Development of a map-based web application showcasing the Dyfi Wildlife Centre

CS39440 Major Project Report

Author: Michael Male (mim39@aber.ac.uk)
Supervisor: Dr Edel Sherratt (eds@aber.ac.uk)

10th May 2020
Version: 0.8 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (includes Foundation Year) (G40F)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name Michael Male

Date 10/05/2020

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Michael Male

Date 10/05/2020

Acknowledgements

I am grateful to Mr Emyr Evans and Mr Thomas Faulkner of the Montgomeryshire Wildlife Trust for their support in the development of this project, and hope that the final product is useful to them.

I'd like to thank the Department of Computer Science for their provision of support and resources, particularly from a remote perspective during the COVID-19 pandemic in the second half of the project. Specifically, I'd like to thank my supervisor, Dr Edel Sherratt, for her help, support and encouragement during all stages of this project, as well as Mr Richard Shipman, who provided useful feedback in the mid-project demonstration that ensured it was on the right tracks.

Abstract

The Dyfi Wildlife Centre is a visitor centre run by the Montgomeryshire Wildlife Trust. It is situated on the Cors Dyfi Nature Reserve in Powys, Wales. The Trust had approached the University, with a request for a solution to assist in showcasing the reserve's work, and its place as an osprey conservation, engagement, and research project. They had procured an 86-inch touchscreen monitor, in the hopes of presenting an application which aids in achieving the aforementioned goal.

The software created in this project takes the form of a map-based web application. The front-end makes use of the Google Maps for JavaScript API, as well as HTML, CSS and Thymeleaf, to show visitors a map of the centre and its surroundings. The map has various markers, and filters, that can be clicked on, showing further information about the point of interest. This is backed up by a RESTful API backend, created using the Spring Framework, a model-view-controller framework using Java Enterprise Edition. An administration panel was also developed, involving authentication, and allowing the authenticated user to add, edit, and delete points of interest. Data was stored in a PostgreSQL relational database.

Planning and development of this project took the form of an agile approach, with ideas from both Kanban and Extreme Programming used and adapted to fit a single-developer project. A meeting with the customer took place prior to development, and user stories and prioritisation of tasks had branched out from that meeting. The concept of test-driven development was a driving force for the development of this project, with various testing suites including JUnit, HTMLUnit and Selenium utilised when creating test cases. A key part of this project was ensuring that the customer was kept up-to-date, and supplementary documentation was created for this project that provided a brief overview of the required setup and how to use the web app.

Contents

List of Figures

List of Tables

Chapter 1

Background & Objectives

1.1 Background Planning & Analysis

A number of key considerations were taken into account during the background planning stages of this project.

An initial meeting took place in February 2020 with the customer, the Montgomeryshire Wildlife Trust. The hardware implementation was discussed and it was understood that the customer would want to run the web application on an 86-inch touch screen monitor. They also expressed a preference for the web application to be run locally, and not published to a cloud service or the World Wide Web. Therefore, an important consideration for the project was enabling the web application to work using touch gestures, and use a responsive layout which would scale to a high resolution.

With the intention being that the application runs locally, therefore research had to be put into various technology stacks, and which web framework would work best for this kind of application. Research into this would have to take into account how much time needed to be allocated to perform spike work; ensuring a comprehensive understanding of the framework was achieved before work on the project began.

Another key consideration was the vendor for the maps API. A number of APIs were assessed for their usability as well as their licensing conditions.

In this section, the early investigative work into the project will be discussed, with an analysis of the steps taken to achieve a conclusion as to which technology stack to use.

1.1.1 Web Development Frameworks and Tools

Early on in the project it was decided that the use of a comprehensive web framework would be beneficial to the usability and production quality of the web application. This was opposed to a simple HTML 5, CSS, and JavaScript stack. Whilst it was inevitable that portions of each language had to be used, a more robust framework provided specific advantages, such as a package manager and cleaner code.

Three primary web frameworks were assessed, through the use of research, prior reading, and spike work, these being the following:

- A JavaScript-based framework, utilising the Node.js and Express backend frameworks and either Vue.js or React as the frontend.
- Django, a Python-based model-template-view framework.
- The Spring Framework, a Java-based model-view-controller framework.

Further research took place into the CSS frameworks, as well as an appropriate database management system. Discussion into these are expanded upon in their relevant sections.

1.1.1.1 JavaScript Frameworks

A full-stack JavaScript framework allowed for the benefit of the software being written in one programming language, which could reduce issues with the code's readability and maintenance. It would have also allowed for a single testing framework, such as Jest, a testing framework maintained by Facebook [1]. The use of a runtime environment would be standard fare for a JavaScript framework, allowing sever-side scripting and dynamic web pages to be run outside of the usual web browser environment that JavaScript runs on. A popular JavaScript runtime is Node.js, which has been touted as a resource-efficient framework, a benefit for a project that is designed to run on a local computer [2].

Frontend frameworks were also looked at, with varying levels of spike work being put into them. React, a Facebook-maintained JavaScript library, and Vue.js, were both considered. The two frameworks are rather similar, such that they rely on sending data directly to the browser's Document Object Model, however Vue.js takes a declarative approach to HTML scripting, whilst React uses JSX, an HTML syntax extension to JavaScript [3].

Ultimately, a lack of familiarity with JavaScript, as well as the relative complexities of the frameworks, proved to be a deciding factor in not going ahead with a JavaScript-driven application. During research, it was deemed that a large amount of time would have to be dedicated to following tutorials and learning JavaScript, and ECMAScript 6, from scratch, and this would have taken too much time and risked a less complete final product.

1.1.1.2 Django

Django is an open-source framework based on the Python programming language. Its creators set the framework's primary philosophies as a quick approach to development, a view to not repeating the design and execution of concepts, and loose coupling - in this context being that the framework layers shouldn't be able to interface with each other unless necessary [4].

Django utilises the object-oriented programming paradigm with Python, and utilises a model-template-view approach. In short, these are three distinct layers of a web application: the model consists of a data structure, the view consists of a representation of

information in a web browser, and a controller allows access to this data with meaningful requests [5].

Whilst Django appeared to be a good choice due to its highly cohesive pattern, testing of the setup proved to be difficult on occasion, with design patterns that were particularly unique to Django. Python is a language that is known to utilise concepts that are simple to understand for users with greater knowledge in other programming languages, utilisation of Python at this level required a higher level of expertise than was had at the time. It was decided that there would be a greater chance of success with the project if attention was placed more towards frameworks with a more familiar language, to avoid an inordinate amount of time being spent on the intricacies of specific programming languages and frameworks.

1.1.1.3 Spring Framework

The Spring Framework is an open-source framework, where its web application features are based upon Java Enterprise Edition, an enterprise specification of the Java programming language that has modules specifically tailored towards web services [6].

Spring provides many of the benefits that Django also provides, and the two are often compared against each other. Similar to Django, Spring utilises a model-view-controller framework, and relies upon high cohesion. As Spring uses Java, it takes advantage of the object-oriented programming paradigm, and it is standard for classes to be written in such a way that they follow this concept. Spring is optimised to work with the Thymeleaf template engine, that provides server-side scripting and an interface between the controller and view layers [7].

Ultimately, proficiency in Java from previous academic study and personal use made Spring an ideal choice for this project. Spring Boot, an addition to the platform that allows for automatic configuration of core dependencies, was also used, to reduce the amount of time spent studying elements of Spring that Spring Boot renders redundant. Spring also utilises Maven, a Java package manager, to provide a large number of dependencies; Spring Security was deemed a useful tool for an authentication layer, for example. An added benefit to Spring is its embedded Apache Tomcat server, that allows for the provision of an HTTP web server environment from opening the application, rather than having to take steps to deploy it into an existing web server environment.

1.1.1.4 CSS frameworks

It was decided in the planning stages of the project that it would be beneficial to use a CSS framework, rather than build a template with 'vanilla' implementations of HTML and CSS. CSS frameworks provide a large number of pre-built elements, many that are rather familiar to users, due to their prevalence in front-end web design. A review into three different CSS frameworks were performed.

Bootstrap, a framework initially developed for use with Twitter, provides elements that have been crafted with the User Experience at their forefront. It is used by a large variety of

web applications and websites, with the developers claiming it is 'the world's most popular front-end component libary.' Whilst this would have been a good choice for a familiar user interface, the framework was assessed to have less customisability, which posed an issue with a unique implementation where the main aspect is a single-page application. Bootstrap is also intended to be mobile-first, a feature that is not required, as the application is designed to run on a desktop computer [8].

Fomantic UI, a community fork of Semantic UI, which had seen a lull in development, is a framework that defines itself as using 'human-friendly HTML.' Classes within Fomantic use syntax from the English language, for example, a user interface with three buttons could be classed simply as `<div class="ui three buttons">`. While Semantic is quite an elegant interface, different frameworks were deemed more familiar to a user, and the User Experience aspect of this project was tailored towards people who may not have a great amount of technical knowledge [9].

The chosen CSS framework for the application was Materialize, a variation upon Google's Material Design language. Material Design is used in a large number of Android mobile phone applications, and on Google's services themselves. Similar to Fomantic, Materialize utilised the concept of human-friendly HTML, and was easy to integrate with Thymeleaf and Spring. Materialize utilises a twelve-column responsive grid system, which made it simple to create components that would scale with screen size. As the customer intends to run the application on a large monitor, this is an important design consideration [10].

1.1.1.5 Database Management System

A database management system was a key component of this project. Points of Interest had to be persistent, and, in later iterations, a database for users had to be created. Whilst an in-memory database management system, H2, was used during the early stages of the project, it was quickly settled upon that a server independent from the application should be created. The investigation into this settled on either using PostgreSQL, an SQL-compliant system written in C, or MongoDB, a NoSQL document-oriented database that takes a JSON-like approach to storage of data.

It was decided upon to utilise PostgreSQL. Whilst there were various arguments for using MongoDB - a more readable schema, for example, studies have shown that PostgreSQL is generally faster in response times with smaller data sets, and it is a popular implementation of SQL that sees good compatibility with Spring [11]. A solid amount of background experience with PostgreSQL also contributed to the decision; whilst MongoDB has been described by its developers as similar to an object-oriented paradigm, the performance benefits of PostgreSQL negated any benefits of learning a new system.

1.1.1.6 Programming Tools

After the technology stack was decided, a decision had to be made as to the best programming tools for the task at hand. The Spring Framework has widespread support within IDEs. An extension, called Spring Tools 4, had been created by the developers, which provided Spring-related functionality to Eclipse, Microsoft Visual Studio Tools, and

Eclipse Theia [12]. A large number of PostgreSQL clients existed, with some examples being pgAdmin 4 and HeidiSQL. PostgreSQL could also be administrated via a command-line interface [13].

However, it was decided upon that the JetBrains' suite of software was to be used for development. JetBrains provides a free license to people with a University e-mail address [14], and its IntelliJ IDEA Java IDE has in-built support for Spring Boot projects, and also includes plugins for web development. DataGrip, a database IDE, was also used, as this allowed for a graphical representation of the database that resulted in the schema being easier to quickly understand. Various browsers, including Microsoft Edge, Google Chrome, and Mozilla Firefox, were used in order to test the web application on varying browsers. Mozilla's Firefox Browser Developer Edition was useful when debugging the web app, as it included useful tools pertinent to CSS and JavaScript debugging [15].

1.1.2 Mapping API

A crucial part of the application was the ability to present geographical data and information in a graphical format. A video had been provided by the customer, showing a map implementation on a similar nature reserve in Dorset. This used a satellite map with markers placed on the screen, and the intention was to take a similar approach [16].

As this application is web-based, a view towards a JavaScript API was adopted when assessing potential mapping solutions. The map must also take clickable markers and allow for quick downloading of satellite images. Two potential candidates were assessed:

- **OpenStreetMap** - A free and open-source map that allowed for users to request changes to be made, which has the potential to provide a more up-to-date map based on changes in geography. OpenStreetMap does not have an inbuilt satellite implementation or an API, however various free and paid sources are available and were assessed during this stage of the project.
- **Google Maps Platform** - Google provides a JavaScript API with a wide array of features attached to it, that uses data and imagery from Google Maps, hosted on Google's infrastructure. A potential hurdle in using Google Maps Platform, however, was its use of API keys and its pricing structure, an issue that is further discussed below.

1.1.2.1 OpenStreetMap

OpenStreetMap is defined as a community-driven repository of map data, that can be contributed towards in a 'wiki-like' manner [17]. It did not, however, appear to provide a JavaScript API, but many implementations of OpenStreetMap have been realised in JavaScript, with Leaflet being one of them [18].

OpenStreetMap did not appear to host aerial imagery, and freeware JavaScript APIs only appear to have implemented a standard map interface. It was considered that this is not what the customer had wanted, and, whilst Mapbox, a closed-source implementation

of OpenStreetMap with aerial imagery and a free tier, was briefly assessed, a lack of customisability and the low-quality resolution of the aerial imagery were deciding factors in not going ahead with this approach [19].

1.1.2.2 Google Maps Platform

Google Maps Platform is one of the features offered with Google Cloud; a suite of cloud-based applications and APIs hosted centrally by Google. Google Maps is a popular mapping interface, and regular input from various countries' national mapping agencies allows it to be reasonably up-to-date. Google Maps Platform's main offering for non-mobile applications is the Maps JavaScript API, and a thorough amount of documentation has been provided for this [20].

Ultimately, a decision was made as to go ahead with using Google Maps Platform. The platform's high-resolution aerial imagery was key to this decision, along with the general reliability and uptime of Google's Cloud infrastructure. Unlike OpenStreetMap, the platform was not free, however includes a free tier that permits for up to USD $200 of free usage a month. After reviewing their pricing scheme it was decided that this would be enough for a locally-hosted application; the geo-coding API was not being used and a levy of USD $2 is placed on every one thousand static map requests, meaning that one hundred thousand requests a month would have to be made to exceed the free tier, which is not likely [21].

1.1.3 Version Control

An effective Version Control System was imperative for the successful completion of the project. The use of branches, and rollback features, were key to ensuring that each part of the project was clearly indicated. An external version control host, GitHub, was used, which uses the Git version control system. This was mainly due to personal experience with both Git and GitHub, the added features that GitHub offers, and the benefit of an externally-hosted backup location, should data loss occur at any point on a local development machine [22].

