CSC7057 – Individual Software Development Project – “Rota App” - a hospitality rota and employee management system

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

in

The Queens’s University of Belfast

By Isaac Barr

Date of Submission: 18th September 2020



School of Electronics, Electrical Engineering and Computer Science

CSC7057 – Individual Software Development Project

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a coversheet will *not* be marked.

# Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook?
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit.
4. Is clearly presented and proof-read.
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via GitLab.
7. Journal has been submitted.

*I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism -http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used. I am aware of the disciplinary consequences of failing to abide and follow the School and Queen’s University Regulations on Plagiarism.*

**Name**: ISAAC BARR

**Student Number:** 40106143

**Student’s signature:**

**Date of submission:** 18/09/2020

Table of Contents

[Declaration of Academic Integrity i](#_Toc50987014)

[List of Figures iv](#_Toc50987015)

[List of Tables vi](#_Toc50987016)

[Acknowledgements vii](#_Toc50987017)

[Abstract viii](#_Toc50987018)

[1 Problem Specification 1](#_Toc50987019)

[1.1 Problem outline. 1](#_Toc50987020)

[1.2 Current application used. 1](#_Toc50987021)

[1.3 Proposed users. 1](#_Toc50987022)

[1.4 Proposed timetable. 2](#_Toc50987023)

[2 Proposed Solution and justification of the development model 3](#_Toc50987024)

[2.1 Proposed Solution. 3](#_Toc50987025)

[2.2 What is a web application? 3](#_Toc50987026)

[2.3 Technologies used. 3](#_Toc50987027)

[2.3.1 Node.js. 4](#_Toc50987028)

[2.3.2 Express.js. 4](#_Toc50987029)

[2.3.3 Sequelize. 5](#_Toc50987030)

[2.3.4 JavaScript Object Notation (JSON). 5](#_Toc50987031)

[2.3.5 Angular. 6](#_Toc50987032)

[2.3.6 Bring all the technologies together. 7](#_Toc50987033)

[2.4 Why this technology stack was picked. 7](#_Toc50987034)

[2.5 Development Strategy. 7](#_Toc50987035)

[2.5.1 Waterfall model. 7](#_Toc50987036)

[2.5.2 Agile model. 8](#_Toc50987037)

[2.5.3 Selected Development model. 8](#_Toc50987038)

[2.5.4 Agile Development within the project. 8](#_Toc50987039)

[3 Requirements Analysis and Specification 10](#_Toc50987040)

[3.1 Requirement Elicitation. 10](#_Toc50987041)

[3.2 Product backlog. 10](#_Toc50987042)

[3.3 Use cases. 12](#_Toc50987043)

[3.4 Non-functional Requirements. 13](#_Toc50987044)

[4 Design 15](#_Toc50987045)

[4.1 Angular design. 15](#_Toc50987046)

[4.2 Application Design Structure. 16](#_Toc50987047)

[4.2.1 General User Interface. 16](#_Toc50987048)

[4.2.2 Sign in screen. 17](#_Toc50987049)

[4.2.3 Navigation bar. 18](#_Toc50987050)

[4.2.4 User Dashboard/Homepage. 18](#_Toc50987051)

[4.2.5 User Profile Component. 19](#_Toc50987052)

[4.2.6 Employee overview component. 20](#_Toc50987053)

[4.2.7 Rota creation component. 22](#_Toc50987054)

[4.2.8 Timesheet component. 24](#_Toc50987055)

[4.2.9 Report component. 25](#_Toc50987056)

[5 Implementation 26](#_Toc50987057)

[5.1 Login and logout. 26](#_Toc50987058)

[5.1.1 Login. 26](#_Toc50987059)

[5.1.2 Logout. 27](#_Toc50987060)

[5.2 Authentication guard. 27](#_Toc50987061)

[5.3 Employee management. 29](#_Toc50987062)

[5.3.1 Displaying all employees. 29](#_Toc50987063)

[5.3.2 Viewing an individual employee. 29](#_Toc50987064)

[5.3.3 Deleting an employee. 30](#_Toc50987065)

[5.3.4 Creation of an employee. 30](#_Toc50987066)

[5.4 Rota creation. 32](#_Toc50987067)

[5.4.1 Date controls. 33](#_Toc50987068)

[5.4.2 Drag and drop of employees. 34](#_Toc50987069)

[5.4.3 Inserting, deleting and editing individual shifts. 35](#_Toc50987070)

[5.5 Routes and controllers. 35](#_Toc50987071)

[5.6 Testing. 36](#_Toc50987072)

[5.6.1 User acceptance testing 37](#_Toc50987073)

[6 Evaluation 39](#_Toc50987074)

[List of external libraries used within application 42](#_Toc50987075)

[Bibliography 43](#_Toc50987076)

[Appendix 1 - Application structural design for admin and general users. 44](#_Toc50987077)

[Appendix 2 - Use Cases. 45](#_Toc50987078)

[Appendix 3 - Testing. 51](#_Toc50987079)

[Appendix 4 – Supervisorial Meetings. 54](#_Toc50987080)

[Meeting 1 14/05/2020 54](#_Toc50987081)

[Meeting 2 28/07/2020 55](#_Toc50987082)

[Meeting 3 11/08/2020 56](#_Toc50987083)

[Appendix 5 – Login method. 57](#_Toc50987084)

[Appendix 6 - GitHub repo link. 58](#_Toc50987085)

# List of Figures

[Figure 1 - Simple Express response 5](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986874)

[Figure 2 - Simple Sequelize Query 5](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986875)

[Figure 3 - Simple Sequelize Model 5](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986876)

[Figure 4 - MVVM Design Pattern 6](#_Toc50986877)

[Figure 5 - Application Flow 7](#_Toc50986878)

[Figure 6 - Agile Model 8](#_Toc50986879)

[Figure 7 - Burndown chart showing percentage of tasks needing competition vs ideal and actual 9](#_Toc50986880)

[Figure 8 - Use Case diagram for system 14](#_Toc50986881)

[Figure 9 - Angular design (Angular.io, 2020) 15](#_Toc50986882)

[Figure 10 -Sequence diagram for the getEmployeeDetails() called inside the user-welcome-component 16](#_Toc50986883)

[Figure 11 - Initial sign in screen sketch 17](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986884)

[Figure 12 - Actual sign in screen 17](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986885)

[Figure 13 - Flow chart for the process of logging in 17](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986886)

[Figure 14 - General User Navigation Bar 18](#_Toc50986887)

[Figure 15 - Admin Navigation Bar 18](#_Toc50986888)

[Figure 16 - Original user home page sketch 18](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986889)

[Figure 17 - Actual user home page design 19](#_Toc50986890)

[Figure 18 - Proposed profile design 19](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986891)

[Figure 19 - Actual user profile component 20](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986892)

[Figure 20 - Employee overview component 20](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986893)

[Figure 21 - Purposed employee component design 21](#_Toc50986894)

[Figure 22 - Individual employee view 22](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986895)

[Figure 23 - Purposed rota creation component sketch 22](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986896)

[Figure 24 - Actual rota component design 23](#_Toc50986897)

[Figure 25 - Flow chart for creating a shift 24](#_Toc50986898)

[Figure 26 - Actual timesheet component design 25](#_Toc50986899)

[Figure 27 - Purposed report component sketch 25](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986900)

[Figure 28 - Actual report component design 25](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986901)

[Figure 29 - onLogin() method 26](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986902)

[Figure 30 - login() method 26](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986903)

[Figure 31 - Logout() method 27](#_Toc50986904)

[Figure 32 - Flow of events for authentication guards 28](#_Toc50986905)

[Figure 33 - Admin authentication guard 28](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986906)

[Figure 34 - User and Admin routes with canActivate property 28](#_Toc50986907)

[Figure 35 - Employee-list component with call to getEmployees() method and viewEmployee() method 29](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986908)

[Figure 36 - paramMap interface used in the viewEmployee component 30](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986909)

[Figure 37 - deleteUser() method 30](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986910)

[Figure 38 - Employee form group 31](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986911)

[Figure 39 - onSubmit() method used in the create-employee component 31](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986912)

[Figure 40 - The use of Nodemailer to send an email to the created user containing their generated password 32](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986913)

[Figure 41 - ngOnChanges lifecycle hook within the manager-area component 33](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986914)

[Figure 42 - convert date function 33](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986915)

[Figure 43 - getShifts method 33](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986916)

[Figure 44 - drag and drop function 34](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986917)

[Figure 45 - removal of user from managerList 34](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986918)

[Figure 46 - Route to return all users inside the user-route file 36](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986919)

[Figure 47 - getUsers function 36](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986920)

[Figure 48 - HTTP request sent to route using Postman 37](file:////Users/Isaac_Barr/Documents/SoftwareProject/Report/Full%20disseration%20draft%201.docx#_Toc50986921)

# List of Tables

[Table 1 - Proposed Timetable 2](#_Toc50986846)

[Table 2 - Product backlog table 12](#_Toc50986847)

[Table 3 - Use Case for login 12](#_Toc50986848)

[Table 4 - Use case for log out 12](#_Toc50986849)

[Table 5 - Use case for reset password 13](#_Toc50986850)

[Table 6 - Selection of defects found during user testing and the solutions 38](#_Toc50986851)

[Table 7 - Flow of Events for viewing upcoming or past shifts 45](#_Toc50986852)

[Table 8 - Flow of Events for viewing total hours for selected week 45](#_Toc50986853)

[Table 9 - Flow of Events for adding a holiday 45](#_Toc50986854)

[Table 10 - Flow of Events for deleting a holiday 46](#_Toc50986855)

[Table 11 - Flow of Events for viewing profile 46](#_Toc50986856)

[Table 12 - Flow of Events for editing general profile information 46](#_Toc50986857)

[Table 13 - Flow of Events for changing a password 47](#_Toc50986858)

[Table 14 - Flow of Events for creating a shift 47](#_Toc50986859)

[Table 15 - Flow of Events for editing a shift 47](#_Toc50986860)

[Table 16 - Flow of Events for deleting a shift 47](#_Toc50986861)

[Table 17 - Flow of Events for viewing all employees 48](#_Toc50986862)

[Table 18 - Flow of Events for creating an employee 48](#_Toc50986863)

[Table 19 - Flow of Events for viewing employee information 48](#_Toc50986864)

[Table 20 - Flow of Events for editing employee information 49](#_Toc50986865)

[Table 21 - Flow of Events for deleting an employee 49](#_Toc50986866)

[Table 22 - Flow of Events for approving employee’s shift hours 49](#_Toc50986867)

[Table 23 - Flow of Events for editing employee’s shift hours 49](#_Toc50986868)

[Table 24 - Flow of Events for deleting a user’s shift 50](#_Toc50986869)

[Table 25 - Flow of Events for adding an unscheduled shift for employee 50](#_Toc50986870)

[Table 26 - Flow of Events for creating payroll reports 50](#_Toc50986871)

[Table 27 - Tests carried out for each user story 52](#_Toc50986872)

[Table 28 - Tests carried out that would show errors 53](#_Toc50986873)

# Acknowledgements

Firstly, I would like to thank my supervisor, Dr Janak Adhikari who was extremely helpfully and supportive throughout the lifecycle of this project. Any questions or problems I had were always dealt with extremely quickly.

