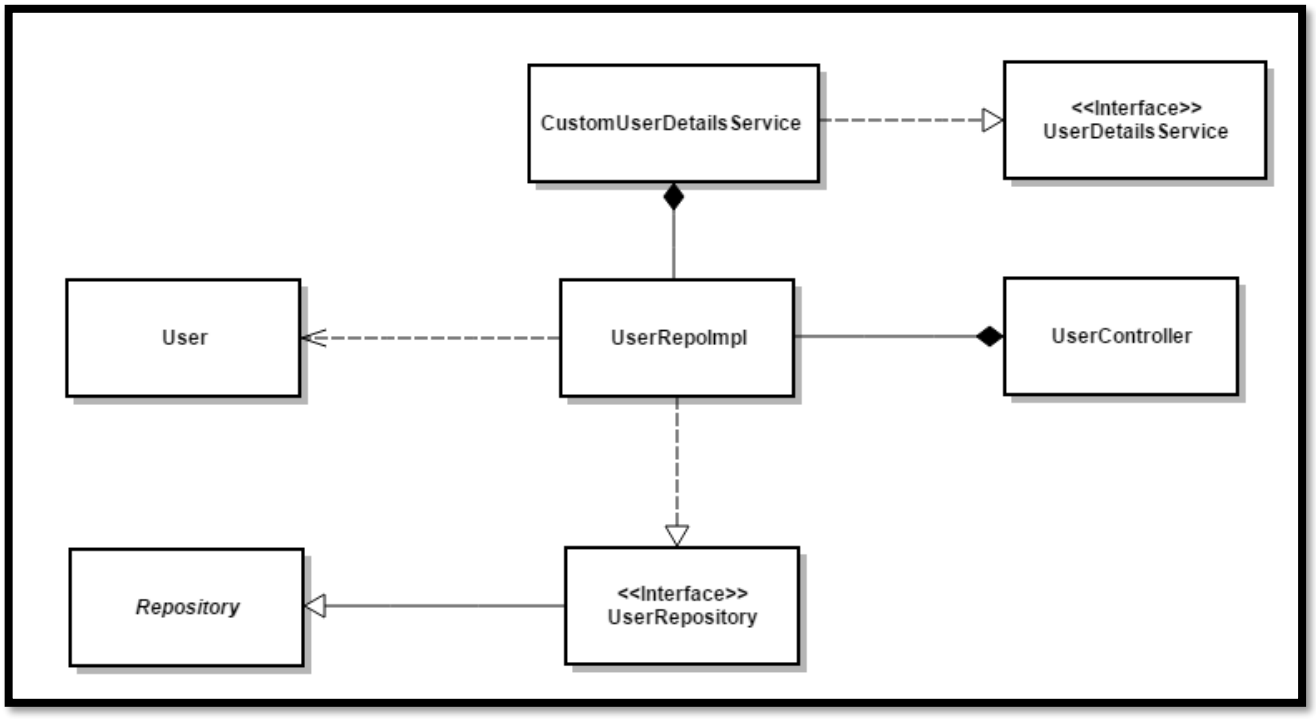


Figure 43: UML diagram for user dependencies

By using inheritance and composition the system acquires flexibility and loose coupling. Another advantage of using the Object-Oriented approach is technology-independence. For example, if the system would be modified and different database are integrated into it, then the User entity would remain the same as a representation of the user's data. It is available because the concept of object oriented programming is based upon objects that aims to incorporate the advantages of modularity and reusability.

Encapsulation is another very strong feature of the object-oriented model. Manipulation of an object's variables by other objects or classes is discouraged to ensure data encapsulation. A class should provide methods through which other objects could access variables. In the Golf'n Home Swap system, data requests are implemented in *UserRepositoryImpl* object and used in *UserController* just by calling the necessary method. Basically, the controller doesn't care how to store new user in the database or update existing one with new data. It just uses the encapsulated *UserRepositoryImpl* requests.

Design Patterns

About Design Patterns

Design Patterns are general reusable solutions to commonly occurring problems **[12]**. Patterns are not complete code, but it can be uses as a template which can be applied to a problem. Patterns are re-usable and they can be applied to similar kinds of design problems regardless of the domain. A pattern used in one practical context can be re-usable in other contexts also. Here are some of the reasons to use design patterns;

1. **Flexibility**: Using design patterns makes code flexible. It helps to provide the correct level of abstraction due to which objects become loosely coupled to each other which makes the code easy to change.

2. **Reusability:** Loosely coupled and cohesive objects and classes can make the code more reusable. This kind of code becomes easy to be tested as compared to the highly-coupled code.

3. **Shared Vocabulary:** Shared vocabulary makes it easy to share the code with other team members. It creates more of an understanding between the team members in relation to the code.

4. **Capture best practices:** Design patterns capture solutions which have been successfully applied to problems. By learning these patterns and the related problems, an inexperienced developer learns a lot about software design.

MVC

Initially when choosing the Spring MVC platform for the enterprise edition (EE) system development it became evidently that the Model View Control (MVC) design pattern would be implemented into the system. The Golf'n Home Swap application is divided into three general sectors which are loosely coupled and interactive.

These are;

- Model - that represents data.
- View - that represents the visualization (UI).
- Controllers – that represent the performance mechanism, that controls the data flow from the model object and updates the view whenever the data changes. It keeps the view and model separate.