1.2 Processes

An important part of the planning for development of this software centred around having a meaningful development process, so as to ensure that the project remained on track and all deliverables could be met. A number of different approaches to software development were researched, including the Waterfall model, a linear, sequential approach that focuses on a series of categories and tasks towards the completed product. It was, however, decided to take an agile approach to software development. An agile approach would allow for changes and improvements to be made to the product as the development process continued and ensure there was more freedom attributed to the implementation of the product.

1.2.1 Agile Development

An important aspect into choosing an agile methodology was the ability to easily adapt it to a one-person project. Most agile methodologies are tailored towards small team projects, and many of the principles behind the Agile Manifesto - conveying information to and within a development team through face-to-face conversation, for example - are rendered irrelevant. Ultimately, I decided upon an adapted version of the core tenets of eXtreme Programming for the brunt of the development process, which I have detailed below. I also decided upon the use of Kanban as a workflow management tool. A few tools for this were assessed, including Trello and Jira, however, as the project was being hosted on the GitHub , GitHub Projects was instead used.

1.2.1.1 Adapted XP principles

A number of studies focused on adapting XP and Agile for one person were consulted when deciding upon its implementation [23]. Implementing a fully-compliant XP approach would have been difficult. It is a complicated framework, and requires multiple roles. Instead, a relaxed version of XP, that loosely followed the twelve core practices, was adopted [24].

1. **The Planning Game** - This was adapted from the initial meeting of a customer, where a series of notes on the type of features the customer wants in the product was compiled. Planning took an approach of user epics and stories, the epics being:

   - As a volunteer, I want to have a web interface where I can show people information about the nature reserve on a map.
   - As an administrator, I want to be able to add and edit information about the nature reserve, and change anything that I want to change.

   The system metaphor became *'A web application that uses a map to show interesting things around the Dyfi Wildlife Centre'*. Appendix A of this report provides the copy of the document used to design user stories, define the technology stack, and prioritise each user story.

2. **Small Releases** - An iterative approach was taken to the development of this project. Releases, each including more features than the last, were tied into the tasks created in the planning game.

3. **Metaphors** - This practice was realised through the aforementioned system metaphor and user stories.

4. **Simple Design** - Attention was placed on the design of the application not becoming too complex, with this being assessed in every release.

5. **Testing** - As will be discussed later in this chapter, test-driven development was used throughout the project. This often involved verification of an interface before its implementation.

6. **Refactoring** - It was ensured this was the last task, after all tests passed, so as to provide as clean, readable and efficient code as possible.
7. **Pair Programming** - For obvious reasons, it was not possible to replicate this practice. However, a benefit of pair programming is being able to have a 'bird's eye' view of the code, and find bugs early. This was achieved through use of the Pomodoro Technique, a time management technique similar to timeboxing, where work is broken down into 25-minute intervals [25]. After three intervals, all code that had been written was tested and reviewed, to see if there were any bugs.
8. **Collective Code Ownership** - As the project had a single developer there was no need to find an adaptation to this, as the entire codebase is owned by the developer.
9. **Continuous Integration** - Use of the Travis CI tool was prevalent throughout the project. Whilst conflicts between different branches were not likely, it ensured that testing was being carried out repeatedly, and did not have to be performed manually [26].
10. **40-Hour Week** - It was ensured that too much work over one period did not occur, as this may have caused code of less quality due to stress and fatigue.
11. **On-site customer** - This wasn't possible with this project, particularly during the second half. Therefore, any communication with the customer tended to be carried out via e-mail and telephone.
12. **Coding standard** - A coding standard was self-imposed, and kept consistent throughout each iteration of the project.

1.2.2 Test-driven development

While researching testing strategies, it was decided that the concept of test-driven development should be incorporated into this project. The main benefit of such an approach was the ability to create a detailed specification for the code, and ensure that thought took place into what was really required from it. It also ensured that feedback was quick, a useful tool in a single-person project where you are not easily able to request constant feedback from your code. Errors and problems become identified more quickly. Overall, this was to be implemented through the use of Java interfaces, with an outline of a class written and tests to achieve what was wanted from each class. Testing suites were created for each class which required them, and the use of technologies such as Selenium was crucial to testing the front-end.

Chapter 2

Design

Once an idea of which technologies to use in the software became prevalent, a design and structure of the application had to be created, based upon the requirements and user stories outlined in Appendix A, section 1.5. This involves a 'big picture' overview of each component of the application, where it is judged as to how to split the components. Later on in the design stages, planning, including through the use of entity-relationship modelling, and class diagrams for object-oriented components, was carried out on each specific component of the software.

This chapter will discuss the design choices made, and link them to the software's requirements. It will include any diagrams that were completed as part of this.

2.1 Overview

2.1.1 Architecture

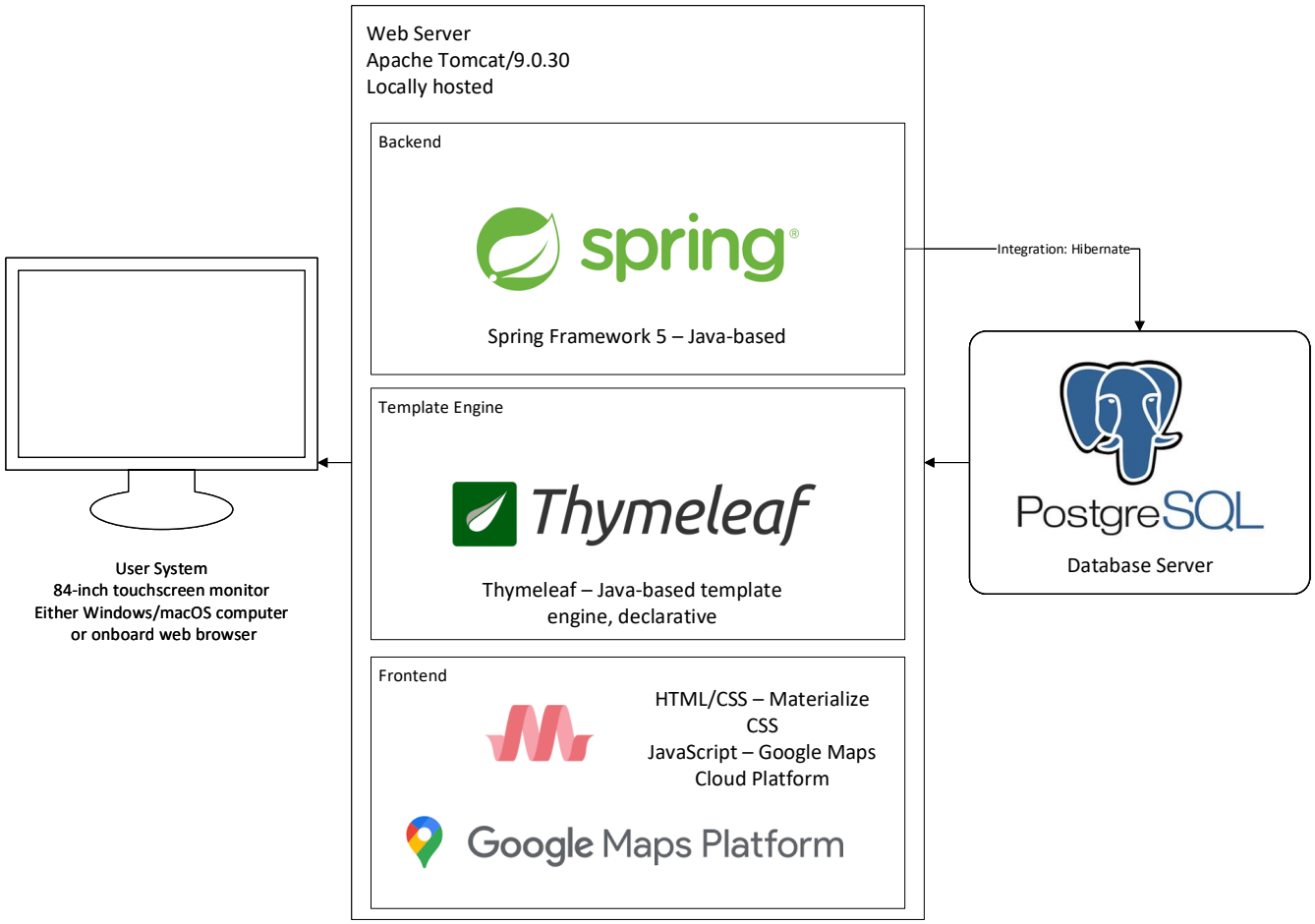


Figure 2.1: Architecture diagram for the Dyfi Wildlife Centre Web App

Developing an architecture diagram ensured that there was a definition to the technology stack that the application is going to use. It also allowed for some prior thought to how different components interact with eachother.

For example, the database management system being used is PostgreSQL. Unlike systems such as SQLite, which is generally provided as a standalone file, PostgreSQL requires its own server instance. Therefore, research had to take place as to how the backend, running on Spring, would interface with the SQL Server. As evidenced in Figure 2.1, this was through Hibernate, an implementation of the standard Java persistence API for Spring's enterprise-level dialect. This was marked as something to research when planning the design for the model and controller layers of the backend.

The architecture diagram also identifies the relationships between each component hosted on the web server, in a downwards fashion. The Spring Framework interfaces with Thymeleaf,

through Spring passing parameters to Thymeleaf templates, for them to be rendered. The frontend and user experience components of the application are clearly defined - with Materialize being used for the CSS layout, and the Google Maps Cloud Platform's JavaScript API being used for displaying the map and its markers. On the left-hand side, the user's system is explicitly mentioned, as User Experience is an important consideration in this project. The user will also be interacting directly with the system and its various components that are locally hosted on the web server.

2.2 Database

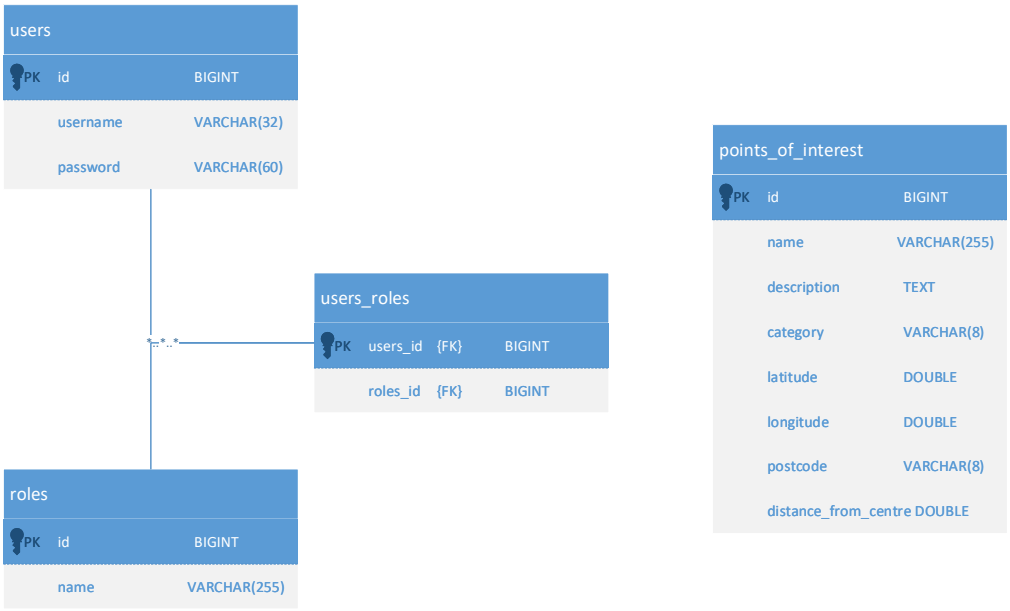
Parts of the database model were able to be generated automatically using the Spring Data JPA, however it was decided that a pre-built implementation of the database model would be easier to implement, as this would ensure adherence to an entity-relationship diagram, as well as any constraints that had been identified.

The initial iteration of the database, to fulfil the first two tasks that had been defined, was to have a single relation containing details about a point of interest. It was decided to initially plan the database in un-normalised form. This was to ensure that the Java object that was to be created in the back-end was readable and easy to maintain, rather than having to create multiple Java objects that represent a large number of relations in the database.

The database was to be iterated upon. To allow for authentication, separate tables for users and roles were to be created. While the consideration with regards to authentication was that every user should be automatically granted admin permissions, the role database allows for further addition of new roles, should this become a concern in a later version of the project.

2.2.1 Entity-Relationship Modelling

Figure 2.2: Second iteration of the Dyfi Wildlife Centre schema



Ultimately, the database design was uncomplicated, as there were only three objects to consider. As the majority of database handling was intended to be carried out through the Spring Data JPA, the nuances that that brings were added into the database at time of design. One example would be the ID primary key on each relation. In a standard PostgreSQL database implementation, the data type of the ID is not likely to be of type *bigint*. This is a type that is, as the name implies, intended for large integers. An ID attribute that is being utilised as a primary key would generally take the type of *serial* - an auto-incrementing column. This is, however, handled within the backend by the JPA, through an automatically-generated sequence stored within the database, therefore there is no need to use the serial type.

It is also true that, in a standard database, there would be an argument for not having an automatically-generated primary key, and instead using another unique property, such as name, or a compound primary key. However, this implementation was chosen as it allowed for faster efficiency within the backend layer of the application - a primitive type would be able to be matched faster than a String object with a string-matching algorithm

behind it.

A many-to-many relationship between both users and roles is achieved through the use of a junction able, *users_roles*. This allows for multiple roles to be added into the database at a later date, which can then be attributed to users through the frontend. As each user is only permitted to have one role, the primary key has been set as the user in the junction table, with a role ID being set with each user.