I would also like to thank the team members at the Parson’s Nose in Hillsborough who took part in the User Acceptance tests and provided helpful feedback for the application.

Furthermore, I would like to thank my girlfriend, Katie Malseed who has been very supportive and loving throughout the whole process.

Lastly, to my parents, whose kindness and generosity made it possible to study this master’s degree.

# Abstract

Rota App is a hospitality employee and rota management web application. The developed system makes it easier for hospitality managers to create weekly rotas, view payroll and allows for easy employee management. This report provides an outline of how the system was developed though an outline of the problem, the proposed solution, analysis of requirements, design decisions, explanation of implementation methods and an evaluation of the project. The application was developed using Angular, HTML, CSS for the front end with Node and Express being used for the backend.

# 1 Problem Specification

## 1.1 Problem outline.

During my time studying at university I have been working in the hospitality industry, while this is extremely enjoyable, the main problems, I continually run into is the time and effort it requires to create a full weekly rota for all staff members that can be distributed easily to all members of staff ang give management an overview. In my experience building a rota from scratch can sometimes take up to 5 hours a week as staff requests, busy service periods, staff training levels and many more factors have to be taken into account. Once all these factors are considered and the rota has been published, there is always an unforeseen circumstance, such as a staff member being unable to work or wanting to swap their shifts with another employee, or an unexpected increase in customers. This again results in time and effort being wasted looking at a rota when a manager’s time could be spent focusing on more beneficial tasks. Every company within the hospitality industry is different but the majority share the same common problem of the time, effort and cost associated with creating rotas for their employees. Furthermore, other ‘back of house’ tasks such as payroll management take valuable time and effort to complete to high standard.

## 1.2 Current application used.

In the current market there is a number of software systems that can be used in order to tackle this problem. One such example is Rota Cloud. While it already is an established system there is a cost associated with using it. The price is based upon the number of employees each company has. If a company has 30 employees, the price to use it is up to £99 per month. Given the current economic climate and the uncertainty associated with the COVID-19 epidemic, this means that some companies will not be able to afford to use the technology. An ideal solution is to create a hospitality rota and employee management system that can be used for free.

## 1.3 Proposed users.

The proposed software system would have two main users of a manager (or administrator) that and a general employee and the way they interact with application will differ depending on what type of user they are.

A general employee can use the application to view their designated shifts for each week. They can request and manage their time off.

A manager would have all the same functionally as a general employee will have however, a manager or admin will be allowed to easily build rotas based upon employee availability, their training and cost. They can use the application to reduce the stress associated with the changing a rota. For example, if a staff member is unable to work, they can easily find another staff member to cover the now unallocated shift. A manager can also view and approve each employees’ timesheet and based on the information provided they can view payroll summaries to track business costs.

## 1.4 Proposed timetable.

By using information provided in ‘CSC 7057 Handbook 2019/2020’ a proposed timetable has been drawn up, that will ensure that both the developed software application and the written dissertation will be delivered within the specified timeframe. As per the handbook, the dissertation has been spilt up into six sections. Each section will be completed simultaneously with the development and testing of the application and a proposed timetable of start and due dates can be seen in *Table 1*. In terms of defining potential problem areas at this stage is difficult as key phases of development have not begun yet and so it is hard to pinpoint when and where problem may arise. As this is the case each task has been allocated a number of extra days as a ‘grace period’ to ensure that if any problems do arise, they can be dealt with and will not hinder the completion of the whole project. On the other hand, some areas may not take as long as expected and so the time saved can be used to start working on the next required section. It should also be noted that some of the required sections are dependent upon other tasks being finished before work can begin on it. For example, in order to complete ‘Chapter 6 – Evaluation and Conclusion’, the majority of all development of the application must be finished before being able to draw a critical conclusion on the project.

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Start Date | Due Date | Total Days |
| Chapter 1 - Submission of Project Plan | 04-May | 22-May | 18 |
| Chapter 2 - Solution and Justification Development Model | 22-May | 05-Jun | 14 |
| Chapter 3 - Requirement Analysis | 06-Jun | 20-Jun | 14 |
| Chapter 4 - UI Design | 22-May | 01-Jun | 10 |
| Chapter 4 - Software System Design | 08-May | 31-Aug | 28 |
| Development of Application | 01-Jun | 31-Aug | 91 |
| Testing of Application | 01-Jun | 31-Aug | 91 |
| Chapter 5 - Implementation | 17-Aug | 31-Aug | 14 |
| Chapter 6 - Evaluation and Conclusion | 17-Aug | 31-Aug | 14 |
| Final Sprint of Application Development and Testing | 01-Sep | 07-Sep | 6 |
| Demo Preparation | 07-Sep | 11-Sep | 4 |
| Submission |  | 18-Sep |  |

Table 1 - Proposed Timetable

# 2 Proposed Solution and justification of the development model

## 2.1 Proposed Solution.

The proposed solution to the problem described above is to develop a dynamic web application that eases the time, effort, and costs associated with the ‘back of house’ tasks in the hospitality industry. The application would aim to solve the problems by developing a staff rota system that makes it easy to plan, edit and publish rotas. As well as this, staff attendance and the hours worked per shift can be recorded allowing for easy creation or integration of payroll summaries. These provided solutions would help to create an application that would reduce the associated problems surrounding ‘back of house’ hospitality management.

## 2.2 What is a web application?

In order to be able to create a web application or web-app it is important to define what a web application is, as this will guide the selecting of what technologies will be of benefit to use. Lotfy (2018), describes a web application as a client-server software program where the backend runs on a web server. The ‘backend’ is what handles the ‘behind-the-scenes’ functionality of the web application, such as storing and retrieving information, and the client-side or ‘front-end’, displays the information provided to a user. Furthermore, the front-end also allows the user to interact and in some cases change the information provided by the backend. There is usually a typical flow when a user interacts with a web application and is as follows, Gibb (2016):

1. A user makes a request to a web server through the front-end interface.
2. A web server forwards this request to the appropriate web application server.
3. The web application server performs the requested task such querying a database or processing the data and generates the results of the requested data.
4. The web application server sends the results to the web server with the requested information or processed data.
5. Web server responds back to the client with the requested information that then appears on the user’s display.

This typical flow of requests from the frontend and backend is what makes up a web application and this idea was used in the creation of the solution. *Figure 5* shows this has idea has been used within the application.

## 2.3 Technologies used.

There are many different frameworks and ecosystems that are used to create web-applications and they are often referred to as a stack. Stacks are made up of technologies used to create the frontend, a backend and a database, some examples include:

* LAMP stack: JavaScript - Linux - Apache - MySQL – PHP
* Django stack: JavaScript - Python - Django - MySQL.

While each so-called stack has its advantages and disadvantages, for this project a M.E.A.N stack was used. The M.E.A.N stack which is often referred to as ‘mean stack’, or just mean, is a collective of four pieces of technology: MongoDB, Express.JS, Angular, and Node.JS. The strength of using the Mean stack is that it relies on the centralised use of JavaScript as the basic programming component (Elrom, 2016). This allows for robust, easily maintainable web applications with a relatively quick building process (IBM, 2019). While, in a traditional mean stack a non-relational database of MongoDB would be used. However, due to having no experience in working with non-relational databases, a relational database of MySQL was used within this web application.

### 2.3.1 Node.js.

Node.js is an open-source and cross-platform JavaScript runtime environment that uses Google Chrome’s V8 execution engine and enables the development of all kinds of server-side tools and applications. As a result, the runtime is intended to be used, outside the browser and as such the environment omits browser specific JavaScript API’s and other traditional API’s including HTTP. This has a number of benefits, one of which being the speed at which it can do this at. This speed comes from it running in a single process and not creating a new thread for every request. Instead Node.js uses a set of asynchronous I/O (input/output) primitives that prevent JavaScript code from blocking (Node.js. 2019). Other webservers such as Apache use the blocking I/O model to handle requests and use multi threads that increases complexity and overhead for the server as it causes threads to wait for I/O while the server processes and retrieves the data and as such diminish speed and performance (D. Dunka, A. Emmanuel and O. Oyerinde, 2018). Instead when Node.js performs an I/O operation, like reading from the network or accessing a database, it won’t block the thread and resumes the desired operations when the response comes back.

When discussing Node.js, its support for its built-in package manager using NPM (Node package manager), should not be left out. This is a tool that comes by default with every Node.js installation and allows the access to a set of publicly available, reusable components, that are available and installed using the NPM CLI.

### 2.3.2 Express.js.

In order to add specific handling for different HTTP verbs (e.g. GET, POST, DELETE, UPDATE, etc), separately handle requests at different URL paths ("routes"), serve static files, or use templates to dynamically create the responses, a web framework can be used (Mozilla, 2018). While it is possible to do this via all this Node alone, it is time consuming and this is where Express.js, or more simply Express is used. It is described as a minimal and flexible Node.js web application that provides a robust set of features for web and even mobile applications (Express.js, 2020). It manages everything from routing to handling requests and views. Express, allows the creation of a REST API server and makes it easier to write secure, modular and fast Node.js applications with higher efficiency, reliability and less duplications.

Within this application Express has been used within Node to provide responses depending on client requests. *Figure 1* shows a simple example of how express responses when a specific URL is requested by the client.

A screenshot of a cell phone

Description automatically generated

Figure 1 - Simple Express response

### 2.3.3 Sequelize.

Instead of writing raw SQL queries to insert, update and delete data from the database an ORM (Object Relational Mapping), called Sequelize was used. Object Relational Mapping is a technique that maps software objects to database schema. By interacting with these objects, it meant that no raw SQL database queries had to written and management of the database was easier. When an object is read, created, updated or deleted, Sequelize creates a SQL string that queries the database and returns the result as a JavaScript object. The benefit of using Sequelize is that allows the use of JavaScript instead of SQL for database querying, which as a result simplifies the code as well as making it more readable.

Sequelize models are an abstraction that represents a table in the database. The model is used to tell Sequelize about the entity it represents, such as the name of the table in the database and which columns it has (and their data types) (Sequelize, 2020). Figure 2 shows an example of a model used within the application.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedTo query the database the models are used along with a number of Sequelize methods that are converted into SQL statements and an example of such can be seen in Figure 3. As mentioned before, by using Sequelize it makes querying the database a lot easier, especially when more complex SQL statements are required.