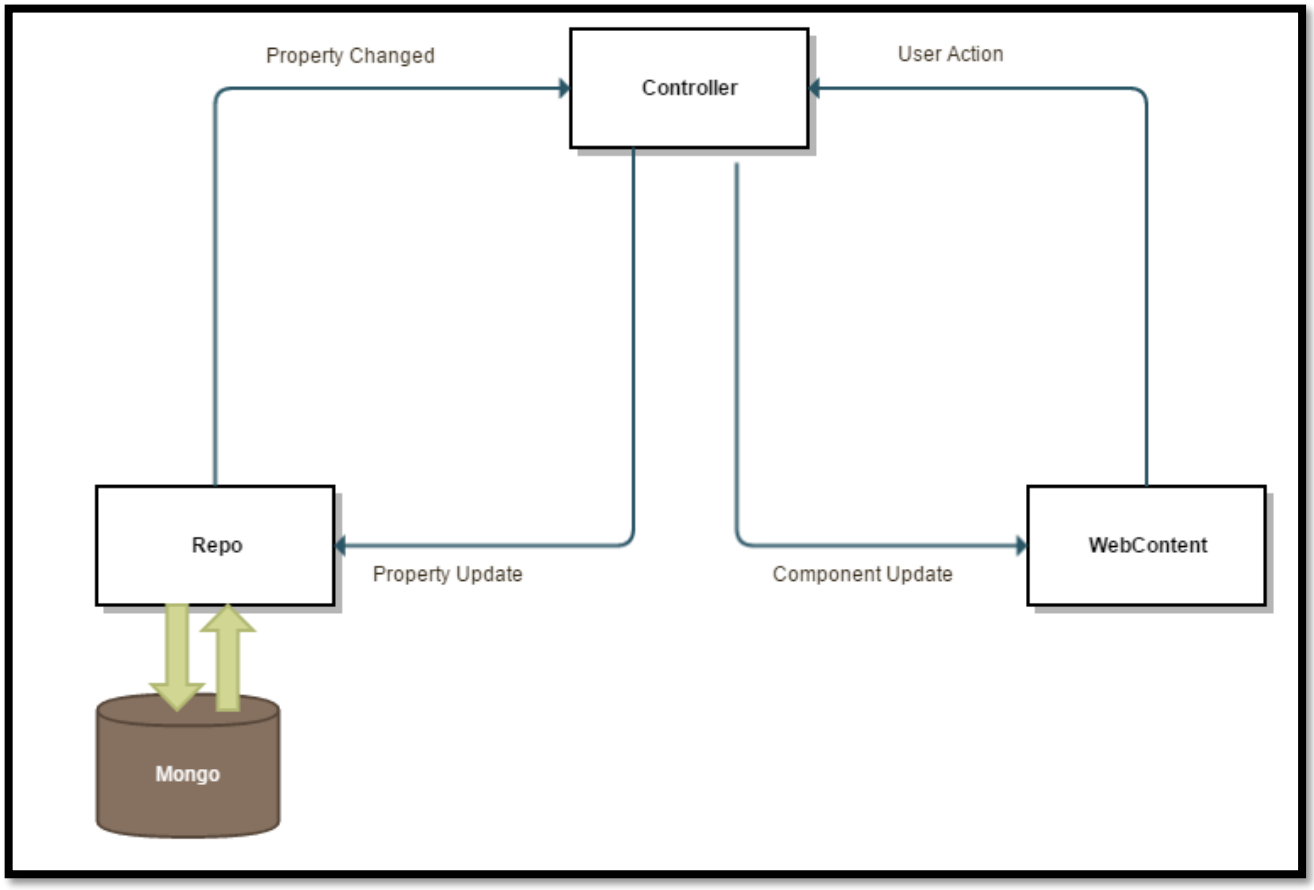


Figure 44: Golf'n Swap MVC design pattern.

In the figure above, the view is represented by *WebContent*. This module contains the folder with jsp (Java Server Page) files and resources for JavaScript, CSS files, and other resources such as images and videos. The JSP's basically are view templates. Every time a user submits, the request goes to the dispatcher servlet which is configured in the web.xml file. *DispatcherServlet.java* is one of the general services in the Spring MVC system. It performs the front controller function and is part of the spring-web component. When the dispatcher servlet gets the request, it delegates it to the controller.

Controller block contain *UserContoller* and *HomeController* classes. These objects have several methods that return the string value which matches the name of the jsp page. These methods are handling requests from clients. Controllers interpret each user input and transform it into a model that is presented to the user by the view. In the Golf'n Home Swap system an annotation driven model is widely used. All controllers are equipped with *@RequestMapping* annotations for mapping web requests. This technique is providing a semantic adaptation of the concrete environment.

The model is representing the data. In other words, the model is pretending to be the data itself. In the schema above, the Repo block is the model component. It contains the entity files with a representation of the mongo database documents, type converters, enums and repositories. Repositories are essentially API's between the physical database and data representation. Using these, the system gets the data for modelling and formats the response accordingly.

When the controller gets the model, the next step is to delegate the rendering of the response. The Dispatcher servlet is supplying the response using the view template which is the JSP. Here is the schema of how the Spring MVC model is processed;

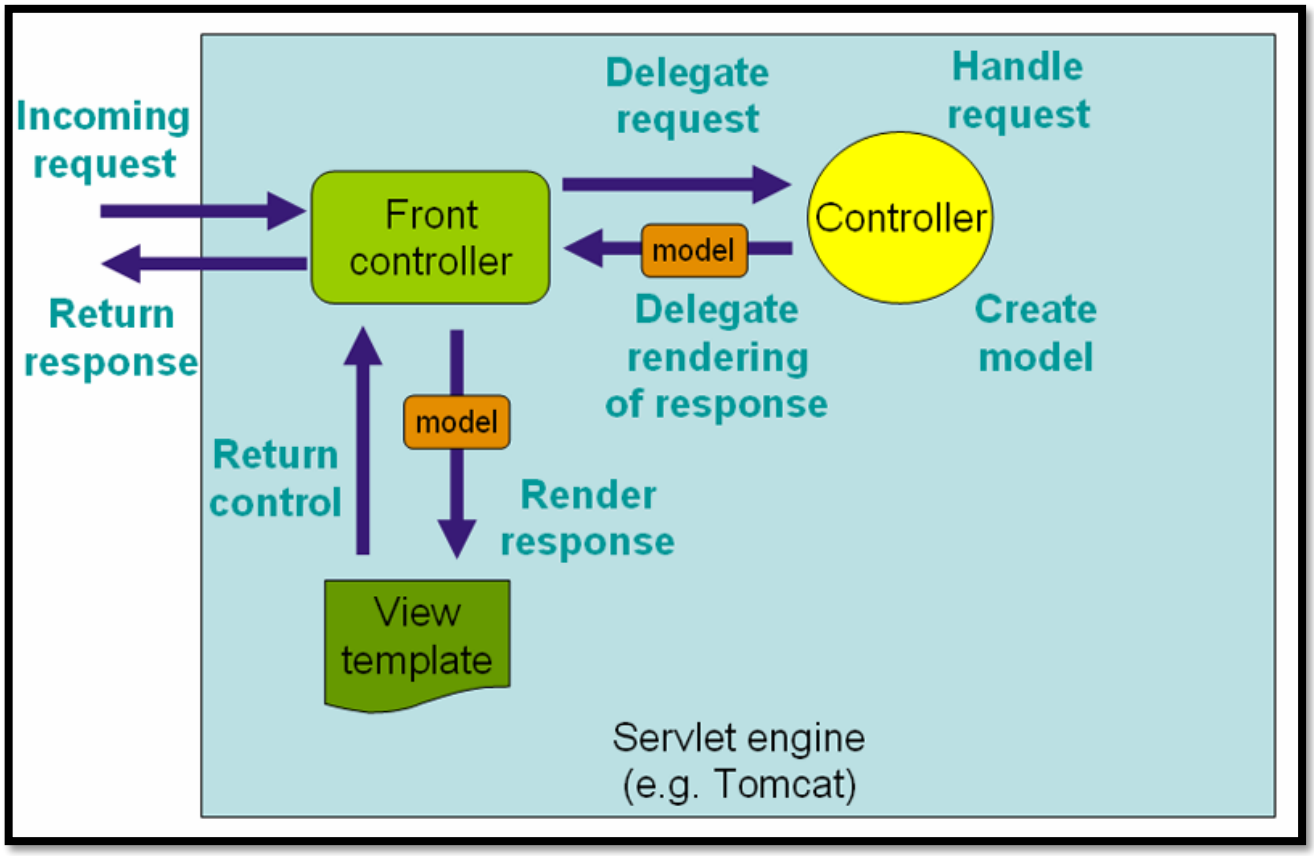


Figure 45: The request processing workflow in Spring Web MVC (high level)

System Deployment

When the system is built, and tested it is time to make the system available for users. Deployment implementation takes several steps:

1. Choose platform provider (Amazon).
2. Configure and launch the environment.
3. Install all necessary components.
4. Deploy the app.
5. Make a connection to a database.

Platform provider

To run the application online it must to be deployed into the system which supports several components that are necessary for app maintenance. These main components are;

- Java 8
- Tomcat server
- Mongo database

A lot of hosting providers support different implementation of databases and Tomcat servers but not all of these have Java. Another problem is that to deploy a java app, specific configuration of the system files is needed, that is not always permitted and restricted by hosting providers because they maintain thousands of web sites which need to keep certain standards that are distributed among all users.