2.2.2 Constraints

Constraints had to be created in order to validate the data that was being entered into the table, this is described below, with the definition in a pseudo-code format:

Constraint Name	**Definition**
latitude_chk	CHECK latitude IS >-90 AND <90
longitude_chk	CHECK longitude IS >-180 AND <180
latitude_not_null_island_chk	CHECK latitude NOT EQUAL TO 0
longitude_not_null_island_chk	CHECK longitude NOT EQUAL TO 0
chk_name	CHECK name IS NOT EMPTY
postcode_chk	CHECK postcode MATCHES UK postcode regex

Table 2.1: A list of constraints in the points_of_interest relation

These constraints tend to perform sanity checks on the input that is being entered, once it reaches the database layer. The constraints on the coordinates are based upon the natural constraints based upon latitude and longitude, and ensures that a user cannot apply an incorrect coordinate to a point of interest, that would likely cause an error at the frontend layer of the application.

The two “not null island” checks make sure that a user cannot simply enter 0 as the latitude and 0 as the longitude, as this could potentially cause issues when parsing the postcode in the backend layer. The primary issue with a check like this would be that 0,0 is a valid coordinate pair. However, the customer has specified that points of interest would primarily be in the United Kingdom, which is far from this coordinate range. It is not likely that any potential coordinate pairs entered for points of interest outside the UK would be equal to these coordinates.

A regular expression, sought from an open data source that provides APIs dealing with UK postcodes [27], was, again, a sanity check. It performs basic error-checking that checks the general shape of the postcode; checking it has a letter at the beginning and ends in a letter, for example. Stricter regular expressions were available, however improvements were marginal and an API to confirm the postcode was valid was still required, due to the inability for a regular expression to rule out false positives. The design accounts for error-checking of postcodes through the use of the aforementioned API, which is further explained in section 2.3.

2.3 Backend

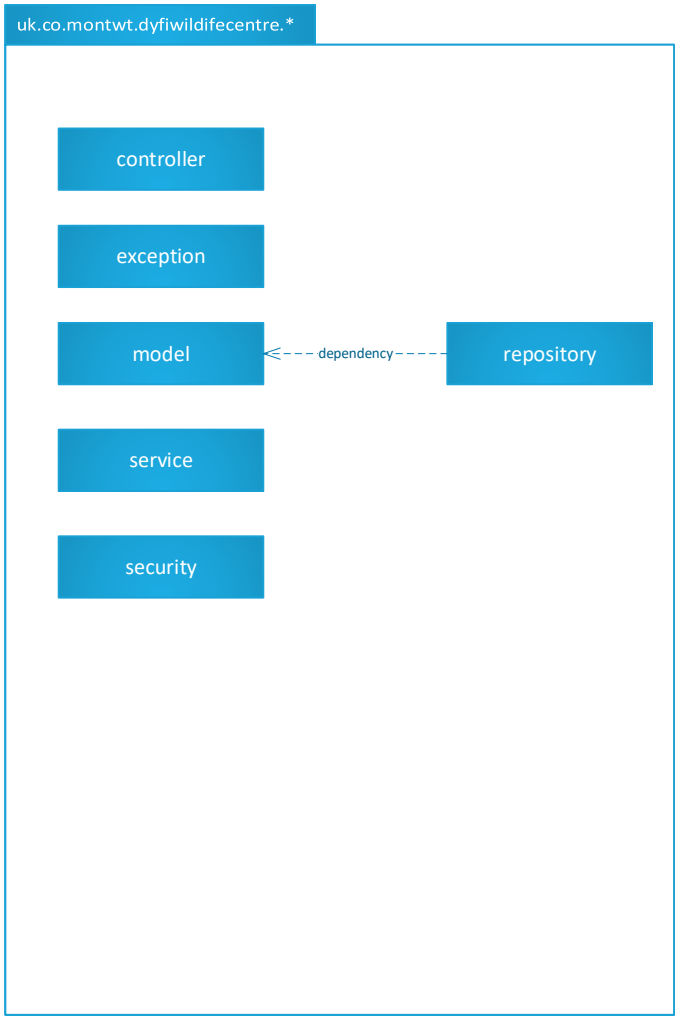


Figure 2.3: Parent UML diagram, showing each package that the software breaks down into

Spring, the framework that is utilised in this backend, follows a model-view-controller design pattern. This allows for a clear distinction between each area of the application, with its functionalities and design being defined as follows:

- **Model** - The application's data structure and logic processing, containing the objects representing a user, a role, and a point of interest. It also includes a repository layer, that interfaces with the PostgreSQL server, and a service layer, that provides a layer of abstraction between the database and the controller.
- **View** - The presentation of data in the application. This is generally further discussed in Section 2.4.

- **Controller** - Classes that accept user and computer input, converting it to commands that affect either the model or the view. In this application, the controller consists of a RESTful API for managing users and points of interest.

Class diagrams were created to be adhered to during the production of this application, and were split by layer. For the sake of clarity; constructor, setter, and getter methods are not represented on the class diagram. The intention was for Java packages to be used, to aid in code navigability and ease of maintenance.

2.3.1 Model layer

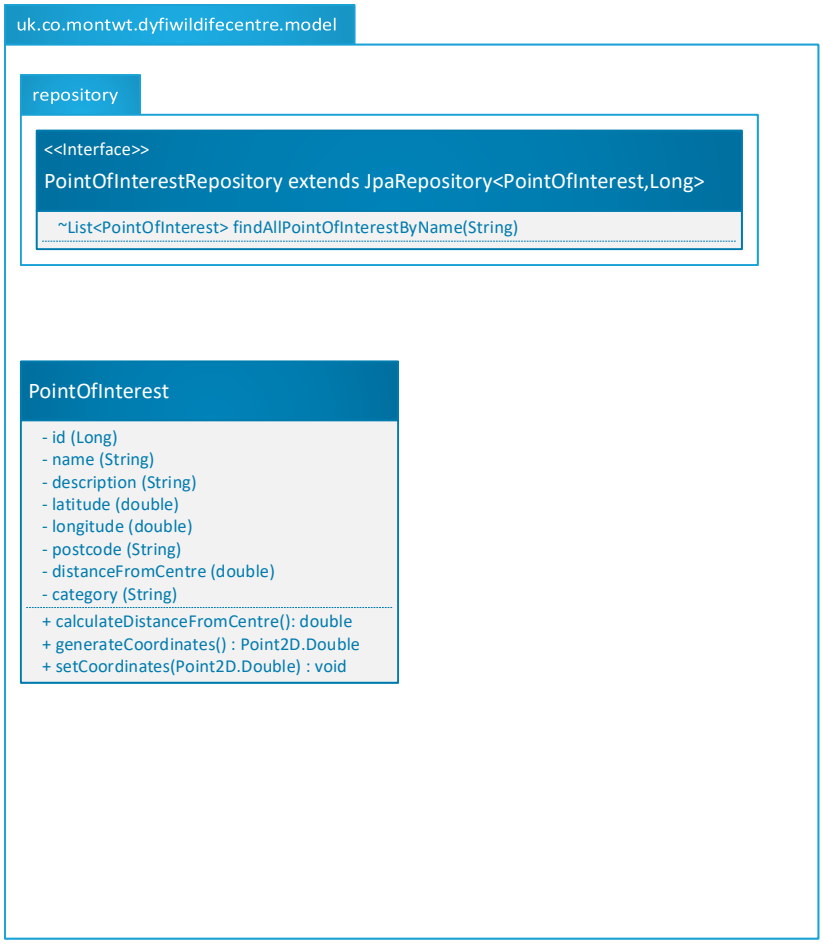


Figure 2.4: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.model` package

The primary model layer contains one class, as well as a sub-package. The sub-package contains an interface that accesses the *point_of_interest* relation in the database, utilising a Spring Data JPA superclass. It contains one method, that is intended to invoke an SQL command directly that finds a point of interest by its name, rather than its ID.

The `PointOfInterest.java` class defines a point of interest in an identical fashion to that of the database layer. This is required by the Spring Data JPA, and results in fewer manual configurations to be carried out. It was noted that a refactoring of this class diagram could include a further class, named Location, that includes the location-based members of PointOfInterest. This could be represented in the database by a one-to-one relationship. However, at least initially, this was not to be represented in the data modelling, and as the project is being developed in an agile fashion could be refactored in the future if considered necessary.

The `calculateDistanceFromCentre()` is designed such that it calculates the great-circle distance between two pairs of coordinates, with one being the coordinates for the Dyfi Wildlife Centre. This calculation would be in miles, to four significant figures, and would help in presenting users the distance between points of interest and the visitor centre. The *Haversine formula*, a formula in spherical trigonometry to determine distance between two points on a sphere, was to be implemented, with the code that was used available in Appendix C, section 3.1. Although the Earth is not a perfect sphere, Haversine's formula provides an acceptable approximation of distance between two points, taking into account the short distances that the user's requirements specify.

2.3.2 Controller Layer

The controller layer at this level of the application is intended to act as a RESTful API that interfaces with a point of interest and its database. Therefore, the HTTP requests that surround the API are intended to be as clear as possible as to their intention with as much avoidance of side-effects as possible. The controller layer also includes a sub-layer, to interface between itself and the database.

2.3.2.1 Services

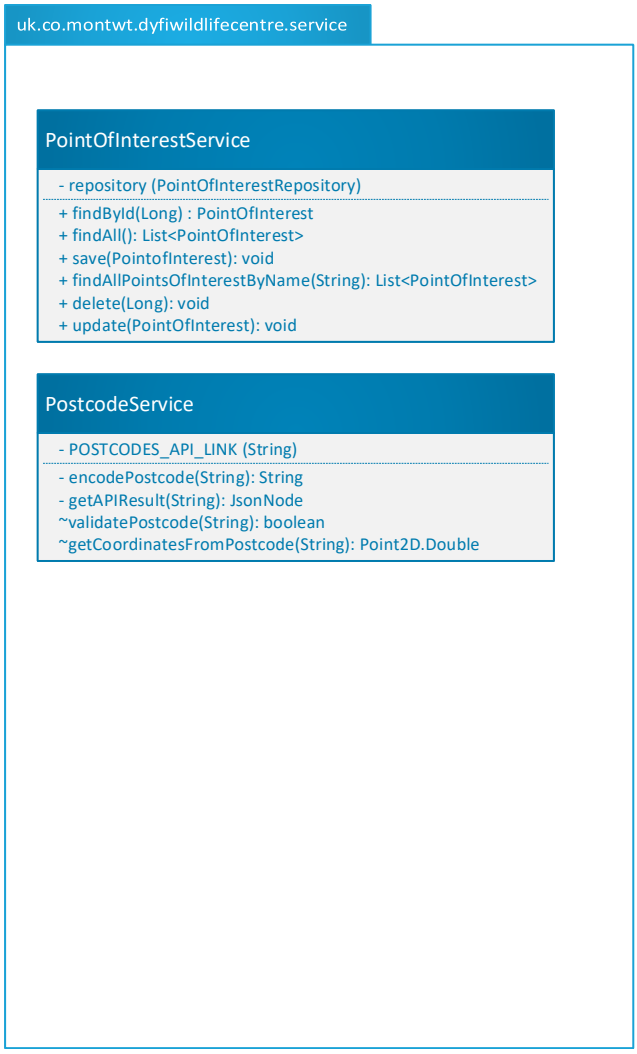


Figure 2.5: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.service` package

The controller layer of this application utilises a service layer - providing an extra layer of abstraction between the database and the controller. It also includes any logic required within the methods, for example, any backend-level error-checking before the object is passed to the database.

The `PointOfInterestService.java` class manages the data transferring between the controller layer, the model layer, and the database. Method such as `findAll()` and `delete(Long id)` simply make a method call to an identical repository method, that manages the SQL transaction. However, some other methods are slightly more complicated, the most prominent example being the `save(PointOfInterest poi)` method. The method first determines if a postcode or a coordinate pair was entered into the form. If a postcode was entered, then a method call must be made to fetch the coordinates

corresponding to that postcode. If not, there is no need to, however an error will occur if both a postcode and coordinates had been entered. The distance from the centre is also calculated and set in the object, before it being passed to the database.

The second service, `PostcodeService.java` manages parsing of postcodes, and includes methods to both validate a postcode, and get the coordinates that correspond to a postcode. This utilises the 'postcodes.io' API, an open-source, RESTful API that utilises open data primarily provided by the Ordnance Survey, the United Kingdom's national mapping agency [28]. Methods in this service are used to validate a postcode - ensuring that it is a real postcode and there is information available for it, as well as looking up a postcode and extracting the given latitude and longitude. This will not be an exact location, particularly if the user is marking a place on a residential street, for example. However, coordinates can still be added manually if more precision is required, and a postcode will still provide visitors an idea of where the point of interest is. Other APIs, such as the geocoding API included in Google Maps Platform, were considered, however further analysis of the pricing model saw that there was a risk of utilisation of such an API being costly.

2.3.2.2 Controllers

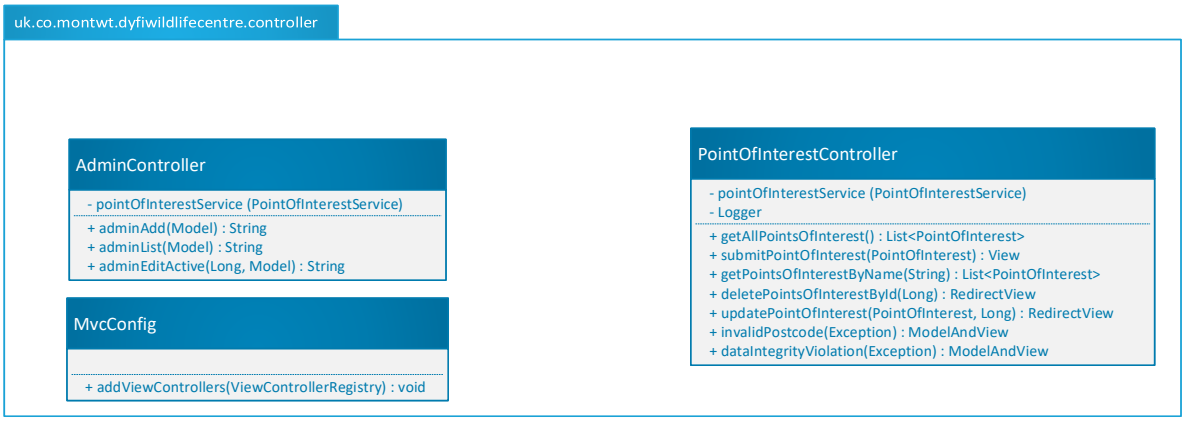


Figure 2.6: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.controller` package

There are three controllers within this layer of the application, each with three distinct responsibilities.