Figure 2 - Simple Sequelize Query

Figure 3 - Simple Sequelize Model

### 2.3.4 JavaScript Object Notation (JSON).

JSON is a way to store information in an organised, easy-to-access manner. It gives a human-readable collection of data that can be accessed in a logical manner. It is also used to transmit serialized data over a network connection (D. Dunka, A. Emmanuel and O. Oyerinde, 2018). In this application it is used to transfer data from the web server (Node) to the front-end (Angular) and vice-versa.

### 2.3.5 Angular.

Angular is a JavaScript framework that allows the creation of client-side or front-end applications (Angular.io, n.d.). It provides many features such as components, directives, data binding, form processing, services, and dependency injection (Joshi, n.d.). Angular applications are written using TypeScript, a superset of JavaScript and provides many object-oriented features such as classes, interfaces, and data types. It is then complied into JavaScript so that it can be used on any browser.

Angular uses the Model-View-ViewModel (MVVM) pattern that is derived from the MVC model. The MVVM separates the development of the user interface from the development of the back-end logic (The MVVM Pattern, 2012). There are three components of the MVVC pattern (*Figure 4)*:

1. *The Model*. It refers to the to the domain model that includes a data model along with business logic and validation logic.
2. *The View*. It is responsible for defining the structure, layout and appearance of what the user sees on the screen.
3. *The ViewModel*. The ViewModel is responsible for handling the view logic. The view model interacts with the model by calling methods in the model class. It then provides data from the model in a form that the view can easily use. (The MVVM Pattern, 2012).

A picture containing screenshot

Description automatically generated

Figure 4 - MVVM Design Pattern

Angular supports this MVVM design pattern by using a service to implement the Model. The View is implemented using Angular’s template view (HTML with data bindings). As the View is bound to the ViewModel when a property is changed in the ViewModel it is instantly reflected in the View. The ViewModel is implemented using the Component decorator.

As Angular follows the MVVM design pattern it cleanly separates the view from the models and allows the creation of HTML templates that are dynamically filled with data and automatically update when the data changes. This was one of the main reasons why Angular was chosen to develop the client side of the application.

### 2.3.6 Bring all the technologies together.

All the described technologies were brought together to develop a dynamic web application. On the front-end Angular has been used to provide an interface for the user, allowing them to make requests and view requests. Node has been used as web server to parse and return user requests. Express has been used to make HTTP requests to the database. Information and is passed between each technology using JSON data (*Figure 5*).

A screenshot of a cell phone

Description automatically generated

Figure 5 - Application Flow

## 2.4 Why this technology stack was picked.

Due to the time constrains associated with such a project, the main reason as why this stack was picked was its wide use of JavaScript. This allowed for a relatively quick build time and meant that it was it was easier to move between writing client side and server-side code, as they are all written in the one language. Furthermore, with Angular using the MVVM architecture it can provide high quality user interfaces. With Node’s unblocking architecture it means that the server-side is ‘quicker’ when compared to other server-side languages. As well as these factors there is a large community of developers, making find solutions to problems that were encountered easier and quicker to solve.

## 2.5 Development Strategy.

A clearly defined development strategy is key when partaking in the development of an application. It defines the rules, helps to monitor time and ensures some task are not overlooked. In technological development there are two main strategies that are followed. These are the Waterfall model and the Agile model. This section aims to provide a background on each model and draw a conclusion as why the selected development model/strategy was used within this project.

### 2.5.1 Waterfall model.

The waterfall model, which comprises of linear successive phases (requirements engineering, design, implementation and testing) with each phase needing to be complete before the next phase can begin. It provides advantages such as providing a clear set of requirements upfront that allow milestones to be easily monitored (Sommerville, 2011). Furthermore, it provides a clear audit trail with responsibilities being easily understood by all involved and can be used to easily define associated costs.

### 2.5.2 Agile model.

The agile model is software development approach based on iterative development. Agile breaks tasks into iterations or sprints that typically last for two to four weeks, with a situation review at the end of the cycle, (van Vliet, 2008), *Figure 6*. Each iteration involves working through a full software development cycle including planning requirements analysis, design, coding, and testing. The main aim of agile to reduce the amount of upfront planning and documentation and via each iteration it builds upon the functionality created in the previous sprint. This allows for continuous delivery of useful software and functionality can be changed depending upon changing circumstances.

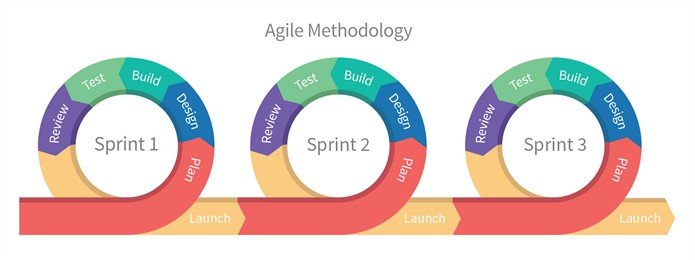


Figure 6 - Agile Model

### 2.5.3 Selected Development model.

Due to the timeframe and size of the project, the Agile model was deemed more suitable for the project as it focuses on individuals and interactions over processes and tools and working software over comprehensive documentation. The planning stage associated with the Waterfall model is usually a long drawn out phase at the beginning of the project that provides requirements, specifications and documentation. While this can be useful in large scale projects, it does require a lot of work at the beginning and it does not allow for changes throughout the process. This process can be useful in large scale projects, but it should be remembered that this is an individual project and changes to requirements and functionality need to be embraced quickly. Furthermore, at the beginning of the project, the developer was not entirely familiar with the chosen technology making the planning stage, related with the Waterfall model, slightly more difficult. Instead, by using the Agile model, changes to design and functionally can be quickly implemented in each iteration as knowledge is improved.

### 2.5.4 Agile Development within the project.

As stated before, the Agile development takes the form of small incremental cycles and throughout this project these sprints took place. Within this project five sprints took place each lasting two to four weeks and with supervisory meetings taking place within this time frame it provided the perfect opportunity to analyse the current process of the project and change requirements, if required.

At the beginning of the first sprint, a product backlog was created to act as a list of the work that needed to be done to the application. In each sprint a smaller list, or sprint backlog, was taken from the product backlog and those features and requirements were implemented. The product backlog consisted of user stories and these were directly implemented as tasks for the project.

A burnout chart was used a visual representation of tasks to complete in the product backlog vs time within the project, see *Figure 7.* At the beginning of the project it can be seen that a large number of tasks were not completed within the first sprints. This is due to the aforementioned lack of experience in using the selected frameworks, however, as familiarity increased it can be seen that the rate of task completion sped up.

Figure 7 - Burndown chart showing percentage of tasks needing competition vs ideal and actual

As each following sprint contained each element of the waterfall model e.g. implementing and testing, it allowed for continuous development and testing of the existing software. As more tasks were completed in later sprints it allowed for more vigorous testing including alpha testing that allowed any bugs not recognised by the developer to be fixed. By carrying out testing at each stage it meant that any potential defects could be quickly identified and take less time to fix (van Vliet, 2008). As well as this, any input from testers regarding functionality could also be implemented.

After each sprint was completed, a sprint review was carried out. A sprint review looks at the pervious sprint, validating the system and decides what user stories should be implemented in the next sprint. By carrying out the Agile development strategy throughout the project it meant that all user stories were implemented to high standard, removing any potential defects at the outset.

# 3 Requirements Analysis and Specification

## 3.1 Requirement Elicitation.

Requirement elicitation is the process of discovering and understanding the user needs and constraints for the purposed system. It provides developers with a helping hand in understanding what problems may arise and what the projects end goal is. During this stage the product requirements are decided, and these can take the form of both functional and non-functional. Sommervile (2016), describes functional requirements, as what a system should do with non-functional requirements not being concerned with what the system does. Instead, they are constraints on the services or functions offered by the system such as cost, performance and reusability. During the first supervisorial meeting the functional requirements were discussed, which led to the creation of user’s stories. Each user story consists of a short description written, from the user’s point of view with natural language. It focuses on what the user needs, instead of what the system should deliver and are a very effective technique in requirements elicitation because their narrative structure helps users to describe what happens in different processes in the system (Sommervile, 2016), (Britton and Doake, 2005). With the creation of the user stories this enabled a product backlog to be developed based upon these user requirements.

The main aim of the system is to provide a functional and easy to use hospitality and employee management product. As stated, these requirements were discussed in the first meeting. However, as the Agile method was used in development new functional requirements were discovered and new user stories created. This meant that these new requirements were added to the existing and were added to the product backlog and were completed in the next sprint.

## 3.2 Product backlog.

As the user stories were created, they helped to develop the product backlog. Each of the task was then given a priority level using the MoSCoW method. Using this method there is 4 levels of prioritisation “must have” (1), “should have” (2), “could have” (3) and “won’t have” (4) with “must have” being the highest level and won’t have being the lowest (BABOK Guide, 2009). Using this method, it meant that tasks given the must have or high priority level were completed in the first sprints, and if time allowed tasks with a low level or “wont-haves” could be implemented later in the development stage. *Table 2* shows the product backlog with corresponding priority levels that was followed throughout the development of the system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | MoSCow Priority Level \* | Sprint | User Type | Wants to… | So that… |
| 1 | 1 | 1 | User/Admin | Login into the system | They can use the application |
| 2 | 1 | 1 | User/Admin | Logout of the system | End the ‘session’ and remove all unsaved changes |
| 3 | 1 | 1 | User/Admin | Reset their password | They can access the application if they have forgotten their password |
| 4 | 1 | 2 | User/Admin | View their upcoming and previous shifts | They can view their upcoming shifts for each week |
| 5 | 3 | 2 | User/Admin | View their total hours for the week | They can view their total hours for each week |
| 6 | 2 | 2 | User/Admin | Add a holiday | Add a holiday so that they are not scheduled to work |
| 7 | 3 | 2 | User/Admin | View their upcoming and previous holidays | View their upcoming and previous holidays |
| 8 | 1 | 2 | User/Admin | View their profile | Can view their information |
| 9 | 1 | 2 | User/Admin | Edit their information | They can make changes to any incorrect information |
| 10 | 2 | 4 | User/Admin | Change their password | They can edit their password if forgotten or want to change it |
| 11 | 3 | 4 | User/Admin | View their training level | They can see their training levels |
| 12 | 4 | ^NI | User/Admin | Create a profile picture | It makes it easier to remember employees |
| 13 | 1 | 2 | Admin | Create weekly shifts for all employees | They can easily create a weekly rota for all employees |
| 14 | 1 | 2 | Admin | Edit an individual shift | Any mistakes or changes can be corrected |
| 15 | 1 | 1 | Admin | Create an employee | A new employee can be included in weekly shift creation |
| 16 | 3 | 3 | Admin | View employee information | Gain a better understanding when creating rotas |
| 17 | 3 | 3 | Admin | Edit employee information | Any mistakes or changes can be corrected |
| 18 | 2 | 3 | Admin | Edit individual employee pay | Timesheets and weekly reports can provide more in-depth information |
| 19 | 1 | 2 | Admin | Change week and day when creating rotas | Rotas can be created for upcoming days/week and past and future rotas can be viewed |
| 20 | 2 | 3 | Admin | Approve employees’ hours | Payroll reports can be created |
| 21 | 2 | 3 | Admin | Edit employee's hours | Any mistakes or changes can be corrected |
| 22 | 2 | 3 | Admin | Add an unscheduled shift for employee that wasn't on the rota to work | Payroll reports and user reports are accurate |
| 23 | 3 | 5 | Admin | Change user’s role | Selected user can access to system can be changed |
| 24 | 3 | 5 | Admin | Add employee training | Users training can be view |
| 25 | 2 | 4 | Admin | Create payroll reports for each individual employee or for all employees | Payroll can be easily calculated based on total hours worked and employee pay |
| 26 | 4 | ^NI | Admin/User | Create a messaging service so that employee can talk to each other | So they can swap shifts or talk about tasks that need to be done when in work |