There are couple of app platforms that supply the service which supports the java application deployment. These are 'Google Cloud Platform', 'Heroku Dev Center', 'CloudBees', 'CumuLogic', 'CloudFoundry', 'Dotcloud', 'Jelastic' etc. These and other services are grouped in one term: 'Paas' – Platform as a Service. Most of Paas providers are charging a fee for using their service and it very expensive. Also, most of those contain tools that manage your app maintenance. Which can be good and bad at the same time.

- Good because deployment process, app management and maintenance are very simple and in most cases automated.
- Bad because application developers are stuck with some standards which are necessary to maintain a lot of different systems.

Cloud services such as 'Amazon aws', 'Microsoft Azure', 'Oracle WebLogic', provide developers with a space in the cloud with many tools and services available that allow each developer to set up their own preferred environment. The main advantages of using cloud platforms is flexibility, adaptability, variety. Disadvantage is complexity, which to contradict the latter comment can be an advantage if a developer likes to delve into configurations and settings. Among all platforms, Amazon AWS can be chosen for two main reasons:

1. "Amazon AWS" allows a developer to assign and use a free tier package that includes almost all available services and tools but is limited to one year. Even so, it can run an application non-stop 24/7 for an entire year.
2. Amazon servers are based also in Ireland, which allows for faster access.

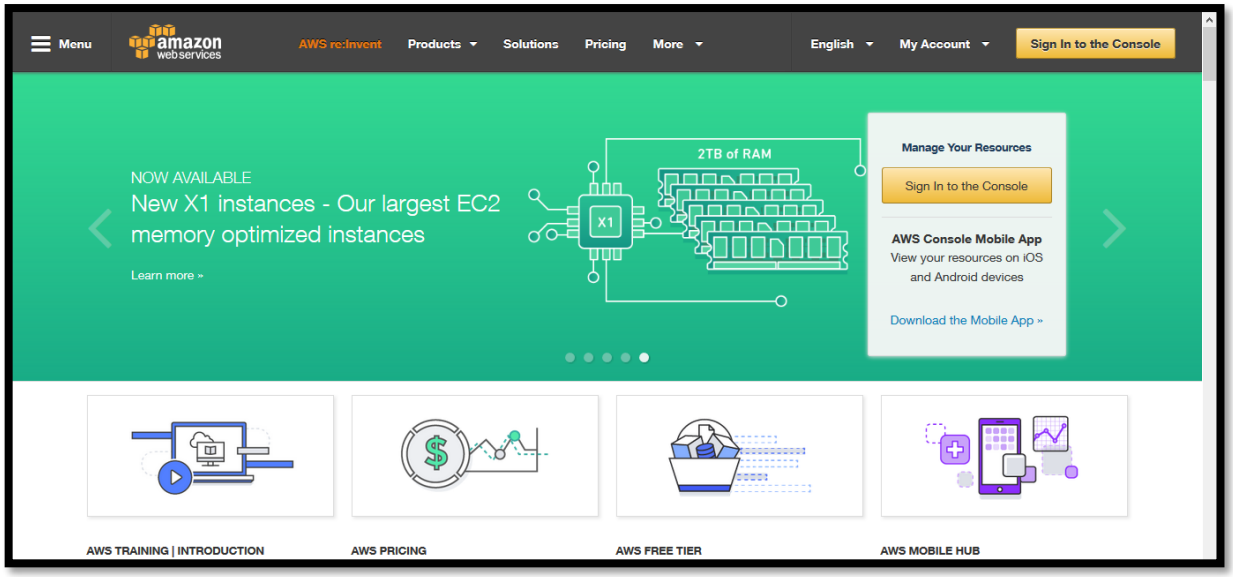


Figure 46: Snapshot of Amazon AWS home page: https://aws.amazon.com/

Launching the serveries in the cloud.

After the Amazon AWS account is created, the services required can be picked. To run the service, the instances must first be created. These instances allow users to configure and use the cloud as they see fit.

For example, the EC2 service uses as instance which provides computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates the need to invest in hardware up front, so a developer can create applications faster.

EC2

Configuring and launching the EC2 instance requires the following steps and instructions;

1. Choose the AMI (Amazon Machine Image), this is the operating system.
2. Choose the Type of Instance.
3. Configure the Instance.
4. Add Storage.
5. Tag the Instance.
6. Configure the Secure group.
7. Review.
8. Launch.

Amazon provides a number of AMI's included in the free tier package such as Amazon Linux, Red Hat, SUSE Linux, Ubuntu Linux and Windows Server 2012. Red Hat is a commonly used operation system that can maintain java enterprise applications.

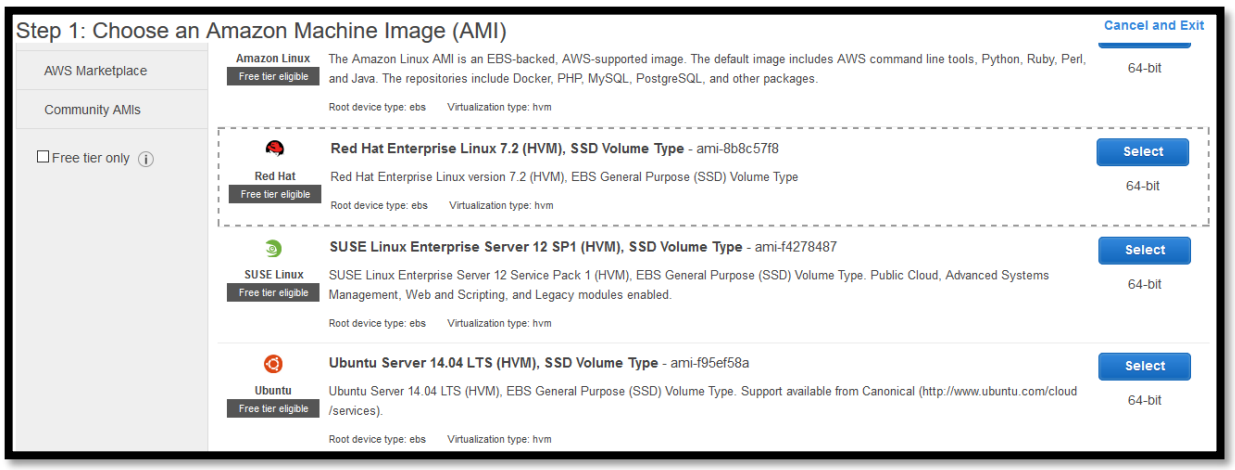


Figure 47: Some AMI's provided by Amazon AWS

After an instance is configured and launched, the next step is to connect to the cloud instance. A Key value pair needs to be generated and stored in the machine. This is used to connect to the cloud. If using command line, the connection can by established with the ssh utility.

Amazon has some restrictions that must be configured during instance setup, these are called Security Group Rules. At this point, all that needs to be done is to add the locations (IP Addresses) where the cloud can be connected from. There is way to set all available connections, just set ip address as 0.0.0.0, which allows full access for everyone.