The `MvcConfig.java` package is designed such as to reduce the number of methods required for simple loading of a view, with other methods designed to either include a payload in its model or related directly to the API that the controller is implementing. The class has only one method, that declaratively adds a view controller with a URL path, along with the correct view name.

`AdminController.java` controls data that is being passed to the admin panel view models, with the methods within the controller either carrying objects within its model, or having a model passed to it via a query string that is appended to the URL. Continuing with

the abstraction granted through the service layer, the methods are short, simply adding attributes to the model for them to be handled in the view.

`PointOfInterestController.java` has a greater number of methods, that handle requests as a RESTful API. A brief specification is included in the table below.

Endpoint	Description	Return Type
/poi/get/id/:id	Returns the point of interest where :id corresponds to the ID of the point of interest	PointOfInterest
/poi	Returns all points of interest in the database	List<PointOfInterest>
/poi/form_create?poi=:poi	Saves a point of interest to the database where :poi corresponds to the point of interest to be saved	RedirectView
/poi/get/name/:name	Returns the points of interest where :name corresponds to the name of the point of interest	List<PointOfInterest>
/poi/delete?id=:id	Deletes the point of interest where :id corresponds to the ID of the point of interest	RedirectView
/poi/update?poi=:poi	Updates a point of interest where :poi corresponds to the point of interest to be updated	RedirectView

Table 2.2: API endpoints for /poi

Whilst RESTful values were adhered to as much as possible, HTML's incompatibility with request that are not GET or POST required methods such as delete to be HTTP GET request rather than HTTP DELETE. This will be further discussed in the Implementation chapter.

2.3.3 Security

Various information security considerations had to be taken into account whilst designing the application, the main one being authenticating and securing the administration panel. As the software is designed for use in a public location, an unauthenticated admin panel, where anybody who wants to could add or edit points of interest and users, was not acceptable. Therefore, a strategy for user authentication had to be put into place.

An example of a threat affecting the authentication of the application, once implemented, could be an SQL injection. A successful attack could expose user passwords and tamper with existing data stored in the database. If an attacker was able to gain access to the wireless network that the customer will be connecting the system to, they may be able to connect to the database remotely, resulting in physical limitations on access to hardware becoming futile. A suggestion could be made to the customer to secure their wireless network, perhaps having one SSID for public use and one for private use, however there is no way to enforce this in the application. With the application being hosted locally, it would be difficult to encrypt requests via HTTPS without creating a standalone server.

To combat these security concerns, Spring Security, an authentication framework designed for use with Spring, was implemented, through the contents of the security package. Users were stored in an SQL database, along with their roles. The default authentication method for users was plain-text, and users passwords could be easily found in the database. This was modified to use "bcrypt", a password hashing function based upon the Blowfish cipher [29]. bcrypt hashes the password that the user has been registered with, and then stores that hash in the password field of the relation. When a user logs in, the password that has been provided is matched against the hash, rather than the hash being decrypted, resulting in it not being reasonably possible to crack a password from the hash.

2.3.3.1 Model layer

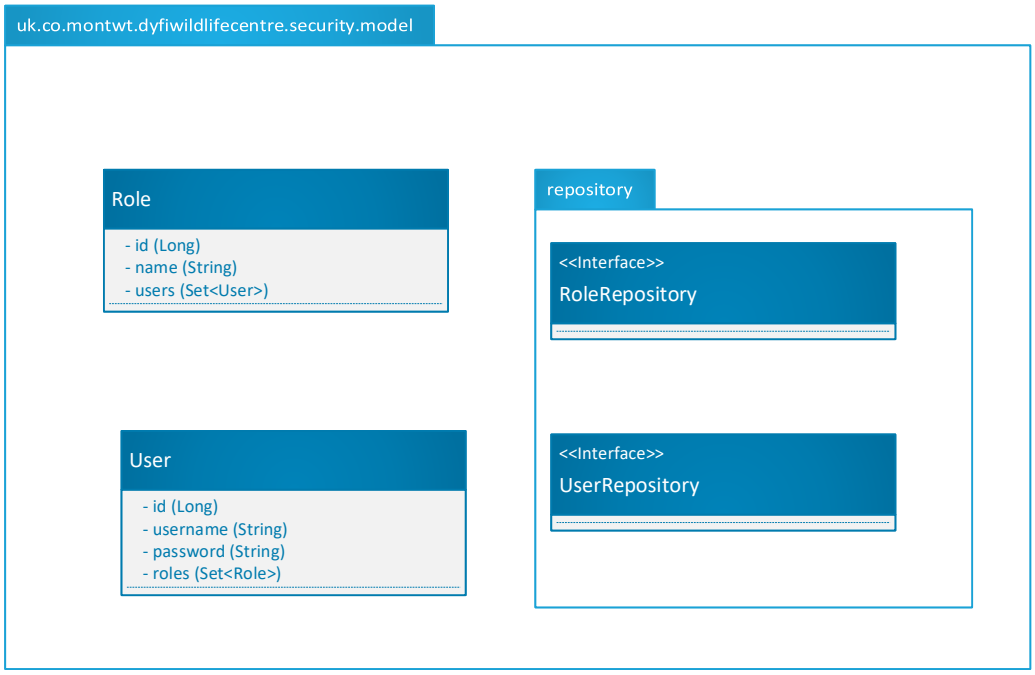


Figure 2.7: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.security.model` package

The model layer of the security package implements the database. There are no special methods and the purpose is simply to represent the database as Java objects.

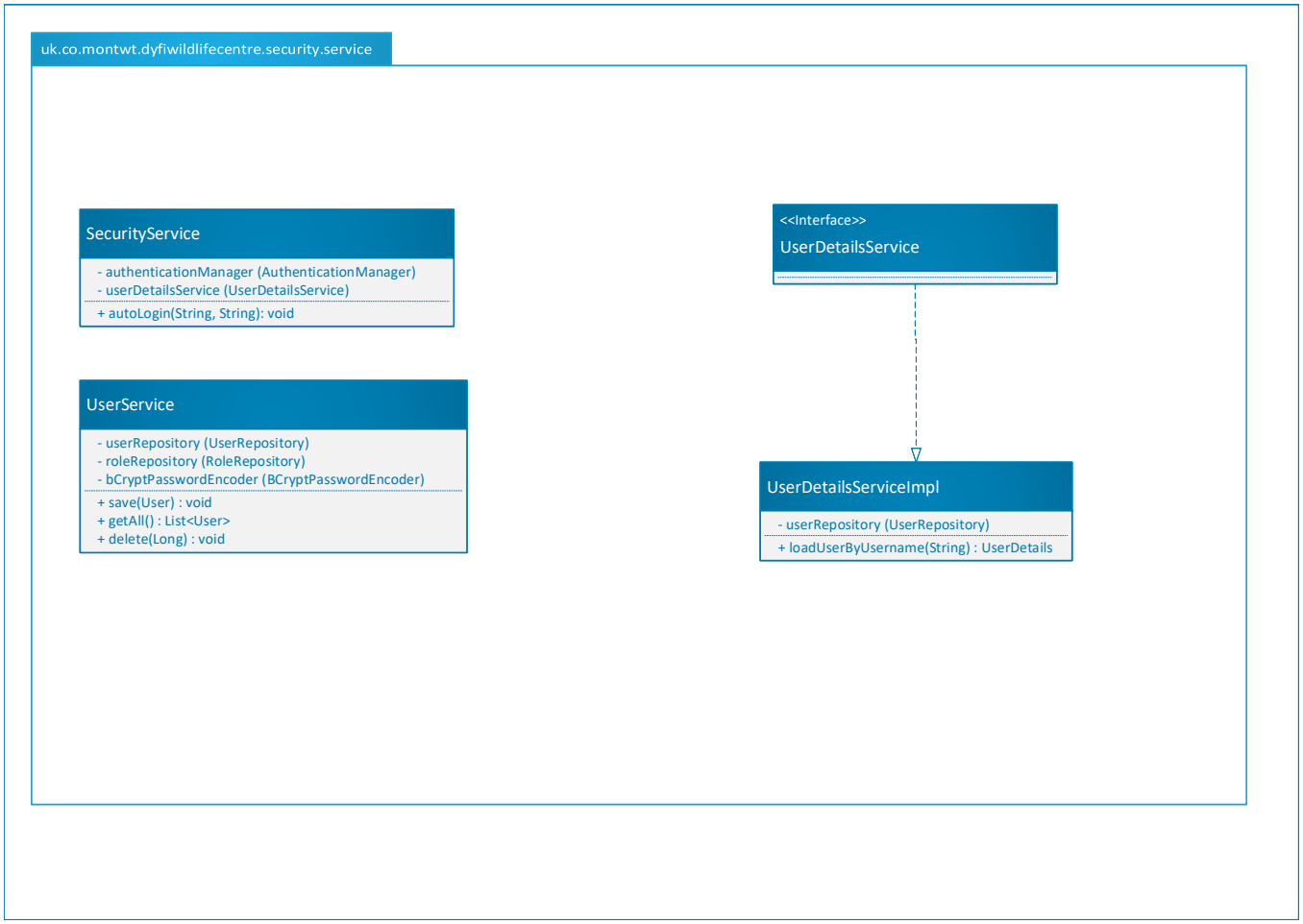


Figure 2.8: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.security.service` package

The security layer contains implementations for an interface from the Spring Security package, as well as instantiating Spring Security's authentication manager. In the `save(User user)` method in `UserService.java`, the BCrypt encoder is called, and the password is encrypted. As discussed above, the password is sent to the service in plain text, however this would be difficult to encrypt without HTTPS, and data transfer is being carried out solely on the local host due to the nature of the locally hosted database.

2.3.3.2 Controller layer

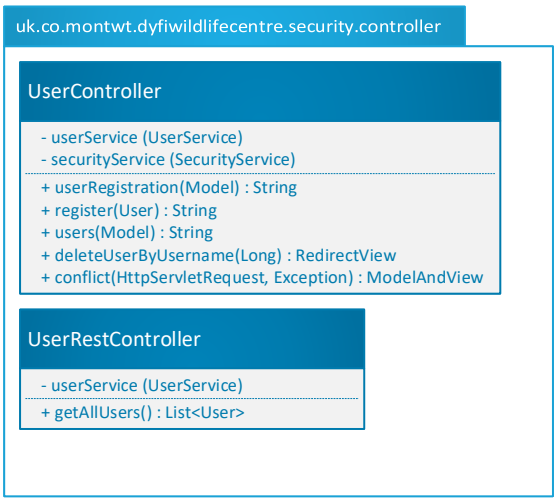


Figure 2.9: Class diagram of the `uk.co.montwt.dyfiwildlifecentre.security.controller` package

The controller layer of the security package includes a RESTful controller as well as a controller for the User object itself, that, as with the PointOfInterestController, interfaces with the service and the database. `UserRestController` implements a GET request, that returns all users, in order for them to be displayed in a list. `UserController` manages the registration and deletion of users, containing method calls to the relevant services. It also includes an exception handler, that reports a conflict message to the user, should they attempt to register a user with the same username as one that has already been registered.

2.4 Frontend

2.4.1 Design considerations

The frontend is a crucial part of this application, as this will be what the end user will be viewing. There were some key considerations that had to take place when making decisions regarding how to design the User Experience.

Familiarity was an important consideration to make, as users may not have a large amount of technical training, and a layout that they would be used to from different applications may be beneficial. This was part of the reasoning behind choosing Material Design as the design language for this software - with most Android, and some iOS apps, using Material Design, as well as popular web services such as Google Maps, there would not be a large number of potentially confusing prompts and dialogues.

An aspect of minimalism was required with the design of the software, particularly on the

homepage. The core function of the software was to show points of interest on a map, and having too many other UI components on the front page could potentially result in the page being too bloated. There was not too much of a need for a change from this in the admin panel, with clear instructions as to what the purpose of the view was.

Inspiration from the design came from similar web applications, which had a map as its main centrepiece. An example of a site assessed is the Airbnb accommodation-booking website [32]. This website uses a clear, relatively minimalistic, design. It is clear that, when searching for accommodation, its main element is a map, that includes markers and some options for filtering the map. This added to the strategy used within the frontend, and the inclusion of distinct, clickable markers.

Another website that was evaluated prior to design of the frontend was an element of Rightmove's website, a website intended to find property to buy or rent [33]. In the cited example, it was noted that an action bar took a small section of the top part of the page, with the map taking the majority of the page. Whilst markers differ in this implementation, with Materialize's modal design being used to hold a card as opposed to a sidebar being opened, inspiration from Rightmove was taken as to the prominence of the map and the lack of any distractions.

2.4.2 Prototyping

Prototyping and mockups were created as part of the planning and design process. These mockups were based on Semantic UI, which was changed to Materialize after the mockups were created. As there were quite a few similarities, it was decided to not redesign the mockups.

2.4.2.1 Home Screen

Figure 2.10: Home Screen mockup

The initial representation of the home screen showed a menu bar, at the top of the page, along with a hamburger menu. The main component of the home screen was the Google Maps component, with filter and reset buttons on either side of it. The general shape of this mockup was carried over to the final build, however a floating action button was used in place of two distinct buttons.

2.4.2.2 Marker information

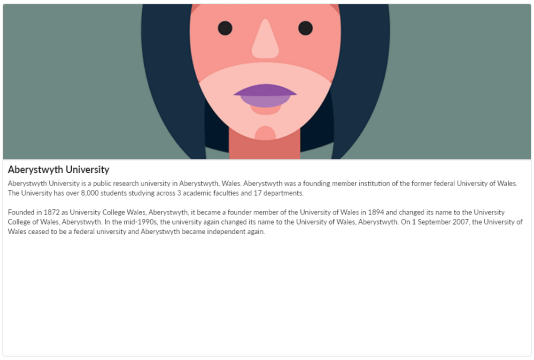


Figure 2.11: Mockup of the POI card

The ‘POI card‘ was designed such that it would contain easily-readable information, including a title, a description, and a relevant image. The POI card appears should a user click on a marker, with the JavaScript function for the Maps API being used to populate these fields. The scripts used to show the POI card are available in Appendix C.