Table 2 - Product backlog table

\*Priority Level: (1) - Must have/Critical. (2) - Should have/High. (3) - Could have/Medium. (4) Won’t have/Low.

^NI – not implemented

## 3.3 Use cases.

A use case view describes the externally visible behaviour of the system. It presents a structed view of a systems functionality by defining actors, which model the roles that users can play, when interacting with the system and describing the use cases that those actors can participate in (Priestley, 2003). The use case view contains a set of use cases that define the complete functionality of the system. Uses cases aid in the development of the system by imposing constraints on the designer and dividing the system into smaller functions, so that the interface can be designed from user’s point of view (Priestley, 2003). A small selection of use cases for the system can be found at *Table 3,Ttable 4* and *Table 5*, with the reminder located in the appendix.

|  |  |
| --- | --- |
| **Flow of Events for the Login Case** | |
| Actors | Admin/User |
| Objective | To log the user in |
| Precondition | 1. The user is not logged in 2. The user is a registered user |
| Main Flow | 1. The user enters their email and password 2. The user clicks the ‘sign in’ button 3. The system navigates to the user’s dashboard 4. A local session is created for that user |
| Alternative Flows |  |
| Error Conditions | 1. If the user enters the wrong details the system will show an error message depending upon what details are incorrect |
| Post Conditions | The user is logged in and the system navigates to their dashboard |

Table 3 - Use Case for login

|  |  |
| --- | --- |
| **Flow of Events for the Logout Case** | |
| Actors | Admin/User |
| Objective | To log the user out |
| Precondition | 1. The user is logged into the system |
| Main Flow | 1. The user clicks the logout button located in the navigation bar 2. The system logs the user out 3. The user is returned to the login screen |
| Alternative Flows | 1. The user closed the browser or the tab |
| Error Conditions | - |
| Post Conditions | The user is logged out of the system and their session is ended |

Table 4 - Use case for log out

|  |  |
| --- | --- |
| **Flow of Events for the reset password** | |
| Actors | Admin/User |
| Objective | To reset the user’s password |
| Precondition | 1. The user is a registered user |
| Main Flow | 1. The user clicks the “forgot password?” link on the sign-in screen 2. The user is navigated to the reset password page 3. The user completes the form associated with the reset password 4. The user clicks the “reset password” button 5. An alert showing that the password has been changed shows 6. The user is returned to the login page |
| Alternative Flows | - |
| Error Conditions | 1. An error will show if some details in form are incorrect |
| Post Conditions | The user’s password is reset, and they are returned to the login page |

Table 5 - Use case for reset password

A use case diagram, summaries in graphical form the different actors and use cases in a system, showing how and which actors can participate in which use cases. *Figure 8* shows the use case diagram for the designed system.

## 3.4 Non-functional Requirements.

As stated, non-functional requirements are not directly concerned with the specific services delivered by the system to its users. Instead they usually specify or constrain characteristics to the system as a whole. Non-functional requirements are often more critical than functional requirements as failing to meet a non-functional requirement can mean the whole system is unstable (Sommerville, 2016). Non-functional requirements can be divided into a number of categories including performance, reliability and usability. During the first supervisorial meeting the non-functional requirements were discussed and a list of these can be found below:

* *Usability* – This measures the characteristics such as consistency and aesthetics in the user interface. It is the ease at which the users operate the system and make productive use of it (Paradkar, 2017):
  + *It must be easy for a user to navigate and pleasant to use.*
  + *The system must be easy to remember for the casual user.*
  + *The user must understand what the system does.*
* *System –* This includes software and hardware responses:
  + *The system must be designed to be user on a web browser, more specifically Google Chrome.*
* *Performance –* A measure of how quickly the system responds to changes:
  + *All pages must not take longer than 10 seconds to load.*
  + *The system must not take longer than 10 seconds to log a user in.*
  + *All buttons must respond when they are clicked.*
* *Security* 
  + *Only registered user can use the system*
  + *To store a password in the database it must be hashed*
* *Reliability -* The ability of the system to function under stated conditions for a specific period
  + *All created features must work all of the time and error messages should show if part of the system fails.*

A close up of text on a white background

Description automatically generated

Figure 8 - Use Case diagram for system

# 4 Design

## 4.1 Angular design.

In Angular applications, modules are the basic building block. Each Angular application has a root module, named *AppModule* and provides the bootstrap mechanism that launches the application (Angular.io, 2020). Within the *AppModule* it contains child modules that are hierarchal in nature. While a small application many only contain the one module, larger applications, like the one developed, have a number of feature modules. A module groups together components, directives, pipes and services that are related so that they can be combined with other modules to create an application.

As well as having at least one module, every Angular application has at least one component. In the application this is *AppComponen*t which is the root component and connects a component hierarchy with the page object model, or DOM. Each component defines a class that contains application data and logic and is associated with an HTML template that defines a view to be displayed (Angular.io, 2020). Shared functionality between components are written into services. Services are injected into components and a component can delegate tasks, such as fetching data from the server or validating user input to the services. *Figure 9* shows how all these main building blocks of an Angular application are related.

A picture containing food

Description automatically generated

Figure 9 - Angular design (Angular.io, 2020)

The organising of code into distinct functional modules with components and services helps to develop complex applications with reusable code, allowing applications which are lean and efficient (Angular.io, 2020). This modularity can be seen throughout the designed application for example, the *profile module* uses a number of components such as *user-information* and *user-shifts* which each have their own class that contains the logic for that component and their own HTML template. These individual components are then combined together to give the *profile* page in the application. Furthermore, the majority of components make use of the created shared services. For example the *user-welcome-component* uses the *getEmployeeDetails()* method that contains a HTTP GET request inside the *employee.service* to gather user information from the database and display it in the relative HTML template. A sequence diagram found at *figure 10* shows this process in more detail.

A screenshot of a social media post

Description automatically generated

Figure 10 -Sequence diagram for the getEmployeeDetails() called inside the user-welcome-component

## 4.2 Application Design Structure.

Following from the requirement elicitation, the structure of the application was designed. Using the Angular module and component design and the product backlog, the application was spilt into a number of key components that provide similar functionality. Through the process of the requirement elicitation and the problem specification it was clear that the application would provide different functionality and components depending on what the status of the user was. If a user was assigned admin status the system would contain a dashboard component, a rota component, an employee overview component, a timesheet component, a report component and a profile component. Whereas, a general user’s system would contain a dashboard and a profile component. Each of these components design will be discussed later in this chapter. The application structure for both type of user can be found in appendix one.

### 4.2.1 General User Interface.

One of the non-functional requirements that was proposed, was that the system should be easy to use so the user interface was designed with this in mind. It should be easy for users to use the system and be memorable, so that once they learn how to use a feature once, they don’t need to learn how to use it again. As well as this, the user interface was also designed in a way that made it simple to navigate between functions and various components quickly and easily. As a simple design and easy to learn features were needed, a CSS framework was used. The framework used is called MDB-Bootstrap.

By using a framework, it meant that the overall design, in terms of typography, colours and input controls was consistent and looked similar throughout the application. The framework also provides many UI elements that general web users are already familiar and so doesn’t require large amounts of time to be dedicated to learning how to use the system and its controls. The general user interface uses a blue navigation bar, present at the top of every page once a user is logged. Furthermore, the main body of the application has a white background with black text making the information easy to read. Buttons are coloured according to their action, for example a delete button is given the colour red as this colour naturally makes user pause and think about whether or not they want to perform the action. As well as this, any error messages are shown in red with success message being given the colour green. Feedback from user tests showed that these designs to the general user interface makes it quick to learn and use the system as a whole.

### 4.2.2 Sign in screen.

**A screenshot of a cell phone

Description automatically generated**The sign in screen is the landing page to the application. It is a simple design and contains a simple form where registered users enter their email and password and click the button to sign in. If the credentials are correct the system navigates them to the relevant user home page. *Figure 11* shows the initial design with *Figure 12* showing the actual design. It can be seen that there isn’t a huge difference between each, this is because a sign screen follows a similar pattern no matter what application you use.

**A picture containing text, photo, looking, light

Description automatically generated**

Figure 11 - Initial sign in screen sketch

Errors will show in red if either input field is touched and left empty. Furthermore, if the email input doesn’t match a standard email format an error will also show. If the user enters the details and it doesn’t match the stored credentials another error will show indicating which input doesn’t match the systems records. At the bottom there is a forgot password link that navigates a user to the forgot password page where they can reset their password. *Figure 13* shows a flow chart for the process of events when a user uses the sign in screen.

Figure 12 - Actual sign in screen

A close up of a device

Description automatically generated

Figure 13 - Flow chart for the process of logging in

### 4.2.3 Navigation bar.

Once a user has logged, every page features a navigation bar, or ‘navbar’, at the top of the screen that is present on every page within application. This navigation bar is different depending on a user’s status and allows for easy navigation between features and pages throughout the application. *Figure 14* shows the general user nav bar with *Figure 15* showing admin navigation bar. A navbar was chosen as allows users to navigate easily through the application features. When the user is on a selected page the item associated within the navbar is given a CSS class of active to ensure that a user knows what page and feature they are using. It also contains a log out link that logs the user out of the application and then they are navigated to the sign in screen.