Components installation

When an EC2 instance using a RedHat container is launched and connection to the service is established, then it is time to load it with the relevant environment components. Initially the RedHat platform has only its core and tools with it. Some useful utilities can be download first to make the environment more comfortable. Those are;

- Wget or curl – for downloading resources
- Nano or vim – text editors for terminal
- Yum – could be used for downloading and installation.

Next step for deployment is to determine the environment components. To run the Golf'n Swap system, there are certain programs that must be setup;

- Java 8 (Java run environment)
- Tomcat web server
- Maven – project manager

To check if all the components installed successfully and everything works, the Tomcat web-server can be launched using the Amazon AWS public IP address, the Tomcat's home page can be opened on the browser. The security group configuration must also be edited by opening the :8080 port.

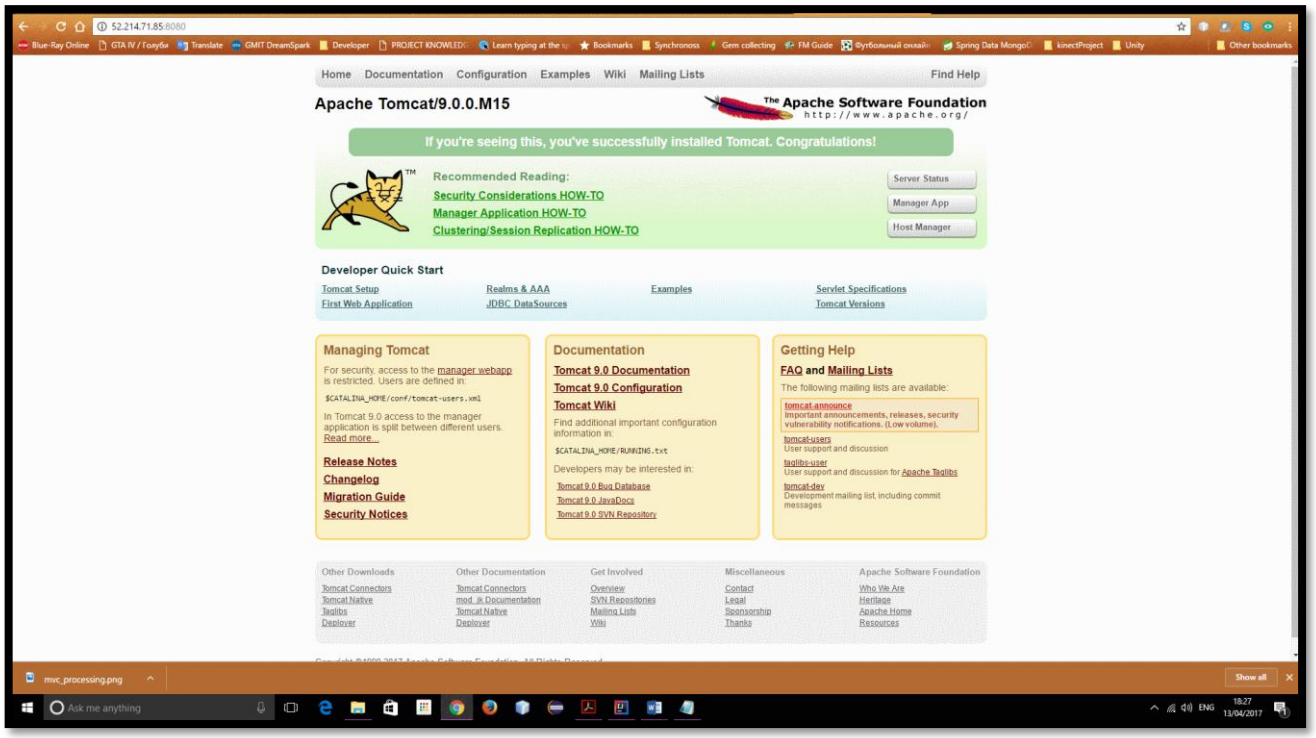


Figure 48: Tomcat Home page

Deployment

When the remote environment has been established and configured, then it's time to deploy the application. Maven is one of the tools that is managing the Golf'n Home Swap application and it is also used to deploy the system to the remote server. It must have access to the Tomcat manager tool so there are some configurations that must be performed. Two tomcat's files, *tomcat-users.xml* and *manager.xml* must be modified by adding user credentials and roles to adjust the access partition.

- In **…/tomcat-users.xml** adds users with their roles:

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="[redacted]" password="[redacted]" roles="manager-gui, manager-script" />
```

- In …/**manager.xml** must configure access for remote or local users:

```
[root@ip-172-31-47-228 conf]# cat ./Catalina/localhost/manager.xml
<Context privileged="true" antiResourceLocking="false"
doBase="${catalina.home}/webapps/manager">
<!--
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
  allow="[redacted]" />
-->
</Context>
```

Now maven can access Tomcat Manager from the local machine. To do this maven must know the credentials, so there is some configuration that needs to be done in the *settings.xml* file;

```
133     <server>
134       <id>TomcatServer</id>
135       <username>[redacted]</username>
136       <password>[redacted]</password>
137     </server>
138   </servers>
139
```

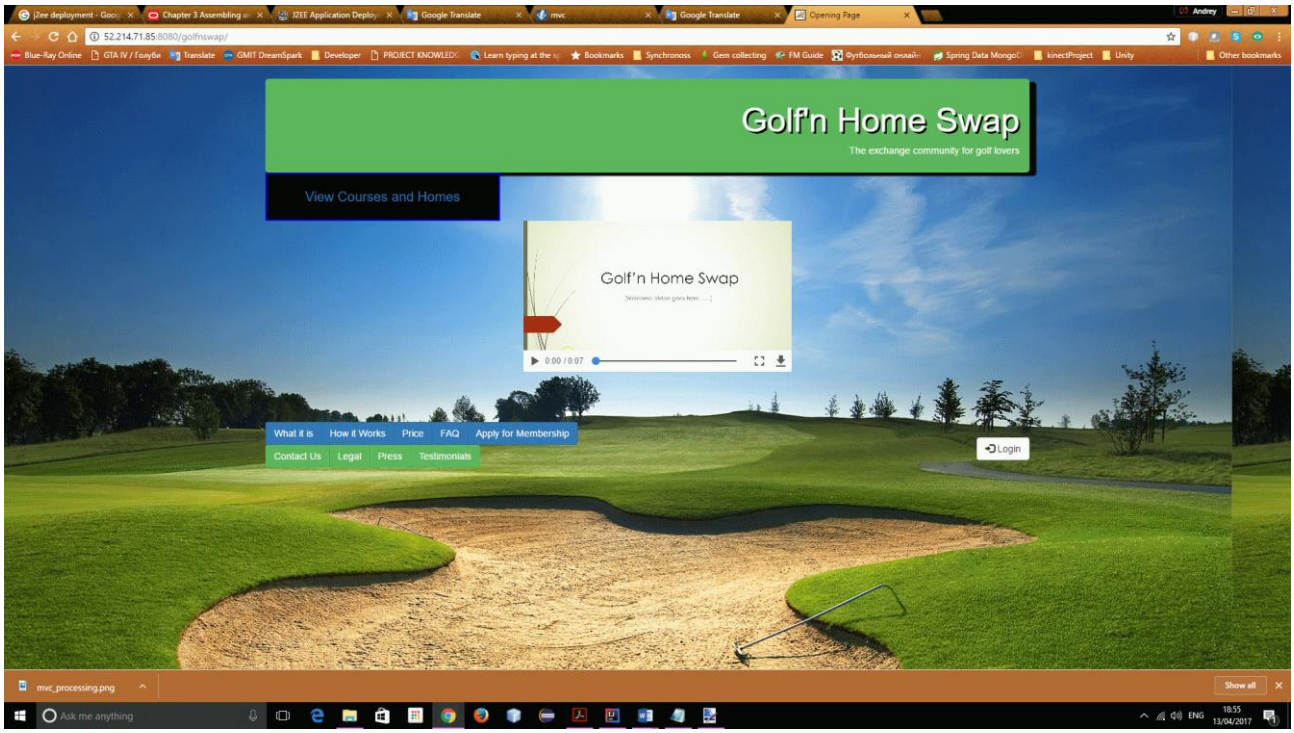
The final step is to include the apache tomcat maven plugin into the application pom.xml file. It's important to know that the server value in the pom files plugin must match the id value in maven's *settings.xml*. In our case this value is **TomcatServer:**