2.4.2.3 Admin Panel - Login Screen

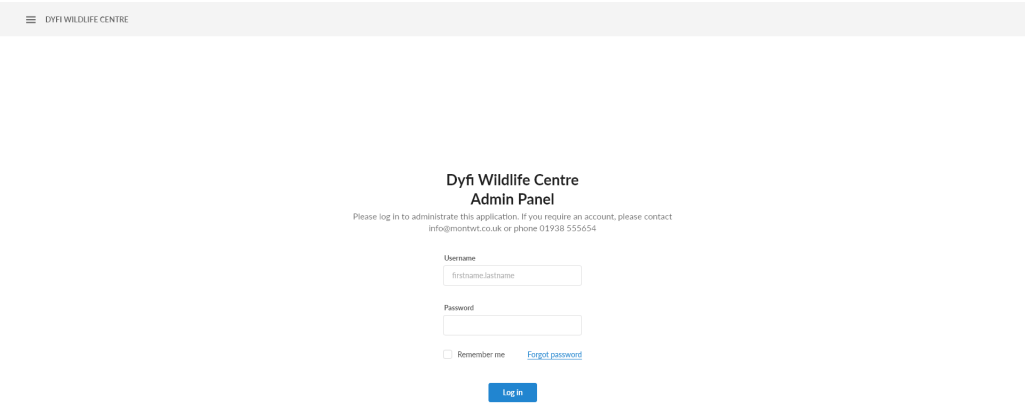


Figure 2.12: Mockup of the admin login screen

As explained above, the admin login screen was designed to be as minimalistic as possible. It simply includes a username and password, as well as a brief piece of information describing how one would go about requesting a login for the admin panel. Spring Security is used to facilitate the login.

2.4.2.4 Admin Panel - POI editing

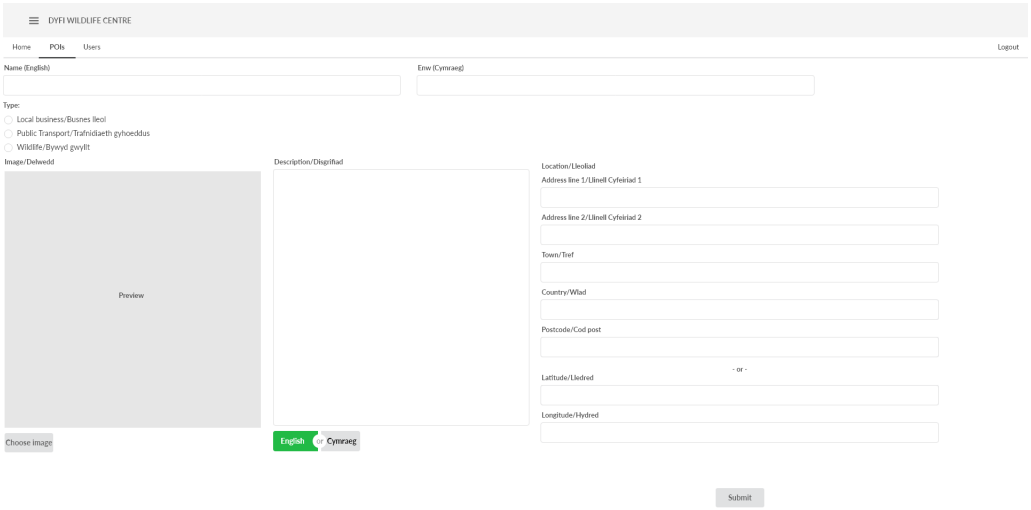


Figure 2.13: Mockup of the POI editing prompt

The initial logic behind the POI editing screen allowed for an entire address to be added, and a coordinate pair if it was deemed necessary by the user. Ultimately, this was considered an overcomplicated solution, and, as explained above, either a postcode or a coordinate pair were required.

2.4.2.5 Admin Panel - User management

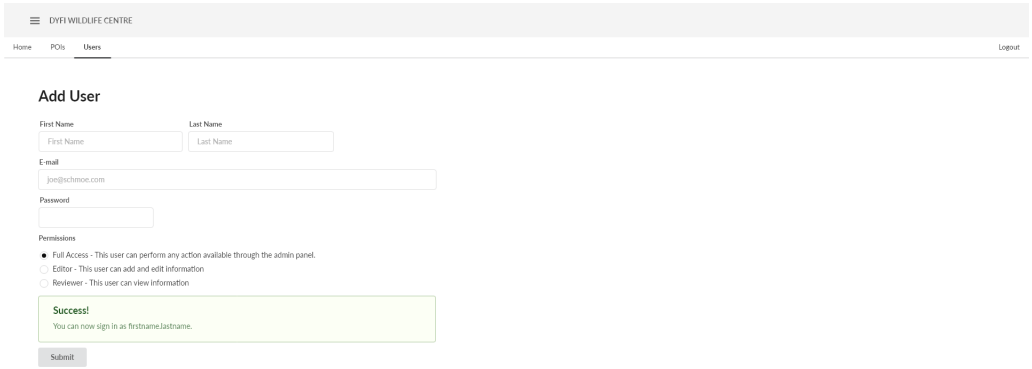


Figure 2.14: Mockup of the 'Add a user' screen

Similarly, adding a user with various details was later decided to be redundant, as a simple username and password would suffice for security purposes. Changes in decisions made will be discussed in Chapter 3.

Chapter 3

Implementation

The implementation of the design of this application followed an iterative methodology, with builds moving slowly towards the design that has been described in the previous chapter. This chapter will break the application's development into distinct iterations, with the initial iteration being what was considered the Minimum Viable Product - that being that the software had fulfilled the essential requirements and its core functionality had been implemented. Each iteration will include a subsection, that defines what was included in the design at that point. The final iteration will have included all facets of the design.

3.1 Iteration I: Minimum Viable Product

3.1.1 Design

The Minimum Viable Product for the purposes of this project was defined to be the successful completion of user stories 1 and 3 (see Appendix A). In this iteration, a user interface which includes a map, that has markers on it, is created, and the markers are able to be clicked on to show their name and their description. Two forms were also created in this iteration; one form to add a point of interest, and another to edit a given point of interest. Notably, no authentication or geocoding was present in this iteration, with any user being given the ability to add or edit a point of interest.

At this stage, data and backend layers for user authentication had not been implemented. The application consisted of two views - one view containing the homepage, and the other containing a form for adding and editing a point of interest. However, all elements of the technology stack were used to implement this iteration.

3.1.2 Google Maps for JavaScript API

The main part of this iteration was the implementation of the Google Maps for JavaScript API, and research into how this needs to be implemented. Google provides a large set of documentation detailing possible uses for the API, as well as a tutorial on how to use it.

One important part of the implementation of the API involves the use of an API key, for authentication. This API key is linked to an account holder, with any deductions over the monthly limit being automatically invoiced to the account holder on a monthly basis by Google. The issues caused by using a Google Cloud API key are, first and foremost, the lack of the ability to transfer the API key to a customer's account, as well as the complexity of the Google Cloud interface. There are many ways in which an account holder, who may have not used cloud-based deployment platforms before, could accidentally make large purchases. Powerful 'one-click' virtual machines can be deployed in a matter of seconds, but can be very costly if not shutdown.

To mitigate this issue, restrictions were placed on the API key, that only permitted its use for requests made through its JavaScript API, and there was no authorisation for transactions over the free monthly allowance. As discussed in section 1.1.2.2, there is little likelihood for requests made to the API to exceed this allowance, if used solely on this application. This also reduces the likelihood of attackers, who may have extracted the API key from the source code of the homepage, being able to carry out any meaningful damage, as the API key can be easily swapped with another account, if needs be.

The core of the API implementation consists of a callback function, `async function initMap()`, that is called asynchronously as the homepage loads. The code for this is stored in a separate file, `googlemaps.js`. The function utilises the API to instantiate a map, which is then populated with points of interest through an iterative for loop. The points of interest are found through use of JavaScript's Fetch API, an interface that allows for fetching of resources. An API call is made to a method which returns all points of interest that are currently in the database.

A problem faced when implementing this part of the application was the element of difficult whilst debugging parts of the JavaScript. Google Chrome and Mozilla Firefox's debugging tools were used, however it was often difficult to pinpoint the exact issue, due to the inability to debug the function calls from Google's API. Ensuring that the click listener was correctly appended to each marker also required some prior readings into utilising JavaScript for this. Ensuring that the code was clean was also a challenge during the refactoring, with attempts made to stop code leakage and ensure the procedural programming that was being used throughout the JavaScript element of this project remained.

3.1.3 Backend

The backend of the application retained the simple database layer at this iteration - an un-normalised table of points of interest. In order to ensure it interfaced correctly with the database, Java Persistence API annotations had to be used. `@Entity` is an example of annotation used on `PointOfInterest`, that declares the object as an entity that has an identical relation in the database. Spring's configuration properties also had to be edited,

to ensure the database that had been manually created is not dropped and replaced with Hibernate's impression of the database. However, constraints were also added directly to the object, via JPA annotations. One example is the `double latitude` member of the class. Two constraints were added to it, `@Min(value = -90, message = "Error: Latitude cannot be lower than -90")`, and `@Max(value = 90, message = "Error: Latitude cannot be greater than 90")`. These provide minimum and maximum constraints, and can pass a message if an exception is thrown, that can be shown at the view layer.

The class had to be implemented in such a way as it could be easily consumed by the data layer, without having to manually call the object's setters and passing each attribute of the form as a single attribute. The JPA allows for automatic setting of class members through either a constructor or setters. In this iteration, a constructor was used. However, setters and getters had also been created to improve the functionality of the code and assist in getting private members in other parts of the application; a unit test, for example.

In this iteration, the controller layer of the application interfaced directly with the database, as opposed to using the service layer to encapsulate the logic of the application. The application was relatively simple at the time, so it was decided that it wouldn't be necessary for an MVP to include that layer, as it would essentially be repeating repository methods without any further work carried out to the object. Spring's `@RestController` annotation, that provides controller functionality without methods being required to return a view, allowed for a RESTful API to be created without much more configuration than would be required if a standard controller was made. At that point, a `PageController`, carrying the `@Controller` annotation, had also been created to permit navigation between simple parts of the app, that did not require a model to be attached to it - the homepage, for example. In this iteration, the RESTful API created a method to list all points of interest, through utilising the `findAll()` method in JPA repositories, and add a point of interest to the database.

Difficulties were faced with creating a distinction between adding a point of interest and updating a point of interest. The `save()` method in JPA repositories will perform an SQL UPDATE, as opposed to an SQL INSERT, operation, if the primary key of the given object matches that of one object in the database. Relying on this, however, seemed to result in a conflict exception. A workaround was implemented - a second, distinct, update method and API mapping was created. This took two arguments, the Point of Interest with its updated members, and the ID of the point of interest. The ID would then have to be set within the method to allow for the SQL UPDATE operation to be triggered - with the conclusion being that the ID of any point of interest is lost within its transmission from the view layer to the controller layer.

3.1.4 Frontend

A number of challenges were posed when scripting the frontend of the application, including following Materialize's standards for layout development, and utilising Thymeleaf correctly and efficiently.

Thymeleaf's implementation in Spring focuses on a "resources" folder, with two subfolders,

“static” and “template”. The first folder, as the name suggests, contains static resources - resources that Thymeleaf and Spring would not change at compile time. This included `materialize.css`, the CSS file that provides implementation of each Materialize UI components, with the JavaScript containing Materialize’s JavaScript helper functions, the Google Maps function discussed above, as well as a third-party algorithm, “lazysizes” [30]. This implements the lazy-loading algorithm on images that are assigned this class; the algorithm ensuring that the resource attached to it is only loaded when required. In this instance, the algorithm was applied to images in the marker popup, to ensure they were not loaded before required, which would impact initial page load times.

The second folder contains the HTML for pages that are to be rendered onto the browser. Thymeleaf expects HTMl to be typed declaratively, as opposed to approaches frameworks such as React take. A feature of Thymeleaf, that was researched and used during this iteration, is its fragments feature. This feature allows you to place reused code into its own file, and called at the position that it would be written in a static HTML website. For example, calls to files such as Material Icons were placed in `fragments/header.html`, and, in the `<head>` tags of all templates, it was called with the line `<fragment th:replace =ẗfragments/header :: header>̈</fragment>`. When this was rendered in the browser, the source would show what was in the fragment. This allowed for less reused code, and easier navigability within the resources files.

Correct placement of the map seemed to pose an issue when creating the homepage. The container for the Maps JavaScript appeared to overflow the page’s default view height, requiring a user to scroll down to be able to view it. This was not a reasonable expectation for the user, particularly as it was intended to be used in a web browser set to full-screen. A workaround for this was implemented, by creating a custom CSS file. This CSS file overrode Materialize’s grid system by having the height be declared manually, with the line being `height: calc(99vh - 56px);`. The height of the menu bar, at the top of the screen, measured at fifty-six pixels, according to the CSS file. With a view height of 100 still appearing to overflow the page, decrementing this by one appeared to resolve the issue, and showed an aesthetically pleasing home screen.

The ‘admin panel’, at this stage, had two forms, one to add a point of interest, and one to edit a point of interest. Research had to take place in how to correctly manage a POST request within an HTML form when using Thymeleaf. The Thymeleaf XML namespace allowed you to call the backend from the view. Therefore, in the ’add’ form, Thymeleaf calls the HTTP POST method in the controller, and instantiates an empty POI with `th:object="${pointOfInterest}"`. Each field within the HTML form also included a Thymeleaf attribute, `th:field`, that included the member of the POI object that the field relates to. This was similar in the edit form, with a duplicate form being used that is instead populated with members from the point of interest provided in the model of the view.

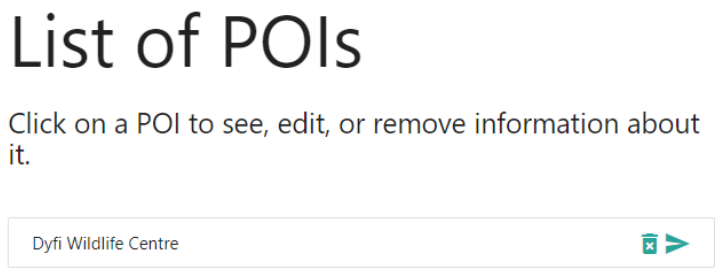


Figure 3.1: An example of a point of interest list in the view layer.