Figure 14 - General User Navigation Bar



Figure 15 - Admin Navigation Bar

### 4.2.4 User Dashboard/Homepage.

**A close up of text on a white background

Description automatically generated**Following a successful login, the user is presented with the user dashboard. The admin and the user dashboard are both the same. During the design phase it was decided that the user home page should show an overview of the users shifts for the current week. *Figure 16* shows the original design thought up during the initial design phase. By using the user details that have been entered during the login in phase an HTTP GET request sends a call to the API. This then returns the relevant information for the user and they are presented with their current shifts. From user testing it was made clear that it would be beneficial for a user to be able to view who is also working throughout the week. A button was added under each day and when clicked it shows a more detailed view of who is working, what time they are working and the area in which they are working. *Figure 17* shows the actual design of the user home page.

Figure 16 - Original user home page sketch

**A screenshot of a cell phone

Description automatically generated**

Figure 17 - Actual user home page design

### 4.2.5 User Profile Component.

A close up of text on a white background

Description automatically generatedAnother component in which the user and admin share is the profile component *Figure 18* shows the original design sketch and *Figure 19* shows the actual design. The profile shows all the general user information, their shifts, their holidays and their training. The general user information shows details such as the users name, email, address and phone number. It was decided that this should be editable so that if any information regarding the user can be changed and the system can be updated according. If the user wants to edit their information, they click the edit button which launches a modal in which they can change their details. Form controls were used to ensure that no field was left blank and information was filled in according. This helps to ensure data integrity with regards to the database.

Figure 18 - Proposed profile design

Below the general user information is the *“*your shifts” section. This is used to show the users upcoming and past shifts. Each shift is shown with the date, start time, finish time and area in which the user is working. Moreover, it shows if the user is working today. If they are it displays the start time, finish time and area. However, if the user is not working today, it will display a message of “You’re off today, have a nice day!”. The original design intention in regards on how to display the shifts was to show every upcoming and past shift. However, after using the system it became clear that this idea would lead to the page being cluttered and messy as a user could have a large number of upcoming and past shifts. Instead, the upcoming shift area is limited to show 5 upcoming shifts and the past shifts were moved into side modal in which the user can scroll through and view their pervious shifts.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedThe ‘unavailability’ section contains all the users upcoming and past holidays. As well as being able to view the dates there are unavailable, they can use it add a date in which they cannot work. The ‘+’ icon opens a dropdown containing a calendar in which they can select a date in the future to request off. This is then added to the upcoming holidays. If the user tries to add a date that they have already requested off an error message will show stating that this day has already been booked off. As well as this a user can delete an upcoming holiday by clicking the red ‘trash-bucket’ icon. One item that was not included in the original design was the “holiday count”. This was added because normally employees are limited to a set number of holidays and this ensures that the user will not exceed this amount.

Figure 19 - Actual user profile component

Lastly the training section shows the users level of training. The original idea was that a user could add their own training, such as having a first aid training certificate. However, this was decided against because it could lead to false or misleading being inserted by the user. Instead, the admin can add training to the user in the employee component.

### 4.2.6 Employee overview component.

The employee overview component was designed so that it would list all the employees registered to use the system. An admin can use this to create an employee, as well as being able to view and edit an individual employee, see *Figure 20*. Furthermore, employees that no longer work for the company can be deleted.

A close up of a logo

Description automatically generated

Figure 20 - Employee overview component

The employee page gives an overview of all employees and individual employee information can be view by clicking the view button beside the user. Upon clicking of this button, the system navigates to user and loads all their information based upon the users ID. Once on the individual employee page a tabular button navigation is located at the top of the page. During the original design sketch the tabs are located on the left however, in the actual design they are located at the top *Figure 21 and Figure 22*. This change in design was because after some experimentation and use of the system the tabs worked better and were more visual pleasing, if they were located at top of the page. Each tab will load the information regarding the individual employee and includes their general information, information regarding their shifts and holidays. As well as this an employee’s training, can be viewed and added. The training area is divided into the areas and other training. This was done as the area training corresponds to the areas within the rota creation area. Other training includes other training such as first aid or health and safety, that an individual employee may have completed.

A close up of text on a white background

Description automatically generated

Figure 21 - Purposed employee component design

The employee profile allows for the editing of the employee information. This is done through a form and the save changes button located at the button only becomes active if the system detects that any changes to the employee information have been made. Furthermore, as with all other forms in the system error messages will show if any input field is left blank, ensuring no null information will be inserted into the database. As well as this, to change the user status a drop displays the various roles in the system and if the user inputs information that does not match the created roles then an error will show and stop the role from being updated. Deleting a user will cause a pop up to open indicating that the user has been removed and the system navigates back to employee page in which the user has been removed and the employee count has been updated.

A screenshot of a cell phone

Description automatically generated

Figure 22 - Individual employee view

### 4.2.7 Rota creation component.

The rota creation component is used to create the weekly rotas. *Figure 24* shows the actual design of the page and *Figure 23* showsthe initial sketch. From the sketch and the actual design, it can be seen that there isn’t much difference. This is because the initial design was based upon other rota creation systems. As some users may have experience in using the other system, it was believed that the design should be as close to it as possible while providing similar features. As a result, it would aid in making the application easy to use, as a user will be familiar with how the functions work.

**A close up of text on a white background

Description automatically generated**

Figure 23 - Purposed rota creation component sketch

A screenshot of a social media post

Description automatically generated

Figure 24 - Actual rota component design

Upon loading of the page, the user is presented with the rota for the current day with the date being present at the top of the screen. Any shifts that have already been created for that day will be shown. Beside the date are two buttons which are used to move forward and back a week. Below this are the day buttons which are used to navigate between the days of the week. Upon clicking the date will be updated making it easy for the user to remember what day they are creating a rota for. Any created shifts will be shown and if no shifts have been created a message is displayed that no shifts have been created for the selected date. From user testing it was made clear that being able to navigate quickly through the days and the weeks enabled quicker rota creation.

The main area is divided into a number of smaller subsections. Each corelates with the various areas in which an employee might be assigned to work in during their shift. These areas were thought up from experience working in the industry as the designed areas are the main areas that employees are always assigned to work in.

The left-hand side shows the list of employees and they are dragged and dropped into a selected area. A drag and drop method enables simple copying of a user into an area. An icon is located to each employee’s name and once clicked will open a modal that contains useful information related to the user such as upcoming holidays. This aids in the creation of rota’s as it means that a user cannot assign a user to work a day in which they are unavailable to work.

Upon dragging a user into an area, the user is promoted to insert a start time and finish time. If these times are left blank an error message will show and shift cannot be allocated to the user, this ensures no blank information cannot be inserted into the database. If the times are correctly filled in, then the shift can be allocated to the user. Once allocated the user can edit the shift if any mistakes or changes need to made and can completely remove the shift by clicking the remove button. An error message will show if the selected employee is already allocated to work or if the user has requested the day off. This pop-up error is useful as it ensures that employees are not allocated to work multiple shifts for one day and means that a user will not be assigned a shift in which they are unavailable to work.

*Figure 25* shows a flow a diagram that indicates the logical steps that were needed in order to insert a shift for a selected employee. By following this earlier created diagram, it meant that the function could be designed accordingly.

A screenshot of a cell phone

Description automatically generated

Figure 25 - Flow chart for creating a shift

### 4.2.8 Timesheet component.

The time sheet component was designed so that employees past shifts and hours could be viewed and approved,*Figure 26* shows the end design. The original concept was to just have an area in which past shifts could be approved for payroll summaries. However, the end design now includes a function in which the hours can be edited or deleted. This was added because often staff members can work for a longer or shorter period of time that what they were originally allocated to work or removed if the employee doesn’t work their allocated shift. For example, an employee might work to 6 o’clock when they were allocated to work to 5 o’clock so the shift will need changed or the employee might be sick and not work. By adding this function, it meant that all hours can be recorded correctly. Furthermore, any unscheduled shifts for an employee can be added and approved.

At the top there is a week control and it displays the week in which is selected. This means that when using the component, the user doesn’t get lost and knows what date frame of shifts they are approving. The component also displays the total amount of shifts that need approval meaning that the user can quickly gauge how many needs approved or edited.

The users are located on the left-hand side and once a user is selected their shifts are displayed along with the associated shift approval count, the employee’s total hours and their total pay for the time period. The card design means that the shifts are separated with each providing information regarding the shift. Upon approval of the shift the card turns green and provides a striking difference in shifts that need approval vs shifts that aren’t approved.

**A screenshot of a cell phone

Description automatically generated**

Figure 26 - Actual timesheet component design

A close up of text on a white background

Description automatically generated

Figure 27 - Purposed report component sketch

### 4.2.9 Report component.

The report component was designed so that payroll reports could be viewed for all employees or an individual employee over a selected date range. The proposed design, seen in *Figure 27* doesn’t differ from the actual design, seen in *Figure 28.*  There is a start and end date picker that allows to select the date range. Employees are selected via a dropdown that lists all the employees and the “All” dropdown element will get the reports for all the employees. Upon clicking the *Get Reports*button it will load all the shifts for the selected employee and date range. Each individual shift is listed with all the information that is associated with the shift. Furthermore, there is *a total hour* and a *total pay* sum that provides a quick summary for the all the hours the employees have worked, and the total pay indicates the total pay for the selected date range. If the selected employee has no shifts for the date range, the system will show a message specifying that the selected user has no shifts for the selected date range.

A screenshot of a cell phone

Description automatically generated

Figure 28 - Actual report component design

# 5 Implementation

## 5.1 Login and logout.

### 5.1.1 Login.

A screenshot of a cell phone

Description automatically generatedThe first page in which a user is presented with is the sign in screen. Once the user has entered their details correctly (email and password), and clicks the ‘sign in’ button the *onLogin()* method located in the *signin.component.ts* is called, seen in*Figure 29*. If the sign in form is not entered correctly, for example, an input is field is left blank, the *onLogin()* method returns and the relevant error message is shown to the user. If the form has been entered correctly this method passes the entered email and password to the *login()* method inside the *auth.service.ts* as shown in *Figure 30.* The *login()* method makes use of Angular’s HttpClientModule to send the data from the form as the body of the request to the server URL endpoint as a POST method *(figure 30****:****31-33).*

Figure 29 - onLogin() method

A screenshot of a social media post

Description automatically generatedThe object from the front-end is passed onto the backend server in which uses the data to query the database, this can be viewed in appendix 5**.** The server *login()* method queries the database and finds an entry where the email is equal to the data sent from the body of the HTTP POST request. If there is a user with the email provided their password is compared against the password stored. As all the passwords that are found in the database are hashed to improve security a node library called Bycrpt is used to decode the stored password. This decrypted password is then compared against the password inputted by the user. If the email and password match the information stored in the database then the *login()* method returns a response containing the user’s ID number and the user’s status as a JSON object. However, if any information is incorrect or a user is not found a HTTP error (which will be discussed in more depth later), correlating to the corresponding error is returned. This response is then displayed to the user on the front-end in the form of an error message.