```
29
30          <!-- AWS Deployment -->
31          <plugin>
32              <groupId>org.apache.tomcat.maven</groupId>
33              <artifactId>tomcat7-maven-plugin</artifactId>
34              <version>2.2</version>
35              <configuration>
36                  <url>http://52.50.110.186:8080/manager/text</url>
37                  <server>TomcatServer</server>
38                  <path>/golfnswap</path>
39              </configuration>
40          </plugin>
41
```

The Application is ready to be deployed. Execution of the command ***mvn tomcat7:deploy*** builds the project, generates the war file and integrates it into tomcat:

```
[INFO] tomcatManager status code:200, ReasonPhrase:
[INFO] OK - Deployed application at context path /altech-mgmt-system
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 33.340 s
[INFO] Finished at: 2016-07-19T09:37:41+01:00
[INFO] Final Memory: 13M/71M
[INFO] ------------------------------------------------------------------------
```

After the “BUILD SUCCESS” notification is displayed, the application can be considered deployed. The access to the system from the client can be performed by entering the ip address and system name into the browser’s address bar;



CHAPTER 5: SYSTEM EVALUATION

Robustness & Efficiency

To prove that this system has been built on a solid foundation, various components of the application need to be analysed, some of which are modular sections and some as a large piece of a cohesive unit. One of the key things that was needed, was to accomplish allowing multiple concurrent users to access the system. What this means in terms of this application, is that any number of users will be able to perform tasks such as applying for membership, logging in, connecting with other users or adding a home and golf club that they wish to swap, all without crashing the system.

Tomcat

Not all systems require concurrent operations to be enabled in order to function, however anything relating to the web would be considered a prerequisite. The Tomcat webserver offers concurrency to solve these issues, which allows for each user to be given a separate thread. This allows automatic scalability of the application. To prove this, an article on Stackoverflow **[5]** was researched and referenced, that suggested what the best approach was regarding concurrency. As suggested, the *maxThreads* remained untouched inside the *server.xml* file as default, which allows 200 concurrent users simultaneously, but could manually specify a larger amount, which may have a knock-on effect with regard to processing power and as the system may need to be migrated at a later stage it was thought that the options were to keep default values where possible. There were a few reasons why Tomcat was chosen, firstly it's something that's been used previously with other projects with a large degree of success, secondly it's a tried and tested piece of software that's robust. Here's an illustration, referenced from this article **[6]**, that solidified the decision;

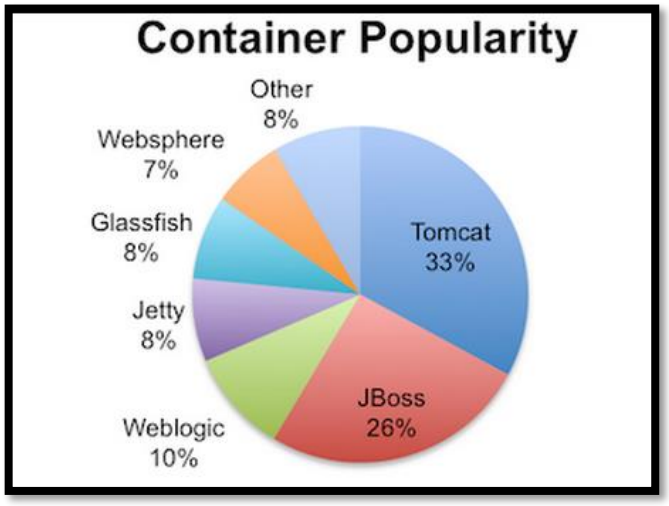


Figure 49: Tomcat Comparison

Just as a side note based on some of the research done, this article **[6]** indicates that Tomcat may be able to hold up to 16,000 concurrent users, which would surpass its rivals, Jetty and Glassfish by a large degree.

MongoDB

The Golf'n Home Swap system incorporates a multi-faceted approach in terms of tiers or levels within the application. As discussed in the Implementation chapter in this report, at an abstract level, there is a Front-end layer, an interface that the user interacts with. There is also a Model or data layer, where the information about each user resides and finally a controller layer, that deals with parsing through information from both sides and communicating requests and responses accordingly. Like Tomcat, MongoDB also requires a concurrency system within the data layer. With mongodb, the *mongo-context.xml* and *mdb.properties* files were created, that allowed for a robust, concurrent data layer that would complement the concurrency level associated with the Tomcat Server.

Here's the code from the mongo-context.xml file;

```
<mongo:mongo host="${mdb.host}" port="${mdb.port}">
        <mongo:options
        connections-per-host="${mdb.connectionsPerHost}"
        threads-allowed-to-block-for-connection-multiplier="${mdb.threadsAllowedToBlockForConnectionMultiplier}"
        connect-timeout="${mdb.connectTimeout}"
        max-wait-time="${mdb.maxWaitTime}"
        auto-connect-retry="${mdb.autoConnectRetry}"
        socket-keep-alive="${mdb.socketKeepAlive}"
        socket-timeout="${mdb.socketTimeout}"
        slave-ok="${mdb.slaveOk}"
        write-number="1"
        write-timeout="0"
        write-fsync="${mdb.slaveOk}"/>
</mongo:mongo>
```

Figure 50: mongo-context.xml

Mongodb properties (mdb.properties file);

```
mdb.host=127.0.0.1
mdb.port=27017
mdb.connectionsPerHost=8
mdb.threadsAllowedToBlockForConnectionMultiplier=4
mdb.connectTimeout=1000
mdb.maxWaitTime=1500
mdb.autoConnectRetry=true
mdb.socketKeepAlive=true
mdb.socketTimeout=1500
mdb.slaveOk=true
mdb.writeNumber=1
mdb.writeTimeout=0
mdb.writeFSync=true
mdb.dbname=golfnswap
```

Figure 51: mdb.properties

Space / Time Complexity

When the first look at system analysis happened during this projects inception, it was discovered that there wouldn't be any major issues with Space / Time complexity. The Development team were boldly able to ascertain at the outset, that all implemented algorithms would execute in polynomial time. To put this into perspective, let's say that an admin person wants to list all the users of the Golf'N Home Swap application, be they Registered or pending. The below code is taken from *UserRepoImpl.java*;

```
public List<User> findAllUsers() {
    List<User> users = mongoTemplate.findAll(User.class);
    log.info("UserRepoImpl--users.size is: " + users.size());
    return users;
}
```

Figure 52: User retrieval

Here's how the information is displayed by the Java Server Page (JSP) view;