A list of points of interests also had to be made, which included edit and delete buttons for every point of interest. This was created through performing iteration within Thymeleaf, with a method call to the GET request that returns all points of interest in the database. This is called within a list object in the view as `th:each="poi : $pointsOfInterest"`, and is populated at runtime, with the rendered view containing every point of interest. This allows for simplicity to be achieved in the view, and negates the need for JavaScript, as an example, to be used, reducing the amount of code within one view.

3.2 Iteration II: Admin authentication

3.2.1 Design

The second iteration of the application focused on implementing authentication for the admin panel. The previous iteration did not have any authentication - this would cause issues at production due to the web app being used in a public location. Therefore, this iteration fulfills story 2, and explored different authentication methods for only part of an application, rather than the entire application.

This stage implemented most of the backend and data layers described in chapter 2, however filtering and geocoding had still not been implemented at this stage. However, further parts of the frontend were implemented, with options to add and delete users from the admin panel.

3.2.2 Spring Security

This was the first iteration to take advantage of Spring Security, a security add-on that interfaces with the Spring Framework, as discussed in chapter 2. The main challenge posed when utilising Spring Security was ensuring that it linked correctly with the database layer. By default, Spring Security worked off of the basis of an in-built User object in the `org.springframework.security.core.userdetails` package. While this implementation included a username and password; the required fields if keeping to the design standard for this iteration, there was little way to customise its behaviour as well as use

a custom `UserDetails` implementation. This facilitated the creation of a custom User and Role class, along with `UserDetailsServiceImpl`, a class that implements Spring `UserDetailsService` interface.

Another challenge when working with Spring Security was implementing an acceptable storage method for sensitive personal data. After reviewing the customer's requirements, it was decided that only a username and an encrypted password should be stored. This fit with the customer's requirements, with the main aim of authentication being that the add/edit forms are inaccessible to the general public. It also stops any legal concerns that could arise from the storage of personal data such as full names, e-mail addresses and telephone numbers, a data set that could arguably be legislated by data protection laws in the UK [31].

It would have not been acceptable to store passwords as plain text. Spring Security accepts plain-text passwords by default, and an algorithm has to be implemented to encrypt passwords. As discussed in section 2.3.3, classes for a one-way hashing algorithm needed to be explored, and one was picked that had been proven to be reasonably difficult to crack.

Unpredictable behaviour occurred when a user was registered with the same name as another user in the database. As discussed in chapter 2, there was more efficiency using a numerical sequence as a primary key, then there would have been if a string was used, as may had been the case in many common implementations of a user relation. Without any constraints or exception handling, two users were able to be created with the same username. Exceptions were thrown at the Spring layer should you attempt to login with a duplicate username. To work around this issue, a unique constraint was added to the database for the username column. This was matched in the controller classes of Spring Security to an exception handler. The exception handler catches `DataIntegrityViolation.class`, an exception that is thrown when a conflict is detected in the database. The exception is handled by display an Error 400 page, with details of what occurred available to the user, and the opportunity to try again with a different username is given.

Initial setup of Spring Security appeared to have it default to requesting a login for access to any part of the web app, including the map. This would've defeated its purpose, given a member of the public could then access the admin panel regardless. This was modified through the use of a class that extends Spring's `WebSecurityConfigurerAdapter`, in its `configure(HttpSecurity http)` method. The source code of this method is available in Appendix C, section 3.4. The use of matchers enabled all users to access the index page, that contains the map, and permitted that map to access any required resources. The line `.antMatchers("/admin/**").hasRole("ADMIN")` ensured that only registered users; automatically given the role of admin, are able to access the admin panel. The security configuration also enabled CSRF tokens. These tokens reduced the threat of cross-site request forgery, by appending tokens to authorised HTTP requests and disallowing requests that failed to include a valid token.

3.2.3 Backend

Modifications had to be made to the backend to accommodate for Spring Security, as well as management of routing between various parts of the admin panel.

Similar to points of interest, the admin panel now included a form to register a new user, as well as a form to list all users and delete ones as appropriate. This required the creation of controllers for both the admin side of the application (that facilitated requests to the authenticated `/admin` URI), and an API to control requests made to the user relation in the database. Similar definitions to the point of interest API were created, including returning a list of all users in the database, and returning a form to add details to an empty user object. The user service encoded the password, received in plain text by the POST request, before storing it in the database.

Issues with infinite recursion occurred when a request was made to list all users. The User object contains a many-to-many relationship with a Role, allowing one user to be assigned more than one role. Whilst this functionality is not being used in this application, it was decided to maintain this relationship, in case further iterations past the first production build introduced different roles. The stack trace of the stack overflow error thrown when calling this request revealed that the error was due to how the Jackson JSON parsing API, included within Spring, was processing many-to-many relationships, adding the same role infinitely to the end of the JSON string of each user. This was resolved by appending the `@JsonIgnore` annotation on all members in the user field, apart from ID and username. The password was encrypted, though there was a marginal risk that the encrypted password could be used to access admin services through other means than an HTTP request. However, neither that or user roles had to be exposed outside of the authenticated areas that had been dictated by Spring security configuration.

3.2.4 Frontend

New pages had to be created, again to accommodate Spring Security. A similar strategy to points of interest was taken to the creation of these pages, including a page to register a user, and a page to list all users. Similar Thymeleaf methods for iteration and field injection were also utilised, with the registration form consuming the HTTP POST request to register a new user.

As explained in the discussion on Spring Security, exceptions had to be handled for duplicate usernames, as well as any other conflict exceptions. An error page was made to handle this, utilising Thymeleaf field injection methods to provide the user with an exception name and message. This can be used by the user to see if they had made any obvious mistakes when entering information (a duplicate username, for example), or could be passed to a support technician for debugging in any future releases.

The layout of these forms re-used concepts from the points of interest pages, with an emphasis on minimalism and ease-of-use. It was, however, decided not to include options for editing users - this was considered relatively futile due to the lack of information required to register a user, and there are seldom cases were only a user's username has to be changed. Some password managers, such as Google Password Manager, appeared to

automatically fill the registration field with a saved username and password. It was difficult to decide on a way to stop this from happening, however, it is clear that the registration form registers a new user rather than logs a user in.

3.3 Iteration III: Geocoding

3.3.1 Design

The third iteration introduces geocoding to the software. Geocoding allows for an address, or part of an address, to be converted into a coordinate pair. Evaluation of different geocoding APIs is discussed in section 2.3.2.1, with an open-source API, Postcodes.io, used to geocode UK postcodes.

The customer for this web app is based in the UK, and they had mentioned that points of interest are likely to be local. Therefore, it was decided that a geocoding API that focuses solely on postcodes would be sufficient enough for the purposes of this application. While issues may arise with a postcode covering more than one location, a row of houses on a street, for example, it was deemed unlikely that points of interest would be in such places, particularly considering that the nature reserve in located in a rural area of Wales, where large, 'catch-all', postcodes are less likely than in urban areas.

This iteration introduces the postcode column to the point of interest relation, with constraints as discussed in chapter 2 being implemented. It also introduces classes to handle interfacing with the Postcodes.io API.

3.3.2 Postcodes.io API

The Postcodes.io API is an open-source interface with data from the Royal Mail and the Office of National Statistics, that provides various tools to work with the British postcode system. In this application, two of the tools were used, postcode-to-coordinate conversion, and the postcode validation tool.

Postcode-to-coordinate conversion allowed for a postcode to be queried in the API. The API returns a JSON string, containing characteristics specific to the postcode. An example of a postcode query is given below.

```
{
    "status": 200,
    "result": [
        {
            "postcode": "SY23 3DB",
            "quality": 1,
            "eastings": 259560,
            "northings": 281846,
            "country": "Wales",
            "nhs_ha": "Hywel Dda University Health Board",
            "longitude": -4.066423,
            "latitude": 52.416527,
            "european_electoral_region": "Wales",
            "primary_care_trust": "Hywel Dda University Health Board",
            "region": null,
            "lsoa": "Ceredigion 002A",
            "msoa": "Ceredigion 002",
            "incode": "3DB",
            "outcode": "SY23",
            "parliamentary_constituency": "Ceredigion",
            "admin_district": "Ceredigion",
            "parish": "Aberystwyth",
            "admin_county": null,
            "admin_ward": "Aberystwyth Bronglais",
            "ced": null,
            "ccg": "Hywel Dda University Health Board",
            "nuts": "South West Wales",
            "codes": {
                "admin_district": "W06000008",
                "admin_county": "W99999999",
                "admin_ward": "W05000362",
                "parish": "W04000359",
                "parliamentary_constituency": "W07000064",
                "ccg": "W11000025",
                "ccg_id": "7A2",
                "ced": "W99999999",
                "nuts": "UKL14"
            }
        }
    ]
}
```


Figure 3.2: An example of a result when querying the postcode SY23 3DB

As evidenced by the query, a large amount of information is returned, much of it not relevant for the purposes of this application. However, key-value pairs of longitude and latitude are of course of use in geocoding. Future iterations could perhaps benefit from other char-

acteristics that can be derived from the postcode; an example perhaps being the ”parish” key - this would contain the location that the postcode is in, and could be added to the information window.

Postcode validation is used to check that information can be derived from the string passed to the request, and returns a boolean confirming if it is valid or not. As discussed in chapter 2, this is used in place of a complex regular expression to ensure that edge cases; postcodes that are not valid however match the regular expression, are accounted for, and there is not a possibility of null pointers. Postcode validation also accounts for terminated postcodes, that at one point were valid however were changed, and may still be matched by a regular expression.

Consumption of the Postcodes.io API takes place in static methods within the `PostcodeService.java` class. An open-source Java library which contains methods that interface with this API is available, however, the requests made in this application are simple, therefore it was decided to use built-in Java and Spring libraries to avoid unnecessary addition of third-party libraries, that have potential to be unreliable or no longer be being updated by its authors. This class contains two static methods; one that checks if a postcode is valid and one that returns the coordinates of a given postcode, with two helper methods that encode a postcode as a UTF-8 string and returns the result of an HTTP request to the API. The `javax.net` library, which provides classes for networking, was used to return results from querying the API. The Jackson JSON processor; included by default with Spring, is used to handle the query and return the correct key-value pair. Another class used within this package is `Point2D.Double`, which allows for an X/Y coordinate pair to be passed in the same object.

3.3.3 Backend

Changes to the model and controller layers of the backend had to be made to facilitate postcode validation and entry. A nullable String object to hold postcodes was introduced as a member of the PointOfInterest class, with the latitude and longitude members of the class still being editable, should a user wish to enter a more precise location for a point of interest.

An issue arose around the inclusion of fields for both a postcode and coordinates. Without any changes being made, a user could potentially add both a postcode *and* coordinates. The user-entered coordinates would override that of the postcode, rendering entry of the postcode futile. Therefore, the save method in `PointOfInterestService` was edited such that it would perform error-checking before geocoding the postcode and adding it to the database.

The error-checking strategy focused on both the latitude and longitude fields, and the postcode field. Should the latitude and longitude both be zero, a decision discussed in Chapter 2, the application would check if a postcode was entered, and then call the static service method to geocode the postcode, setting the coordinates through a helper method in the PointOfInterest class that assigns the relevant members of the returned Point2D.Double object to the latitude and the longitude. If no location was entered, or only one half of a coordinate pair was entered, this would violate the “not null island” constraints, and a data

integrity violation would be thrown. Conversely, a `PostcodeException`, a new subclass of `RuntimeException`, would be thrown, with the message that both a postcode and a set of coordinates were entered. `PostcodeException` is also thrown should an invalid postcode be entered.

3.3.4 Frontend

Changes to the forms for adding a point of interest, as well as exception handling templates, had to be designed in order to facilitate postcode entry. This took the shape of a "location" section of the form, where it was specified that either a postcode or a pair of coordinates had to be entered, otherwise the aforementioned errors would occur.

Error handling took the same approach as previous iterations, with seldom changes required due to Thymeleaf's ability to inject the type of exception and the message that the exception carries into the template.

3.4 Iteration IV: Map additions and final build

The final iteration, and the submitted build, of the application, generally fulfilled the tasks marked as "Could have" - tasks that were not particularly necessary to include in the application in order for it to work well, however are useful additions, either for aesthetic reasons or conveniences.

Much of the work that took place in this iteration was on the Google Maps script. Third-party JavaScript libraries were utilised to implement marker clustering, that allowed for better visibility of markers. Filtering was also implemented, with changes to all parts of the stack to facilitate this.

A number of non-essential requirements were not able to be implemented in this final iteration. This will be briefly discussed in this section, and expanded upon in Chapter 5.

3.4.1 Design

This iteration completed the final build and implemented the design as described in Chapter 2. Differences to the previous iteration included the addition of a category column and member in the database and model layers respectively. Controller and service methods had to be updated to accommodate for this.

3.4.2 Google Maps

`googlemaps.js`, the implementation of the Google Maps API for this application, was edited to include the MarkerClustererPlus library. This is parts of Google's Maps Utility Library, and is accessed via a Content Delivery Network. Theoretically, it was possible

to implement this library locally, however Google's recommended method of installation was via npm, a JavaScript package manager. As the JavaScript file was using a vanilla implementation of JavaScript, rather than taking a modular approach, common with ECMAScript 6 applications, it was difficult to download and include locally without major edits having to be made.

Marker clustering was relatively simple to implement. A group of marker images for various zoom levels was downloaded from Google's tutorial on the library, and the marker clusterer was instantiated at the beginning of the script. Markers were added to the cluster within the for-each loop that added information to each point of interest's information window.

The strategy for filtering markers involved creating a new function, and changing the marker and cluster arrays to global variables. When the user clicked on a filter, a function with the given category was called, and an iterative loop would be instigated. The loop would create a new array of markers, through checking each marker's category property within the array and setting the marker's visibility dependent on it matching the category given. User-level testing of these methods was successful, with options also added to clear any filtering and show all markers again. A function for a 'reset' button was also created. When the reset button was clicked, the map would pan back to the original location; centring on the Dyfi Wildlife Centre. Again, user testing proved that this works.