Figure 30 - login() method

Once the response is returned the *login()* methodinside the *auth.service.ts* stores the *userId* and *userStatus* in local storage in the browser (*Figure 30:37)*. While using the browsers local storage is sometimes considered bad practice, it was used in this case as the userId and userStatus have been used multiple times throughout the application, so they needed to easily accessible. This ease of access to items is an advantage of using local storage. Another storage method is to use a session. This method was decided against because it removes all data from storage if the tab is closed. By using local storage, it means that if the user accidently clicks off the tab they will not be logged out of the system and can return to the page and continue where they left off. In terms of security, the system does not store any sensitive information, such as passwords and emails, in the local storage so any malicious attempts to get users data have been reduced slightly.

The *login()* methodalso returns an observable in which the *onLogin()* method subscribes to (*Figure 29:29).* As the application has two user groups and each group has different functionality the status of the user is checked by using the data that has been subscribed to. If the user has the status of admin, they are navigated to the admin home page and the related functionality is loaded. However, if the user has the status of user the system navigates them to the user home page and the related user functionality is loaded.

### 5.1.2 Logout.

In order to log out of the application the user clicks the log out button located in the right of the navigation bar. By clicking this button it calls the *logOut() (Figure 31:78)* method located in the auth.service.ts. which in turn, removes all the items from the local storage and navigates the user back to the sign in screen.

A screenshot of a cell phone

Description automatically generated

Figure 31 - Logout() method

## 5.2 Authentication guard.

The created application has two types of users both with different functionality and permissions depending upon their status. If a user has the admin status, they have access to all the functionality, where as a general user does not. As this is the case some pages need to be protected so that only a user that has the authentication status of admin can gain access to these pages, while a general user is blocked from accessing these pages. In order to accomplish this Angular route guards were made use of and they are used to prevent users from navigating to parts of an app without authorization (Angular.io, 2020). This will explain how the admin authentication guard has been implemented, however, it should be remembered that a user guard has also been used within the system but as both guards follow the same process only one will be discussed in depth. *Figure 32* shows a typical flow of events with regards to the authentication guards within the system.

A screenshot of a cell phone

Description automatically generated

Figure 32 - Flow of events for authentication guards

A screenshot of a social media post

Description automatically generatedWhen a user logs into the system their details are passed to the admin authentication guard as seen in *figure 33:25* as an observable. The admin guard first checks to see if the user exists and if the user status is that of an admin. If this is the case the guard returns true. Inside the *app-routing.module.ts* file the admin guard is added to all the routes, for example, the rota creation page, that only a admin has access to by adding it to the *canActivate* property, as seen in figure 34:20. By adding the admin guard to the canActivate property it ensures that anytime someone tries to access a page in which they need admin status the application checks to see if they have the correct authentication status, (are logged in and have the admin status) through the admin guard before proceeding.

Figure 33 - Admin authentication guard

A screenshot of a social media post

Description automatically generated

Figure 34 - User and Admin routes with canActivate property

As mentioned early this is the same process for the user guard however, it has been used to allow access to general user pages by adding the created guard to all user routes. By using the authentication guards, it helps to protect routes and denies access to pages and features of the system if the user does not have the correct status. Furthermore, it also stops any user that is not a registered user from accessing and using the system. If an unregistered user enters the URL that is used for rota creation, “/admin/rota” they are denied access as first of all they are not logged in and secondly, they don’t not have the admin status. As a result, the system blocks their access as the authentication guard on that route returns false and returns them to sign in screen.

## 5.3 Employee management.

As the one of the main functions of the system is to provide an employee management. The *employee* component does this by providing a list of all employees and allows the user to create a new employee as well as editing and deleting existing employees. This section aims to provide an overview of this was achieved using the created the system.

### 5.3.1 Displaying all employees.

A screenshot of a cell phone

Description automatically generatedWhen the user navigates to the employees page it invokes Angular’s ngOnInit lifecycle hook that calls the *getEmployees() (Figure 34:28)* method in the *employee service* that makes an HTTP GET request and returns all the registered employees as an JSON object. The *employee-list* component then subscribes to the results and the items are stored in an employee object. Using a structural directive, *\*ngIf,* in the employee-list HTML template it checks to see if any employees exist. If no employees exist a message of “No employees yet” is displayed. However, if employees exist another structure directive, *\*ngFor,* binds to the HTML pages and loops through all the employees displaying each individual employee as a new row in the created employee table. As a result, the user can view all the employees registered to use the system as a list. Furthermore, a total of all employees is displayed to the user. The total is declared as *total* inside the component by setting it equal to the length of the employee array. This calculated value is then embedded into the HTML template by using interpolation.

### 5.3.2 Viewing an individual employee.

In order to view an individual employee, Angular’s event bindings were made use of. Event bindings respond to any DOM events and are triggered by user input (Angular.io, 2020). Beside each employee a “view” button has been placed that enables to the user to view the selected individual employee. The following syntax on the button is used – *(click)=”viewEmployee(user)”.* The click event on the button calls the *viewEmployee()* method and the user is passed to it (*Figure 35:42).* The *viewEmployee()* method then navigates to the view employee page in which the *viewEmployee* component loads all the selected employee’s information.

Figure 35 - Employee-list component with call to getEmployees() method and viewEmployee() method

The *viewEmployee* component parses the user’s id number from the URL by using the paramMap interface provided by Angular (*Figure 36:27)*. This component then subscribes to the *getEmployee()* method which sends a request to the server and returns all the users information based upon their id. This information is then used to display the employee’s profile, shifts, holidays and training.

By looking at the URL’s this process can be understood in more detail. When the user navigates to the employee-list component the URL is “/admin/employees”. If the user would like to view an individual employee, for example John McDonald, who has the user id of 32. They would click the view button and the system would navigate to “admin/employees/32”. Inside the view-employee component it would parse the users id from the URL, in this case 32, and load all the information associated with the user with the id number of 32.

A screenshot of a cell phone

Description automatically generated

Figure 36 - paramMap interface used in the viewEmployee component

One problem with passing the user id through the URL that was realised during user testing was that if the user manually entered a user id that didn’t exist the view employee page would not load. It was realised that this was because all the features on the page relied on the user id. If there was no user for the selected id, then the system could provide any information. This was changed and if a user manually enters a user id into the URL for a user that does not exist the page will load with a message stating “Sorry, a user for the selected id does not exist”.

### 5.3.3 Deleting an employee.

A picture containing knife

Description automatically generatedOnce the user has navigated to the view employee page, they have the option to remove an employee from the system. This feature was added as it allows for the removal of staff who no longer work for the company and therefore do not need to use the application. Event binding is again used but this time on the delete button. This event calls the *deleteUser() (Figure 37)* which makes a call to the employee service method of delete user that removes the user from the system. A window alert is used that shows the user that the employee has been successfully deleted and then navigates to the employee homepage. After the user has been deleted the employee list updates and the user is removed from the list and the employee total also decreases by one.

Figure 37 - deleteUser() method

### 5.3.4 Creation of an employee.

To create an employee the user clicks the create employee button located in the employee list component. This in turn opens a modal which contains a reactive form in which the user inputs the new employee’s information, such as name, email and date of birth. Reactive forms use an explicit and immutable approach to managing the state of a form at a given point of time (Angular.io, 2020). This helps to maintain the integrity of the model between changes as each change returns a new state. The creation of the form involved three main steps.

The first was to create a property and set it to a new form group instance. In this case a property of *employeeForm* was created and set to a new FormGroup instance. Individual form controls were added to the created form group (*Figure 38:29-36)* and the validators property were added to each control to ensure that they weren’t left blank by the user. If any input was left blank the system would display an error message below the respective input.

A screenshot of a cell phone

Description automatically generatedThe second step involved was to associate the FormGroup with the model and the view. Each input in the view was given a FormControlName that binds each individual input to the form control defined in the created FormGroup. The form controls communicate with their respective elements and store any inputs from the user.

Figure 38 - Employee form group

Finally the *create-employee* component in which the form group is located, listens for a submit event by the form and emits an *ngSubmit* event that is bound to the *onSubmit()* method which can be seen in *Figure 39:40.* This method then sends the user inputted form, values to the server via an HTTP POST request, that is called inside the *createEmployee* method located in the employee service. The new employee’s details are then stored in the database and if successful an alert is shown, the window reloads, and the new employee is added to the employee list. However, if unsuccessful, or if a user with the provided email, is already registered to the system, an error message will show corresponding to the error that has occurred.

*A screenshot of a cell phone

Description automatically generated*

Figure 39 - onSubmit() method used in the create-employee component

Upon creation of a user, an email is sent to the created user’s email address. Within this email it provides the new created employee with their password, that they can use to log into the application. The process of generating a secure password and sending it to the new employee via email is done within the server-side code, more specificity within the *createUser* method, located in the user-controller. The createUser method receives the inputted form values from the user as a JSON object. Using Generate-password, an external password generator, it produces a random password for the user. The user’s email is taken from the JSON object and using Nodemailer, an imported module that allows the application to send emails. By using this, it allows the generated password is sent to the user’s email. Upon receiving the email containing the password, the user can use it to log in and use A screenshot of a social media post

Description automatically generatedapplication. *Figure 40* shows the use of Nodemailer within the system.

Figure 40 - The use of Nodemailer to send an email to the created user containing their generated password

While the functionality of creating a user was created in one of the first sprints, the extra functionality of the system, to send an email containing a generated user password was implemented at a later stage. Before it was implemented the created user had to be verbally told their password. It was decided that this was not be the most secure or easy way to inform new users of their password, to gain access to application. As the password that is generated by the system produces a password that is 20 characters in length and contains a mix of letters in upper and lower case as well as numbers, verbally telling a new user a generated password would not be ideal. One incorrect character would mean that the user would not be able to gain access to the system. Furthermore, if one user had access to all the created user passwords it would mean that they could gain access to each individual employee’s information and could lead to possible unwanted changes being made to their profile, shifts and holidays.

## 5.4 Rota creation.

The rota creation component can be divided into 3 main functions, that all combine together to provide the designed functionality needed in order to create rotas. These functions are the date controls, the drag and drop of users into a selected area and inserting, edit and deleting individual employee’s shifts. This section will discuss each of these functions and how they combined in order to create a weekly rota for employees. Upon loading of the page, the user is presented with the list of employees on the left-hand side, the date controls that show the current at the top and the different areas located in the middle of the screen. Any shifts that have already been created for the current day are displayed in their respective areas. This is done by making an HTTP get request to the API that returns any shifts for the current date.