```
<c:forEach var="candidate" items="${users}" >
        <tr>
           <td><c:out value="${candidate.firstname}" /></td>
           <td><c:out value="${candidate.surname}" /></td>
           <td><c:out value="${candidate.email}" /></td>
           <td><c:out value="${candidate.golfregnum}" /></td>
           <td><c:out value="${candidate.countryissued}" /></td>
           <td><c:out value="${candidate.dateofissue}" /></td>
           <td><c:out value="${candidate.homeclubname}" /></td>
           <td><c:out value="${candidate.homecluburl}" /></td>
           <td><c:out value="${candidate.useraccesslevel}" /></td>
        </tr>
</c:forEach>
```

Figure 53: JSTL user display all users

This algorithm will execute for the length of the List of users. In terms of Space and time complexity, the Big O notations would be O(n). For example, if the list were 20 in size, then the big O notation would be O(20). This confirms that the algorithm operates in a linear fashion as the size of the input and the time it takes to execute, grow at the same rate. The below diagram illustrates how a linear time operation works;

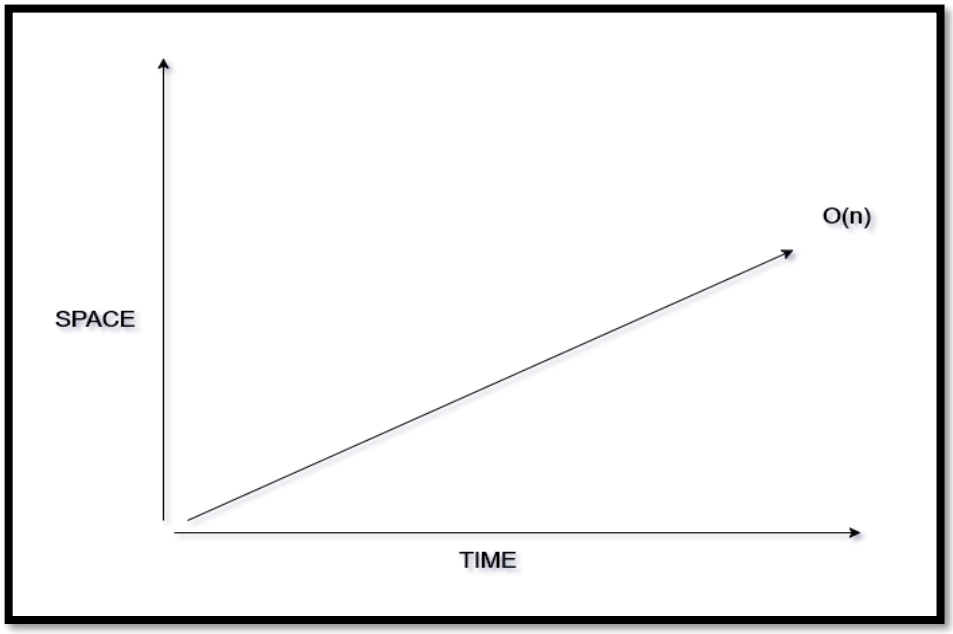


Figure 54: O(n)

Security and Validation

One of the highlights of this project, was the keen focus on Security and Validation from the outset of Project development. Security has already been covered in the System Design section of the Implementation Chapter, however this is more of an overview of what has been achieved versus what was set out to be done. Essentially, if this is discussed at a high level, the system is as secure as any large-scale application can be, which to be honest at the outset, seemed unrealistic. Spring Security features have been implemented and used well throughout the application and users won't be able to log in unless they have the correct credentials.

There were several different ideas about how to upgrade a user from being in a *TEMPORARY* state to being fully *REGISTERED*, whilst keeping the system always secure. There is no one way to do this, suffice it to say that the shortest, less complex route was taken, which yielded the same result, than if an alternative route had been taken. It was decided that when a user had applied for membership and the administrator approved the submission, to send an email with a temporary password. The key to keeping the system secure, is that a user will still be in a *TEMPORARY* state and because of the efficiency of Spring Security, won't be able to access other parts of the system until they have changed their password to a permanent one, which will make them fully registered automatically. The other benefit of this is that admin will have total visibility of which users are still *TEMPORARY* at any time.

The other part that was successfully implemented and incorporated into this application, is validation. This was done using a duel forked approach.

Client Side Validation

This is used to validate users input from an application form for example. The user is notified before submission is completed, if they are missing something that's required. The below example, illustrates what happens when a user forgets to input their email address;

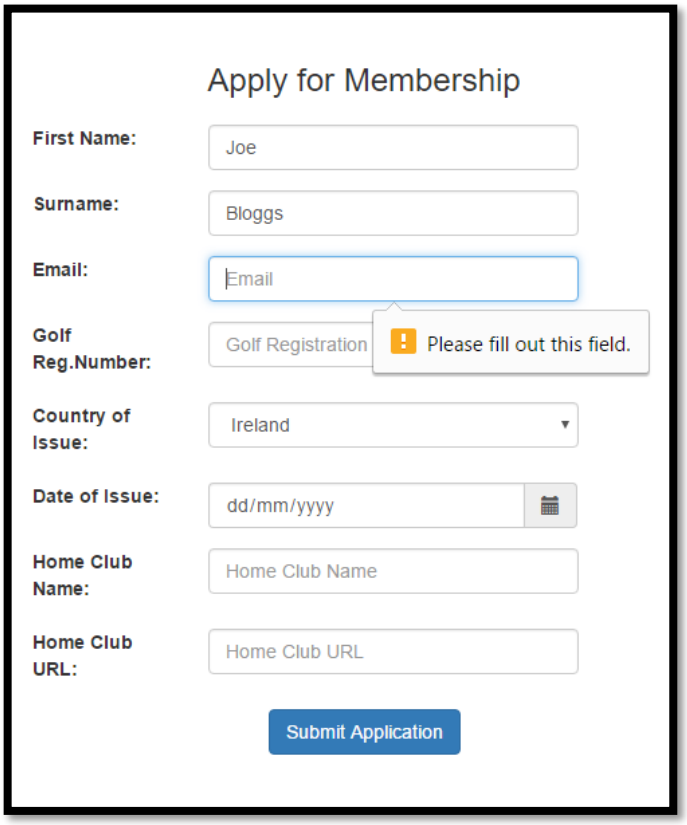


Figure 55: Client Side Validation Example

Server Side Validation

This type of validation is done on the server side. An example of when this would be needed with regard to this application, is when a candidate is applying for membership, but their email address is already registered on the system. This type of validation can't be checked on the client side as no information is stored there and as previously discussed, the user's information is held off site using mongodb, so before a candidate can successfully apply for membership, a check with the database needs to be completed to see if that email address isn't currently in use. Because of this, the database will always contain unique email address information.

The below example, illustrates this exact scenario;

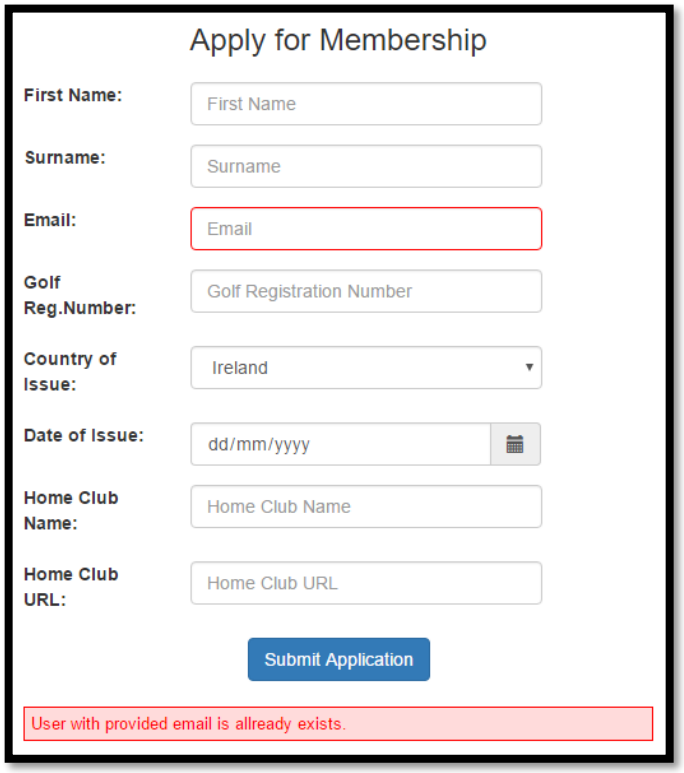


Figure 56: Server Side Validation Example

For a visual representation of who can access the various levels of the system, refer to Figure 7.

Deliverable Software Analysis

At this point, a large portion of how the project was put together has been discussed along with the methodologies used and how the project was managed etc. Now it's time to analyse what's actually been delivered. Upon starting this project, the aim was to incorporate as much of what has have learned over the last number of years in college, into a fully-fledged working web application.

Whilst its hugely important to analyse what works in this application, it's also equally important to look at other aspects of what has been delivered. Good Java programming is hinged on the Object-Oriented Programming (OOP) paradigm. It promotes a loosely coupled, highly cohesive, reusable approach through its fundamental pillars, Abstraction, Encapsulation, Inheritance and Polymorphism. This style of programming along with various cocktails of Design patterns have been used well throughout the application. This is key when designing any scalable system and its vitally important when highlighting the Deliverables of a software system, because quite frankly what's been incorporated from an OOP perspective works and works well. With the help of the Spring framework plus developer's knowledge, the team were able to accomplish this.

Key Operational Features

The below are the key features of the system and how they work;

- **Client Interface** – The front end of the application was created from the perspective of a given user and what needs and requirements they would have. Whilst the main focus was on the Software development itself, every multi-faceted part of the application was treated as a whole. There was a conscious effort to try and emulate the project storyboard as much as possible to deliver an operational front end, but also one that looks professional.

- **Security** – Works well in every area. As highlighted, a thorough security system has been incorporated into the application, that is robust to meet the needs highlighted in requirements which not only keeps a user's data private and safe, but also the system in its entirety.

- **Concurrency** – The system allows multiple users to access the application simultaneously, allowing for swift and scalable access.

- **Validation (Client and Server Side) –** Two types of Validation have been incorporated into the application. Client Side validation using JavaScript which restricts users from entering wrong or incomplete information. Server side validation which is completed on the server.

- **Database / mongodb –** Mongodb was used as a way of storing data layer information. This works with Key value pair information and uses a structure similar to JSON called BSON.

- **OOP / Design patterns –** An Object oriented approach was used in the project and where possible opted to work to abstractions over concrete classes, which allowed for Loose coupling and high cohesion. Several Design Patterns were also incorporated into the project, which helped aid the code flow and promote code reuse. Factory patterns, Singletons, MVC and Template patterns are some of them.

- **Spring Framework –** The Spring framework was quintessential during the Software Development phase. Here are some of the key Spring elements that this application used;
    - Project dependency management using Maven.
    - Web server and servlet incorporation using Tomcat.
    - Bean XML integration which allowed beans to be configured for use with project objects.