3.4.3 Backend

Changes had to be made to the backend to accommodate for a category to be added to each marker. Initially, this was to take the form of three booleans, `isWildlife`, `isBusiness`, and `isTransport`. Difficulties with this approach appeared centred on difficulties with Thymeleaf passing booleans from the view layer to the model; appearing to either return a string containing the value or not being able to discern the type of the field. An incompatibility between Thymeleaf and Materialize was also discovered - this is expanded upon further within the frontend subsection.

The workaround solution was to have the category hold a string, with a maximum length of the longest category. This was not a perfect solution, as it could have potentially allowed for any string to be entered if the HTTP request was edited in any way other than through the form. However, this was deemed a low risk - any object with a category that is not wildlife, business, or transport would appear if the filtering was set to all. It was already possible for a point of interest to have no category; this was a deliberate design to account for any point of interest that didn't fit with any of the defined categories. The time that would have been required to research an alternative solution outweighed the potential benefits of a more robust solution for filtering and categorisation.

3.4.4 Frontend

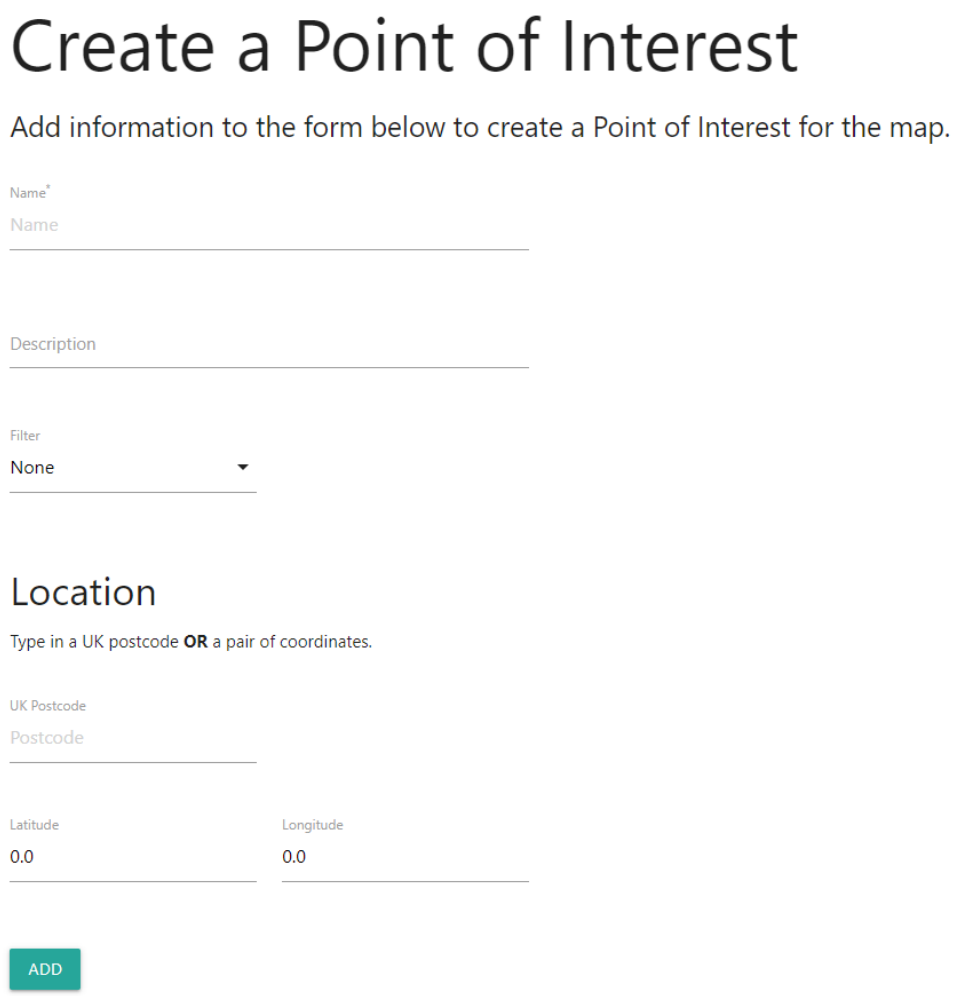


Figure 3.3: The 'Add a POI' form that is present in the final build of the application.

Initially, errors were thrown when filtering was introduced to the frontend. The initial solution to modification of the add and edit point of interest forms was to include three checkboxes, one for each possible category. With the preliminary design for categorisation focusing upon the addition of three booleans to the PointOfInterest object, each checkbox would be able to toggle between true and false, and be labelled with the category.

However, when implementing this solution, Thymeleaf had modified the behaviour of Materialize checkboxes, resulting in them not being rendered in the view. Certain workaround were explored, including adding custom CSS to the `materialize.css` file, however this did not work, and it was difficult to change this facet without breaking the CSS file in its entirety. Therefore, an alternative solution was proposed, involving a String object, where the form passes the text of the category selected in a dropdown menu. This interfaced well with the backend, and remained as the final implementation.

Figure 3.4: The front page that is present in the final build of the application.

Changes to the index page were also made to allow for filters to be selected, and for the reset button to be selected. This made use of Materialize's Floating Action Button component - a button that overlayed the map and was fixed at the bottom right of the browser's viewport. This button expanded to include five other buttons - four of these were filters, with a relevant icon, and the fifth was a reset button, which reset any movement to what the application was at the beginning. Whilst refreshing the page would have achieved the same result, given the client-side behaviour of JavaScript, the transition looks more aesthetically pleasing through a simple pan of the map.

3.4.5 Requirements not implemented

The main requirements that were not implemented were the inclusion of internationalisation for Welsh speakers, and the inclusion of a mechanism to upload images.

During the planning stages of this project, it was proposed that Spring's I18N internationalisation libraries would be used to include a version of the web application that was in English, and one that was in Welsh. The location of the Dyfi Wildlife Centre was in a rural part of the country, where Welsh is more widely-spoken, and would have more readily adhered to legislation surrounding bilingual services in Wales. However, this appeared to be more difficult to implement than was originally thought. An unfamiliarity with the Welsh language would have required translations of each part of the application to be provided by an external translation service, or a fluent Welsh speaker, which could have taken a considerable length of time. It was also noted that the description and names of points of interest could easily be provided in both English and Welsh, without any considerable complexities.

Uploading of custom images was also attempted, however, similarly, it appeared to be more complex than originally thought. There were difficulties with provision of a persistent storage location for images. It was briefly considered to allow hotlinking of URLs from

other locations, however it was unlikely that most of the images that users would want to add to points of interest would have not been photographed by themselves. It was decided to keep a placeholder image with each point of interest, and ensure that textual information was as visible as possible.

Chapter 4

Testing

4.1 Automated Testing

4.1.1 Backend

4.1.2 Frontend

4.2 User Testing

4.2.1 Feedback

4.3 Test Tables

Chapter 5

Evaluation

Bibliography

[1] C. d. S. Antonio, *Testing React Components*. Berkeley, CA: Apress, 2015. [Online]. Available: https://doi.org/10.1007/978-1-4842-1260-8_9

This is a book discussing approaches to testing React components, and different testing tools, including Jest.

[2] I. Chaniotis, K.-I. Kyriakou, and N. Tselikas, "Is Node.js a viable option for building modern web applications? A performance evaluation study," *Computing*, vol. 97, no. 10, pp. 1023–1044, 2015.

This article examines Node.js in terms of resource efficiency, and appears to conclude that it is a good tool for developing fast web applications.

[3] E. You, "Comparison with other Frameworks - Vue.js," https://vuejs.org/v2/guide/comparison.html, Accessed: 2020-05-05.

This web page makes comparisons between Vue.js and other frameworks, including React, noting its differences in HTML modelling.

[4] Django Software Foundation, "Design Philosophies | Django Documentation | Django," https://docs.djangoproject.com/en/3.0/misc/design-philosophies/, Accessed: 2020-05-05.

This web page describes the philosophies behind the design of Django, including its adherence to loose coupling and tight cohesion.

[5] A. Leff and J. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, vol. 2001-, no. January. IEEE, 2001, pp. 118–127.

This article provides an indepth explanation and analysis of the MVC design pattern

[6] Oracle Corporation, "Java Platform, Enterprise Edition (Java EE) | Oracle Technology Network | Oracle," https://www.oracle.com/java/technologies/java-ee-glance.html, Accessed: 2020-05-05.

A description of the purposes of the Java Enterprise Edition standard.

[7] D. Fernandez, "Thymeleaf," https://www.thymeleaf.org/, Accessed: 2020-05-05.

Vendor description of Thymeleaf, a common view technology for Spring

[8] D. Cochran, *Twitter Bootstrap web development how-to a hands-on introduction to building websites with Twitter Bootstrap's powerful front-end development framework*. Birmingham, UK: Packt Pub., 2012, 1-283-96085-0.

A tutorial book on utilising Bootstrap, that discusses its use cases.

[9] Various, "Fomantic-UI," https://fomantic-ui.com/, Accessed: 2020-05-05.

A community fork of Semantic UI, a CSS framework that uses natural language syntax to provide different web components.

[10] A. Wang, A. Chang, et al., "Documentation - Materialize," https://materializecss.com/, Accessed: 2020-05-05.

A front-end framework based on Material Design.

[11] Y. Cheng, K. Zhou, and J. Wang, "Performance Analysis of PostgreSQL and MongoDB Databases for Unstructured Data," *Proceedings of the 2019 International Conference on Mathematics, Big Data Analysis and Simulation and Modelling (MBDASM 2019)*, 2019.

This is an article that discussed the perfomance of PostgreSQL and MongoDB. It came to the conclusion that the smaller data sets that this application would see are better suited to PostgreSQL.

[12] VMWare, Inc., "Spring Tools," https://spring.io/tools, Accessed: 2020-05-05.

This is a website providing information about Spring Tools, an addon to IDEs to work with the Spring Framework.

[13] PostgreSQL Global Development Group, "PostgreSQL Clients," https://wiki.postgresql.org/wiki/PostgreSQL_Clients, Accessed: 2020-05-05.

A website providing information about the range of tools available for work with PostgreSQL.

[14] JetBrains, Inc., "Free Educational Licenses - Community Support," https://www.jetbrains.com/community/education/#students, Accessed: 2020-05-05.

A website detailing the free educational licenses that students can use for JetBrains products.

[15] Mozilla Foundation, "Firefox Developer Edition," https://www.mozilla.org/en-GB/firefox/developer/, Accessed: 2020-05-05.

A website detailing features offered by the web browser Mozilla Firefox Developer Edition.

[16] E. Evans, "84 inch touchscreen Poole Harbour Paul," https://www.youtube.com/watch?v=XYIwcAfgFkA, July 2018, Accessed: 2020-05-05.

A video showing details of a touchscreen application for the Poole Harbour nature reserve.

[17] OpenStreetMap Foundation, "Openstreetmap," https://www.openstreetmap.org/about, Accessed: 2020-05-05.

A website detailing information about OpenStreetMap, an open source geodata repository.

[18] V. Agafonkin, "Leaflet - a JavaScript library for interactive maps," https://leafletjs.com/, Accessed: 2020-05-05.

A JavaScript API for OpenStreetMap.

[19] "Maps - Mapbox," https://www.mapbox.com/maps/, Accessed: 2020-05-05.

A paid-for implementation of a mapping API, using OpenSteetMap.

[20] Google LLC, "Maps JavaScript API," https://developers.google.com/maps/documentation/javascript/, Accessed: 2020-05-05.

Documentation regarding an implementation of the Google Maps' API for JavaScript.

[21] ——, "Pricing & Plans - Google Maps Platform," https://cloud.google.com/maps-platform/pricing, Accessed: 2020-05-05.

Details pertaining to the pricing plan for Google Cloud Platform, specifically its Maps component.

[22] GitHub, Inc., "GitHub," https://github.com/, Accessed: 2020-05-05.

A repository hosting service using the Git Version Control System, that has been utilised for all work on this project.

[23] A. Nystrom, "Defining and evaluating an agile software development process for a single software developer," Master's thesis, Chalmers University of Technology, 2011. [Online]. Available: https://hdl.handle.net/20.500.12380/147143

A study into different types of Agile Development for single developers.

[24] R. Agarwal and D. Umphress, "Extreme programming for a single person team," in *Proceedings of the 46th Annual Southeast Regional Conference on XX - ACM-SE 46*. ACM Press, 2008. [Online]. Available: https://doi.org/10.1145/1593105.1593127

A study on how Extreme Programming could be adapted to work with a single developer.

[25] G. Gavett, "You versus the clock: testing the latest time-management advice.(pick three: You can have it all (just not every day))(the pomodoro technique: The acclaimed time-management system that has transformed how we work)(make time: How to focus on what matters every day)(hyperfocus: How to be more productive in a world of distraction)(off the clock: Feel less busy while getting more done)(book review)," *Harvard Business Review*, vol. 96, no. 5, p. 150, 2018.

A study into the Pomodoro Technique, that was used to perform an element of timeboxing when working on the software.

[26] Travis CI, GMBH, "Travis CI - Test and Deploy your code with confidence," https://travis-ci.org/getting_started, Accessed: 2020-05-05.

A continuous integration tool with direct compatability with GitHub, that is used to ensure that testing is carried out at every build.

[27] IDDQD Limited, "Postcode Validation," https://ideal-postcodes.co.uk/guides/postcode-validation, Accessed: 2020-05-05.

A website containing details pertaining to postcode validation.

[28] ——, "Postcodes.io," https://postcodes.io, Accessed: 2020-05-06.

A RESTful API containing methods to perform geolocation on UK postcodes.

[29] N. Provos and D. Mazières, "A future-adaptable password scheme," in *USENIX Annual Technical Conference, FREENIX Track*, 1999. [Online]. Available: http://www.usenix.org/events/usenix99/provos.html

A paper that defined the bcrypt hashing method.

[30] A. Farkas, "lazysizes," https://github.com/aFarkas/lazysizes, Accessed: 2020-05-08.