### 5.4.1 Date controls.

As stated before, upon loading of the rota creation component, the user is presented with the current date. A date pipe provided by Angular is used to make the date more readable for the user. For example, if the date pipe wasn’t applied the current date would be formatted as *“Thu Sep 03 2020 12:00:00 GMT+0100 (BST*)”. However, using the date pipe it is a displayed as *‘Thursday, September 3, 2020” and makes it much easier for the user to understand.* As discussed in chapter 4, there is controls that the user can interact with to change the week or day. By clicking the weekly controls an *onChange* event is emitted, that will either increase the displayed date by a week or decrease it by a week depending upon which button the user clicks*.* The date is bound to each area component as the property *selectedDate*. Within each area component a *selectedDate* property is declared and decorated with the @Input(). By decorating the property with input, it ensures that when a change is made, such as clicking the button to change the week the new selected date is updated within each area component.

#### Upon changing the day any created shifts for the selected day are shown to the user. By using the ngOnChanges lifecycle hook within each area component it detects that the data bound property (selectedDate) has changed. When a change has been detected, each area component calls the getShifts method (Figure 42) in the schedule-service that returns a shift object as an observable. Each area component then subscribes to the observable and any shifts for the new selected date are displayed via the view. Figure 41 shows the ngOnChanges lifecycle hook within the manager-area component and shows it calling the getShifts method by passing the new updated date to it. It also shows the component subscribing to the observable that the getShifts method returns.

#### A screenshot of a cell phone Description automatically generated

Figure 41 - ngOnChanges lifecycle hook within the manager-area component

As a JavaScript date is formatted as “Wed Mar 25 2015 00:00:00 GMT+0000 (Greenwich Mean Time)” it needed to be converted to the SQL date format (YYYY-MM-DD) to retrieve the shifts for the selected date. In order to convert the selected date a function was created (*Figure 43)* that converts the JavaScript formatted date into an SQL formatted date. The selected date was passed into the function and a variable of convertDate was set to the result (*Figure 41:67)*. This convertDate variable is then used in the getShifts method and allows for the database dates to be queried using the correct date format.

A screenshot of a cell phone

Description automatically generated**A picture containing knife

Description automatically generated**

Figure 42 - convert date function

Figure 43 - getShifts method

### 5.4.2 Drag and drop of employees.

In order to create a shift for an employee, the card which holds their name must be dragged and dropped into one of the 4 created ‘areas’. The ‘@angular/cdk/drag-drop’ module was used as it allows for easy creation of drag and drop interfaces *(Figure 44)*. The users and area components are wrapped within the cdkDropListGroup attribute that allows for items to dragged between multiple lists. A list of employees is located on the left-hand side of the window and are set as the cdkDropListData containing all the data for each employee. The area components listen to the drag event and when an employee is dragged and dropped into the selected area, the employee’s data is copied and inserted into the selected area list. For example, the user selects the user John King and drops it into the manager area. That information is then copied from the employee data list and inserted into the array, managerList, within the manager-area component.

A screenshot of a cell phone

Description automatically generatedUpon dropping the user into the selected area, the system checks to see if the user isn’t already allocated to work that day or that they haven’t requested the day off as a holiday. If either of these conditions are true an error will show at the top of the screen and remove the employee from the area. Two Rxjs behavioural subjects have been created with the schedule service. One behavioural subject has been created to show an error if the user is already allocated to work and the other is to show an error if the user has requested the day off. Within the schedule service both subjects have been set to false. However, when an employee has been dragged into an area and either of the conditions are true then the area component will change the value of the behavioural subject to true. As the area-error-handler component subscribes to both behavioural subjects any changes in the value will cause the corresponding error to display to the user. For example, if a user drags a user that has already been allocated to work, the area component will change the userExists variable to true and as a result the area-error-handler component will display the message “User is already allocated to work this day”.

Figure 44 - drag and drop function

A screenshot of text

Description automatically generatedIf one of the errors do occur the employee needs to be removed from the area so that they are not allocated to work. To remove the employee from the area, the JavaScript array filter method is used. First of a forEach loop is used to go through each of the shifts and the holiday for the employee. If the employee is already in the created area array the filter method removes the employee from the area (*Figure 45)*. The view then is updated and the employee is removed from the area.

Figure 45 - removal of user from managerList

### 5.4.3 Inserting, deleting and editing individual shifts.

When an employee is dragged into an area, a card with the employee’s name, a start time and finish time is presented. The user must insert a start time and finish time in order for the shift to be allocated. If the start and finish time are entered an HTTP POST request is sent to the API in which the shift is inserted, allocated to the selected user and added to the corresponding shift area array.

To delete a shift for an employee the user clicks the delete button and a HTTP DELETE request is sent to the API and as a result the shift is removed from the database. To remove the shift from the view, the filter array method is made use of again. Upon the click of the delete button the shift id is passed to the *deleteShift* method. The filter method finds and removes the shift based upon its id from the shift array and returns a new array that is displayed to the user in the view.

Editing an individual is done by clicking the edit button that allows for the start and finish times to become editable. The previous values are bound to the ngModel on the start and finish time inputs. This allows the user to see what the previous values were and therefore can update the times correctly. Upon clicking the save changes button the new form values are sent to the API as a PUT request and changes the values based upon the shift ID.

## 5.5 Routes and controllers.

Routing refers to how an application’s endpoints respond to client-side requests (Express, 2020). Routing is defined within the application using methods of the Express routing object that correspond to specific HTTP methods; for example *router.get()* to handle all GET requests. These routing methods specify a call back function and are called when a request is made to the specified end point. The application listens to requests made by the front-end and when a route is matched, it calls the function associated with that route.

Within the application separate routes have been set for different end points. For example, all routes that are associated with user have been included in a user-route file. Furthermore, call back functions, or handler function have been separated into controllers and placed into separate controller files. These methods have been exported and are called within the associated route. When the client side sends a request to the specified route the handler function is executed and returns the results of the request back to the client as a JSON object, in which the front-end manipulates this data and displays it to the user.

By walking through one example of how this process takes place within the application, it will explain how routes and controllers have been used. It should be noted that while this is only one example, routes and controllers have been set up throughout the application and allow the client to get, update and delete information.

In order to allow the user to view all employees the *employee-list* component makes a call to the *getEmployees* method. The *getEmployees,* inside the employee service makes use of Angular’s HTTP client to send a GET request to the URL http://localhost:3000/api/users/. Express listens to this this request and a get route within the user-routes file has been created to respond to this request. The handler-function, in this case *getUsers*, is executed (*Figure 46 and 47)*. The *getUsers* method uses Sequelize to query the database and returns the results from the database query as a JSON object (Figure 48:33). However, if no users were found or there was an error with the server it would respond with the corresponding message. If successful, the results are returned to the client and the process described in section *5.3.1* is carried out and all the employees are displayed to the user. This process of sending and responding to requests has been used throughout the created application and is the main driving force in how the user can interact and use functionality of the system.

A screenshot of a cell phone

Description automatically generated

Figure 46 - Route to return all users inside the user-route file

A screenshot of a cell phone

Description automatically generated

Figure 47 - getUsers function

## 5.6 Testing.

Testing was carried out throughout the development process and a number of techniques were used to ensure that the application worked as expected. As one of the seven principles of testing is that early testing saves time and money, testing was carried out right from the beginning of the project, in order to find as many defects as possible. By carry out testing early and often it meant that the time and effort regarding fixing the defects was greatly reduced. Use case testing was carried out and for each use case that was created during the design stage a number of tests were implemented when a feature was created. As the project followed the agile methodology and new changes to the system were being introduced, tests were derived from the new created use cases and regression testing ensured that when new features were added the application still worked as expected. Appendix 3 shows all the black box testing that was based upon the use cases that was carried out.

Angular’s *ng serve* feature was made use of, throughout the development stage and aided in testing. This CLI command builds or serves the application in real time. This enabled in testing as viewing and using any features, could be done in real time, with any defects to these features to be quickly spotted and corrected. As the *ng serve* command builds creates a new build every time changes are made to the source code it allowed these changes to be viewed and tested continuously.

With regards to the server-side an application called Nodemon was used, that motions any changes that are made and restarts the application when file changes in the directory are detected. This again allowed the motioning of changes and testing as new features were added to the application. An application called Postman was used that simulated calls to the API. For each route that was implemented in the application, Postman used to test the HTTP request and return the results. If the results returned by the test request did not match the expected result the server-side code was updated and retested until the expected result was returned. *Figure 48* shows a test HTTP requested and the returned results that was sent to the created end point http://localhost:3000/api/users/:id using Postman.

A screenshot of a social media post

Description automatically generated

Figure 48 - HTTP request sent to route using Postman

## 5.6.1 User acceptance testing

In the final sprints when the majority of the system functionality was implemented user acceptance testing took place. Users who have experience in using a rota building application and those who do not, were asked to participate in the testing. This provided important feedback, as it showed that the system was easy to understand and use when the user was a novice and had no experience in using such applications. Furthermore, from user testing it showed that the system was easy to navigate, pages didn’t not take more than 10 seconds to load and only registered users could use the system.

While, this type of testing could have taken place at an earlier date, it was believed that as not much functionality was implemented during the first sprints, those testing the system would not be as able to provide as clear feedback as possible it most of the features were missing. Those testing returned feedback verbally and any clear defects found find were noted and fixed. *Table 6* shows a selection of defects that were found in the system during user testing and the solution used to fix the defect.