    - In depth security implementation

- **Cloud based AWS implementation –** The application is housed in the Cloud using Amazon Web Services. This includes the mongodb database layer. Refer to the Architecture Diagram in Chapter 4. The Golf'n Home Swap application is hosted by Amazon Web Services and can be viewed at the below address; http://52.214.71.85:8080/golfnswap/

- **Administration CSM implementation –** A client service management system has been incorporated into the application for the Administration personnel. This provides a robust, simplified and user friendly way for them to manage all types of users, be they fully registered or candidates who are waiting to be approved.

Limits of the system.

This system has been built on a large foundation and whilst it is scalable to a degree it's also worth noting that there will be limitations. For example, if the system were to experience a dramatic growth the current setup might need to be revisited to properly deal with the large amounts of traffic. This might include multiple servers clustered

together to help with load balancing. The same may need to be said of mongodb, which from research, doesn't support atomic transactions that would be required should a change be made to the system that incorporates a module which deals with the exchanging of money online. In this case, MySQL or some type of relation database would be the preferred option as they support transactions and incorporate the "All or nothing" atomic approach. The system isn't fully finished and there could be further limitations which have yet to be uncovered.

CHAPTER 6: CONCLUSIONS

Summary

Throughout this project's development lifecycle, there have been many experiences and learning outcomes achieved. These ranged from issues that arose at various stages such as the general System analysis associated with Project inception, to delivering a fully working, finished product. The truth is that without glitches or bumps in the road, no one can properly learn and so whilst it's imperative to reach a given goal within a specified time, it's also equally important that due diligence is given to the learning outcomes achieved from embarking on a journey. Of course, mistakes have been made along the way, however, nothing of gargantuan proportions impeded progress. In all honesty, the way that the project was organised from the outset, speaks volumes for the level and quality of work that was delivered. When the project specifications were received, work commenced immediately and rapidly. Sprints were set out and were nearly always completed before their due date. This approach allowed for time to deal with any problem areas, without having to deviate from the main plan or push out certain milestones. What helped, was brainstorming ideas during the inception stage of the project, creating mind maps and sketching out visual perspectives of which direction the project needed to go in. The storyboard received at the start of the project was quite large and although a large portion of the application is up and running, there is still more to be completed. That said, the application so far is robust, scalable and user-friendly. As detailed at great length throughout this dissertation, the approach taken was multi-angled regarding breaking down sections of work to be completed within the project.

The Methodology chapter outlines how this was achieved. Planning played a large part in the grand scheme of things, this meant holding regular meetings, delegating workloads, communicating information and documenting development strategies. Several new technologies were used to do this. Slack is a collaborative communication tool that was used by the development team. GitHub was used to not just manage version control commits and branches, but to manage all project tasks, which allowed for real-time knowledge to be displayed within the GitHub interface. This opened the door for a user-friendly, robust and concurrent way to collaborate on a project, manage tasks such as Milestones and issues, bugs and improvements. As also detailed in the Methodology Chapter, Slack was used to integrate with GitHub allowing for real-time repository update information to be displayed to the team.

During the development of this project, several aspects of various methodologies were used. The pair programming area associated with XP Programming, which is a subset of Agile was used at great length to work on specific tasks that required group development. In general, an Agile approach was adhered to, which followed an iterative, cyclical development process that also worked perfectly within the group. RAD or rapid application development was used to quickly build visual prototypes, that would eventually work their way into the main branch on the Golf'n Home Swap repository. Sprints, which are part of the scrum subset were used to set and monitor project tasks and deadlines.

Chapter 3 documents the various technologies used to create this project. Some of them included Communication tools such as Slack, GitHub and Google Drive that were used to collaborate and perform group tasks. IDE's such as Eclipse and Brackets that were used for development and command line used for MongoDB queries. Programming languages such as Java, JavaScript, CSS, HTML, MongoDB and frameworks Such as Spring, Bootstrap were used as the fundamental pieces of the projects workbench. Amazon Web Services (AWS) was used as the Cloud based platform as a service (PaaS). We chose Amazon because it allows for a more manual approach to cloud computing, which offered more freedom and more options and allowed for both the web portion and the data later of the application to be housed under the one roof, but in separate sections. The Golf'n Home Swap application is hosted by Amazon Web Services and can be viewed at the below address;

http://52.214.71.85:8080/golfnswap/

Some experimentation was used with Docker, which is a useful piece of software that allows for specific sections of the application, ie Databases, and The Web to be containerised. Then by running a single *DockerFile* or *.yml* file, these services can be launched within these containers, which allows for a neat separation of concerns, portability, and scalability. This part is still work in progress.

In Chapter 4, the Architecture used in this project was Documented. To summarise, it has several layers or tiers, that at an abstract level, each perform specific tasks within the application. The client /view tier concentrates on the user interface that they interact with. The Model or data layer focuses on the actual data that's being stored and used in the view. The Server or controller layer communicates between the two and acts like a glue that deals with processing requests and responses. This type of Architecture is known as MVC or Model, View Controller and is designed to logically segregate sections of the system in what's known as separation of concerns. The System design section details the fine grain granularity associated with the various modules of low-level development components, such as Spring, Maven, Object oriented programming, Design Patterns, MongoDB and general external and internal packages that were used within this project.