Code for a 'lazy-loading' algorithm for images, that defers loading of images until their container is opened.

[31] Information Commissioner's Office, "Guide to Data Protection," https://ico.org.uk/for-organisations/guide-to-data-protection/, Accessed: 2020-05-09.

A guide to data protection regulations in the United Kingdom.

[32] AirBnb, Inc., "Paris - Stays - Airbnb," https://www.airbnb.co.uk/s/Paris/homes?source=mc_search_bar&search_type=section_navigation&refinement_paths%5B%5D=%2Fhomes&federated_search_session_id=eaf0e317-cfac-4dc8-9792-9230d81d877d, Accessed: 2020-05-09.

An accommodation website that includes a mapping API.

[33] Rightmove Group Limited, "Properties to Rent in Aberystwyth - Rightmove," https://www.rightmove.co.uk/property-to-rent/map.html?locationIdentifier=REGION%5E11&numberOfPropertiesPerPage=499&propertyTypes=&includeLetAgreed=false&viewType=MAP&mustHave=&dontShow=&furnishTypes=&viewport=-4.14013%2C-4.05773%2C52.4015%2C52.4257&keywords=, Accessed: 2020-05-09.

A property website that includes a large map tool.

Appendix A

User Stories

1.1 Purpose of this file

This file is intended to provide user stories as part of a modified XP approach to this project. It will be modified into tasks and sub-tasks that will be created through GitHub's issue tracker.

1.2 System metaphor

A web application that uses a map to show interesting things around the Dyfi Wildlife Centre

1.3 Epics

1. As a volunteer, I want to have a web interface where I can show people information about the nature reserve on a map.
2. As an administrator, I want to be able to add and edit information about the nature reserve, and change anything I want to change.

1.4 Priorities

1.4.1 Legend

Using MoSCoW analysis.

- **M: Must** - Non-negotiable, must be satisfied for the project to be considered a success.

- **S: Should have** - High priority, a critcal requirement that should be included if at all possible. However, it can be implemented in other ways if necessary.
- **C: Could have** - A desirable requirement but not necessary, will be included if there's enough time.
- **W: Won't** - A requirement identified as not necessary to be implemented at this stage in planning, but may be considered during future stages.

Each story in the following section is marked with it's priority by a letter corresponding to each priority.

1.5 Stories

1. As a volunteer, I want to be able to view points of interest on a map, so that I'll be able to tell visitors what is around the centre. (M)
2. As an administrator, I want to be able to login to an authenticated administration panel, so that I can ensure that only authorised people can change information about the points of interest. (S)
3. As an administrator, I want to be able to add and edit information about points of interest, so that I can ensure that there is up-to-date information about the wildlife centre. (M)
4. As a volunteer, I want the ability to showcase the information in both the medium of English and of Welsh, so that I can accommodate for all people who visit the centre, regardless of their preferred language. (C)
5. As an administrator, I want to be able to add images to different points of interest, so that I'll be able to show a visual representation of the point of interest. (C)
6. As a visitor, I want to be able to find directions to and from the centre using the application, so that I'll be able to find out how to get back home, and find out how to get to the centre by another form of transport if I want to visit again. (C)
7. As a visitor, I want to be able to find out about local businesses that are featured in the application, so that I can get more a feel for what is around the wildlife centre, and potentially visit some interesting businesses. (S)
8. As a visitor, I want to be able to be told about what the wildlife centre is doing, so that I can get more of a feel and be able to appreciate the area around me. (M)
9. As a visitor, I want to be able to look at webcams of the Ospreys that live at the wildlife centre. (W - This has been implemented separately by the project manager on the Dyfi Wildlife Centre's side)
10. As a volunteer, I want to be able to filter the map by different defining features, and also press a button in case I go too far from the centre. (S)

1.6 Tasks

Tasks will be stored as GitHub issues which will be broken down into further subtasks. Each subtask will have its own branch that will merge onto the task branch. Tasks will be carried out sequentially. Each story has a task attached to it, but it is broken down and interpreted in a software engineering sense.

1. Create an interface that includes a map, which has markers that can be added to it. The markers should link to a popup containing information about the point of interest. At this stage, the POIs can be hard-coded.
2. Create an API that allows for information about points of interest to be stored in a database. Program this API to automatically update the Google Maps API with markers and create a page to allow for this to be edited in a textbox and posted to the database. Focus on not having to 'hack' through the HTML or database, similar to a CMS. Worst case: use something like WordPress headless.
3. Re-visit the interface created in task 1 to ensure that the information popup is easy to present to someone. Ensure that the interface is easily-readable, for example by having marker clusters. Consider performing user surveys and informal user testing with friends, and even the customer if possible.
4. Re-visit the interface created in task 2 and use a simple authentication algorithm, such as OAuth 2.0. This should include some sort of user/password table and the ability to create new users. It does not necessarily have to have enterprise-grade encryption but should be taken seriously enough to disallow any simple attacks on the web server.
5. Create a list of filters that can be selected by the volunteer on the main screen. This will filter out some markers based upon specific boolean values. Will require some further spike work through online tutorials.
6. In the admin panel, allow for addresses to be entered and geocoded into coordinates, so that local businesses and places such as educational institutions can be added.
7. Enable I18N internationalisation, creating a basic Welsh version of the web page. Ensure that the admin panel accepts name and description information in both English and Welsh. Don't worry about the accuracy of Welsh as this can be corrected by the customer or by Welsh-speaking staff in the University.
8. Include the ability to upload images to the web server, or, failing that, hotlink images. Have them show up with each POI. A card component could be useful for this.
9. Ensure that forms of public transport are clearly marked on the map, if they haven't been already. Potentially import the directions API and allow visitors to ask the volunteer for driving and public transport directions back to wherever they live to be displayed on the screen.

- Potential task to implement bus and train times through an open source transport tims API. Examples include the NextBuses API. Transport API and Darwin, but this is of C priority.

10. Get the links for video feeds for the customer and implement them as buttons that open up an HTML5 video component.

Appendix B

Ethics Submission

Date: 20th February 2020 **Reference:** 15221

AU Status

Undergraduate or PG Taught

Your aber.ac.uk email address

mim39@aber.ac.uk

Full Name

Michael Male

Please enter the name of the person responsible for reviewing your assessment.

Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment

rrz@aber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)

CS39440

Proposed Study Title

Development of a map-based web application to be used by visitors and staff at the Dyfi Wildlife Centre

Proposed Start Date

27 January 2020

Proposed Completion Date

1 June 2020

Are you conducting a quantitative or qualitative research project?

Mixed Methods

Does your research require external ethical approval under the Health Research Authority?

No

Does your research involve animals?

Yes

Does your research involve human participants?

Yes

Are you completing this form for your own research?

No

Does your research involve human participants?

Yes

Institute

IMPACS

Please provide a brief summary of your project (150 word max)

A web application that provides information about specific points of interest around the Cors Dyfi Nature Reserve on a Google Maps API, with persistent data stored in a PostgreSQL database. The application will be accessed on a computer based at the Dyfi Wildlife Centre; its intention is for staff to more easily explain the work that the centre carries out. Proposed data collection includes regular user experience surveys and collaboration with staff at the Montgomeryshire Wildlife Trust regarding information about the centre and local businesses/transport they wish to showcase. There has also been a proposed task to include geospatial data from GNSS trackers that were placed on western ospreys prior to the beginning of this project. The ospreys live at the nature reserve and migrate to Africa during the winter.

I can confirm that the study does not involve vulnerable participants including participants under the age of 18, those with learning/communication or associated difficulties or those that are otherwise unable to provide informed consent?

Yes

I can confirm that the participants will not be asked to take part in the study without their consent or knowledge at the time and participants will be fully informed of the purpose of the research (including what data will be gathered and how it shall be used during and after the study). Participants will also be given time to consider whether they wish to take part in the study and be given the right to withdraw at

any given time.

Yes

I can confirm that there is no risk that the nature of the research topic might lead to disclosures from the participant concerning their own involvement in illegal activities or other activities that represent a risk to themselves or others (e.g. sexual activity, drug use or professional misconduct).

Yes

I can confirm that the study will not induce stress, anxiety, lead to humiliation or cause harm or any other negative consequences beyond the risks encountered in the participant's day-to-day lives.

Yes

Please include any further relevant information for this section here: Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?

Yes

Does the research pose more than minimal and predictable risk to the researcher?

No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?

No

Please include any further relevant information for this section here: If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here:

N/A

Appendix C

Code Examples

3.1 Distance between two coordinates

Th Haversine Formula calculates the great circle distance between two given points on a sphere, the Earth, in this instance. It utilises spherical trigonometry, which allows it to calculate the distance between two points by using a spherical function and calculating the result from it, utilising a measure of the Earth's radius. For the purposes of the project, this is used to calculate the distance between the Dyfi Wildlife Centre and a second coordinate pair.

```
/**
 * Calculates the distance between the given coordinates and the
 * coordinates of the Dyfi Wildlife Centre. This is calculated using the
 * Haversine formula.
 * <p>
 * Code adapted from
 * <a href="https://rosettacode.org/wiki/Haversine_formula#Java">here</a>.
 *
 * @return double containing the distance in miles to 4 significant
 * figures.
 */
@Override
public double calculateDistanceFromCentre() {
    final double EARTH_RADIUS_MILES = 3958.8; // Approximate radius of
    // Earth in miles, used to calculate the distance.

    /* Local variables used for cleaner code */
    double dyfiLat = 52.568774;
    double dyfiLng = -3.918031;

    double currentLat = this.getLatitude();
    double currentLng = this.getLongitude();
```

```
        if ((dyfiLat == currentLat) && (dyfiLng == currentLng)) {
            return 0; // If there is no difference between both coordinates,
            // distance of 0 is returned, to avoid unnecessary calculation.
        } else {
            double dLat = Math.toRadians(currentLat - dyfiLat);
            double dLng = Math.toRadians(currentLng - dyfiLng);
            dyfiLat = Math.toRadians(dyfiLat);
            currentLat = Math.toRadians(currentLat);

            double a =
                    Math.pow(Math.sin(dLat / 2), 2) + Math.pow(
                    Math.sin(dLng / 2), 2)
                    * Math.cos(dyfiLat)
                    * Math.cos(currentLat);
            double c = 2 * Math.asin(Math.sqrt(a));
            double result = (EARTH_RADIUS_MILES * c);
            /* Converts result into a BigDecimal that is then rounded to 4
            significant figures.
             */
            MathContext mathContext = new MathContext(4, RoundingMode.DOWN);
            BigDecimal bigDecimal = new BigDecimal(result, mathContext);
            return bigDecimal.doubleValue();
        }

    }
```

3.2 Google Maps callback function

This is a JavaScript function that performs clustering and adding of markers, it is loaded when the index page is loaded.

```
async function initMap() {
    const allPointsOfInterest = await getPointsOfInterest('/poi');
    const dyfiWildlifeCentre = {lat: 52.568774, lng: -3.918031};
     // Co-ordinates for the Cors Dyfi Nature Reserve,
    // which should be located at the centre of the map.
    map = new google.maps.Map(document.getElementById('map'), {
        zoom: 16,
        center: dyfiWildlifeCentre,
        mapTypeId: 'hybrid',
        disableDefaultUI: true,
        clickableIcons: false
    });
    /* Creates a marker clusterer. Initial markers are null as they are
     added iteratively
     */
```

```
    markerCluster = new MarkerClusterer(map,null,
    {imagePath: 'images/clusters/m'});
    allPointsOfInterest.forEach(
        poi => {
            const marker = new google.maps.Marker({
                position: {lat: poi.latitude, lng: poi.longitude},
                map: map,
                title: poi.name,
                description: poi.description,
                distanceFromCentre: poi.distanceFromCentre,
                category: poi.category
            });
            allMarkers.push(marker);
            markerCluster.addMarker(marker); // Iterative addition of a
            // marker to the cluster, that performs clustering automatically
            marker.addListener('click', function () {
                const element = document.getElementById('poiCard');
                element.querySelector('#poi_title')
                 .innerHTML = marker.title;
                element.querySelector('#poi_description')
                 .innerHTML = marker.description;
                element.querySelector('#poi_distance')
                 .innerHTML = marker.distanceFromCentre;
                const instance = M.Modal.init(element, {
                    dismissible: true,
                    inDuration: 500,
                    outDuration: 500
                });

                instance.open();
            });
        }
    );
}
```

3.3 POI card

This is a card containing Point of Interest information, that is populated using the aforementioned function.

```
<div class="modal" id="poiCard" th:fragment="poiCard">
    <div class="modal-content">
        <div class="container-fluid">
            <div class="card large z-depth-0">
                <div class="card-image">
                    <img alt="An example image" class="lazyload"
```

```
                        data-src="images/osprey.jpeg">
                    <span class="card-title black-text" id="poi_title"></span>
                </div>
                <div class="card-content grey lighten-4">
                    <p>This place is
                        <span id="poi_distance"
                              style="font-weight:bold"></span>
                        miles from the
                        Dyfi
                        Wildlife Centre</p>
                </div>
                <div class="card-content" id="scrollable">
                    <p id="poi_description"></p>
                </div>
            </div>
        </div>
    </div>
</div>
```

3.4 Security configuration

This is the method used to configure Spring Security across all parts of the web application.

```
 /**
     * Configures the security parameters for the application. Further
     * details are included within inline comments in the source code.
     *
     * @param http  An instance of HttpSecurity
     * @throws Exception    if there is an issue with the security
     * configuration.
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/", "/index").permitAll() // Permits all
                // requests to the single-page user-end application
                .antMatchers("/poi").permitAll() // Permits a GET Request to
                // the getAllPointsOfInterest() method, used only to update
                // the map
                .antMatchers("/css/**", "/js/**", "/images/**").permitAll()
                // Permits all request to static resources
                .antMatchers("/admin/**").hasRole("ADMIN") // Only
                // permits
                // authenticated users with the 'ADMIN' role to access the
```

```
                // admin panel
                .anyRequest().authenticated()
                .and()
                .formLogin()// Sets authenticated requests to be routed via
                // the login page
                .loginPage("/login")
                .permitAll()
                .and()
                .logout()
                .permitAll();
    }
```