Chapter 5 looked at analysing the working parts of what's actually been delivered. In other words, does the application work. Firstly, robustness was measured and to prove this and as part of the iterative, incremental process that was used during project development, black and white box testing was used at the end of each sprint. When these parts or modules of the system were tested and working, only then were these changes merged with the main master branch. This created a degree of comfort, knowing that no major problems could emerge that would halt progress for too long. The next part that was documented was scalability and asked the question, is the system able to hold a satisfiable number of concurrent users and is it able to hold more if necessity dictates? Through research, it was discovered that if the current system experiences a large growth rate, then a more commercial software approach would need to be implemented, to facilitate large numbers of users. However, the application in its current form is sufficient for the short to medium term. This chapter also looked at validation and how the system responds to incorrect or incomplete information that's entered by the user, where client and server side validation disabled users from doing this. Space and time complexity of the application was also analysed in this chapter, where it was proven that the application can run any of its algorithms in polynomial time.

Learning Outcomes

Now that each chapter has been reviewed and summarised, it's time to reflect on some key learning outcomes;

- **Team Players / Teamwork** – Being able to work as part of a team is probably the most important aspect of working in a group project. The software development team that were chosen for this project, work extremely well together, but in the real world, a person won't usually have the opportunity to choose who they work with, but the job will still need to be completed.

- **Communication** – This is a large part of anyone's daily life, especially where a team is working together to try and achieve a common goal. It became evident that as the project progressed that the better the communication, the better the result. The more communication between a team, the more issues can be noticed and ironed out before it bubbles into something much larger.

- **Exposure to new Technologies** – Being immersed in new technologies is something that is both challenging and rewarding. In one hand, there is the learning curve associated with learning something new, but equally the elation experienced when something finally clicks into place. Most of the technologies used in this project are

modern and cutting edge, displaying modern architectures and methodologies. However, a lot of these were new to the development team and so required a strong mindset and hard work to eventually grasp the core concepts associated with each new technology.

- **Analytical Approach** – From the outset, an analytical approach was encouraged and nurtured in order to reach goals and avoid potential problems. This wasn't always easy as sometimes it's easier to just dive into something without thinking things through properly, but as a team learned that analysis even at its smallest level, can help to overcome large-scale issues.

- **Planning** – Is something that can easily be taken for granted and overlooked. All projects need to have some degree of planning in order to succeed. Before starting each section, learning to take a careful approach on how to tackle specific sections of the project, was quintessential.

- **Documentation** – Another learning outcome from this project was the importance of documentation. This may seem obvious, but without properly documenting your project, it makes things extremely difficult for anybody else to understand what's happening. Everything in this project has been documented. This might be for example using GitHub to commit, comment and branch specific versions of the project, or simply keeping track of a weekly diary which detailed what tasks were completed each week. It might also have been the detailed, in code comments used to explain how each part of the system works. All this makes it much easier for several reasons. For the current developers working on this project, who need to refresh what's happening at certain points of the application or for the next set of developers who will be involved in the handover process.

- **Problem Solving Skills** – This is one learning outcome, that has made an impact. What was realised as the project progressed, was that there may be any one problem, but there are many ways in which to solve it. In software development, it's easy to overlook a simple problem and look for something more complex, that takes a person down a never-ending rabbit hole. To avoid this, each problem was analysed, broken down into its simplest form, where an equally simple, but agreeable solution was offered.

Reflection

When a team of student software developers embarks on a journey, the route is not always certain, but the destination is. Essentially, there can be many routes that lead to a goal. That goal could be a finished or completed task or a fully operational piece of work that has been carved and molded into something that does exactly what it should and something that the team should be proud of. A Goal could also be resolving a specific issue that could potentially evolve into a larger 'Show Stopper'. The point is that as software developers, it's never clear what route should be taken or what exactly that goal should be, it's really just all part of the Software Development process and there won't always be a level of control over things. However, with a solid team, detailed plan of action and reliable tools, anything in software development is possible.

Golf'n Home Swap is still a work in progress application, with the next steps being continued by another development team, but a solid framework has been created. If given the opportunity to go back and start again, there is very little that would be done differently, however, perhaps a front-end framework such as AngularJS could have been incorporated into the project, to add another layer of concern separation. Or maybe some jQuery to add some dynamic animations to add to the user's visual experience. As mentioned, Docker was another option that was experimented with, but not implemented. This would offer a more portable friendly option, allowing a complete application to run in any location with the minimum amount of effort. This would offer an agreeable

alternative to AWS (Amazon Web Services), that took quite a bit of time to set up and configure to suit this application.

On a positive note, it’s been a joy to work on this project, where there has been a wealth of knowledge both learned and shared. It’s not every day that one gets the opportunity to work on a project of this scale or work with a team of developers, project owners and project supervisors in such a professional capacity.

REFERENCES

[1] Spring Docs: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[2] Mind Map: https://www.text2mindmap.com

[3] Bootstrap: http://getbootstrap.com/

[4] Threads: http://stackoverflow.com/questions/6402681/how-to-increase-number-of-threads-in-tomcat-thread-pool

[5] Java Scalability: http://www.javalobby.org/java/forums/t92965.html

[6] Tomcat Comparison: http://www.asjava.com/jetty/jetty-vs-tomcat-performance-comparison/

[7] Spring Security: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

[8] Java Documentation: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

[9] Apache maven: https://maven.apache.org/guides/index.html

[10] Web MVC framework: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[11] Git documentation: https://git-scm.com/doc

[12] JavaTPoint design patterns: http://www.javatpoint.com/factory-method-design-pattern

[13] Spring Security: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/