It should be noted that user acceptance testing took place on the computer in which the system was developed using the localhost. Better user acceptance testing would have involved launching the application and getting users to test it using their own web browser. This would have meant that the system would have been exposed to a number of various environments and this would have meant more defects and improvements could have been made to the system. However, due to the time constraints of the project this wasn’t possible.

|  |  |  |
| --- | --- | --- |
| Page on which defect found | Description of defect | Solution to problem |
| Sign-in Page | Upon click of forgot password link the system is not navigating to forgot password page and showing 404 page when clicked | Set up router link for page that when forgot password link is clicked the system navigates the user to the forgot password page |
| All | User could enter URL manually and if not admin gain access to the admin functionality | Admin authentication guard was added to all admin routes and stops general users accessing admin functions |
| User Welcome Page | Total shift count for week did not update when week was changed. First loaded total shift count shown | Weekly shift count was added to the onChanges method that counts the total shifts for week when change is detected |
| User Welcome Page | When no one is working the information modal does not show any information | ngIf statement added to the view that outputs message if total shifts for that day are 0 |
| Rota Creation Page | User can insert characters instead of numbers into start and finish time inputs | Inputs changes so they are of type time |
| Rota Creation Page | Editing one shift cases all already created shifts to be edited to the same time | Shift ID was passed to the edit shift method |
| Employee Overview Page | User can manually enter userID in URL for user that doesn’t exist | Message added so that when a user does this the system will output a message saying user for selected ID does not exist |
| Individual Employee Page | Upon creation of training for a user the created training is not added to the view | The result returned from the HTTP call was added to the training array using the push method and spread operator |
| Individual Employee Page | Users role can be changed so it is not admin or user and as a result user can’t use system | Dropdown input added that only shows admin and user values. If input is not admin or user, the form regrets the input and displays a message saying user role is invalid |
| User Profile Page | Removal of holiday not removed until after page reload | Filter method was used to remove holiday from array so it would be removed from view |
| User Profile Page | Showing all upcoming and pervious shifts makes the profile page cluttered | Modal added to show all shifts upon button click. Upcoming shifts limited to 5 |
| Logout button | Logout button clicked and user navigates to sign in screen but user not removed from local storage | Clear auth data method created that removes all items from local storage when logout button is clicked |

Table 6 - Selection of defects found during user testing and the solutions

# 6 Evaluation

As defined in chapter two the aim of the project was to create a web application that makes employee management, rota creation and payroll management for hospitality managers easier. It can be seen that the created application provides all the functionality that was drawn up during the requirement elicitation stage. All the requirements that were given a “must have” or “could have” priority level where implemented and all the functionality based upon the user’s stories have also been implemented. Furthermore, from user acceptance testing it also showed that the non-functional requirements in terms of usability have also been met. Users said that the interface was easy to understand, and the design of the system and its features meant the it was easy to learn as a whole. Any problems that were found by users were fixed and meant the system become more reliable and increased the user’s confidence in the system. In terms of meeting the main aim of the project, it can be seen that the system meets those aims created at the start of the project, and as such can be considered a success.

As a whole the project can be considered a success, however, there are a number of downfalls or areas within the development of the system that could be improved. Given the time constraints associated with the project these downfalls may not have existed as more time could be given to solving them. Furthermore, the lack of experience in working with chosen technologies as this was the developers first time taking on a project of this size also introduced some weakness. In future development of a project of this size it will be implemented and designed differently due to the now required knowledge and experience.

More experience in using the Angular framework would enable the shared functionality of components to be separated into the created services. Instead, the majority of the component’s logic takes place with the component and should instead be placed within the designed services. The services have in the most part only been used for calling the API and making requests to the backend, as a result there is a large amount of code repetition within some components. Some components could share the same methods and functions but instead of placing them into services these methods have been repeated within each component that needs it. For example, the rota creation component has four separate areas, then manager-area, bar-area, floor-area and chef-area. Within these four components they all make use of the same methods and functions, however, these functions have been produced 4 times, when they only needed to be produced once. Looking back, it would have made sense to make use of Angular’s services and place all these methods into one service that each component could make use of. However, given that this knowledge was gained at a later date and due to the time constraints, this use of services wasn’t implemented and the already designed components were left untouched. Making use of the services would have reduced the amount of code and the size of the application and if future development of the project was to take place this is one of the main improvements that would be made straight away.

In terms of testing the application, only user acceptance testing, regression testing and testing of use cases was only carried out. More comprehensive testing would have included using the Jasmine test framework that is included with Angular upon each download and install. Unit testing of each module, component and service in project would have allowed more defects to be shown to the developer and as such, the system would lead to a much more reliable system. However, due to the aforementioned time constraints associated with the project, testing of this nature was not able to take place with only user testing being implemented. Again, in the future this is something that would be looked into and all modules, services and components would be tested to provide a more reliable system.

The way in which error’s that are returned from the server and dealt with by Angular or displayed to the user could also make use of some improvement. Upon each method call to the API the component that subscribes to the results handles the error. This again leads to a lot of duplicate code that could be reduced by introducing the ErrorHandler hook provided by Angular that creates centralised error handling. This error handler class would handle any errors returned in one global class that can be used by all components. However, this was not made use of instead each component handles its own errors and again this led to a lot of code duplication. As this error handler class was discovered at a later stage it was not implemented to the already existing code. Again, in any future development this feature will be made use of.

In terms of the technology picked to develop the application, it was the correct choice. The use of Angular allowed for an easy to use front end and through its features allowed for the creation of a dynamic web application. The backend server creation meant the functionality could be provided and all information associated with the application could be stored, edited and deleted. As all technologies shared the use of JavaScript in some capacity it meant the application could be quickly developed to a high standard. As experience in using the technology grew, more dynamic features could be added, and already existing features could be improved upon based upon user feedback.

While the application was only designed to be a web application to be used on computers an increased amount of web users are using a smartphone device, with mobile accounts accounting for approximately half of the web traffic worldwide (Clement, 2020). Using the application on a mobile device is near on impossible as many features become useable once the screen size goes below 400px in width. Some pages don’t even display the features if it detects that the screen size is below a certain size, instead it displays a message indicting that the feature is not available on a screen of that size. While the application was never designed to be used on mobile, the growing popularity of using mobile devices to browse the web means that any future development of the system should look at making the application mobile friendly.

Although there are downfalls in the created application, the project can still be considered as success as it meets all the user requirements. The downfalls exist in the organisation of the code and features that could have implemented however, which were discovered at a later date. These do not have any impact on the system itself and it still does what it was designed to do. Given more time, experience with the technology would develop and the problems shown would be changed. However, at this moment in time, the designed application meets the specification of creating a dynamic web application that makes rota creation, employee and payroll management easier. As well as this all the functional and non-functional requirements have been met, and as such, the project can be considered successful.

# List of external libraries used within application

Angular, version 9.0.7, [Accessed 1st September 2020]. Available from: <https://angular.io/>.

Bycrpt, version 5.0.0, [Accessed 1st September 2020]. Available from: <https://github.com/kelektiv/node.bcrypt.js#readme>.

Express-validator, version 6.6.0, [Accessed 1st September 2020]. Available from: <https://express-validator.github.io/docs/>.

Express, version 4.17.1, [Accessed 1st September 2020]. Available from: <https://expressjs.com/>.

Jsonwebtoken, version 8.5.1, [Accessed 1st September 2020]. Available from: <https://www.npmjs.com/package/jsonwebtoken>.

MDBootstrap, version 9.3.1[Accessed 1st September 2020]. Available from: <https://mdbootstrap.com/docs/angular/>.

Mysql2, version 2.1.0, [Accessed 1st September 2020]. Available from: <https://github.com/sidorares/node-mysql2#readme>.

Node, version 12.16.3, [Accessed 1st September 2020]. Available from: <https://nodejs.org/en/>.

Nodemailer, version 6.3.10, [Accessed 1st September 2020]. Available from: <https://nodemailer.com/about/>.

Nodemon, version 2.0.4, [Accessed 1st September 2020]. Available from: <https://nodemon.io/>.

Password-Generator, version 2.3.2, [Accessed 1st September 2020]. Available from: <https://www.npmjs.com/package/password-generator>.

Rxjs, version 6.5.5, [Accessed 1st September 2020]. Available from: <https://rxjs.dev/>.

Sequelize, version 6.2.3, [Accessed 1st September 2020]. Available from: <https://sequelize.org/master/index.html>.

# Bibliography

2009. *A Guide To Business Analysis Body Of Knowledge (BABOK Guide)*. Toronto, ON, Canada: International Institute of Business Analysis.

Angular.io (2020). Angular. [online] Available at: <https://angular.io/docs> [Accessed 17 August 2020].

Articles for Developers Building High Performance Systems. Available at: <https://blog.stackpath.com/web-application/> [Accessed 14 August 2020].

Britton, C. and Doake, J., 2005. *A Student Guide To Object-Oriented Development*. Amsterdam: Elsevier Butterworth-Heinemann, pp.21-38.

Clement, J., 2020. *Mobile Percentage Of Website Traffic 2020 | Statista*. [online] Statista. Available at: <https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/> [Accessed 14 September 2020].

D. Dunka, B., A. Emmanuel, E. and O. Oyerinde, D. (2018). Simplifying Web Application Development Using - Mean Stack Technologies. International Journal of Latest Research in Engineering and Technology (IJLRET), 04(01), pp.62–76.

Dickey, J. (2015). Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node.js. San Francisco, CA: Peachpit Press

Docs.microsoft.com. 2012. The MVVM Pattern. [online] Available at: <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN> [Accessed 17 August 2020].

Elrom, E. (2016). AngularJS SEO. Pro MEAN Stack Development, 197-219. doi:10.1007/978-1-4842-2044-3\_8

Gibb, R., 2016. What Is A Web Application? | How A Web Application Works. [online]

IBM Cloud Education, 2019. Mean-Stack-Explained. [online] Ibm.com. Available at: <https://www.ibm.com/uk-en/cloud/learn/mean-stack-explained> [Accessed 14 August 2020].

Joshi, B., n.d. Beginning Database Programming Using ASP.NET Core 3.

Lotfy, M. and Pyatt, K. (2008). The mean stack web application development platform: tutorial presentation. Journal of Computing Sciences in Colleges, [online] Volume 34(Issue 2). Available at: <https://dl.acm.org/doi/10.5555/3282588.3282602> [Accessed 14 Aug. 2020].

Nodejs. [2019]. Introduction to Nodejs. [Online]. [17 August 2020]. Available from: <http://www.nodejs.dev/learn>

Paradkar, S., 2017. *Mastering Non-Functional Requirements*. Birmingham: Packt Publishing.

Perrenourd, M. (2015). Learning web development with the MEAN stack

Rumbaugh, J., Booch, G. and Jacobson, I. 2005. The Unified Modelling Language Reference Manual. 2nd ed. London: Addison-Wesley

Sequelize.org. 2020. Manual | Sequelize. [online] Available at: <https://sequelize.org/master/manual/model-basics.html> [Accessed 17 August 2020].

Sommervile, I.,011. *Software Engineering.* 9th ed. London: Pearson.

Van Vliet, H., 2008. *Software Engineering: Principles and Practice.* 3rd ed. Hoboken, NJ: Wiley.

# Appendix 1 - Application structural design for admin and general users.

**A screenshot of a cell phone

Description automatically generated**

# Appendix 2 - Use Cases.

|  |  |
| --- | --- |
| **Flow of Events for viewing upcoming or past shifts** | |
| Actors | Admin/User |
| Objective | To view upcoming shifts |
| Precondition | Must be logged into the system |
| Main Flow | 1. Once the user has logged into the system, they are presented with their shifts for the current week 2. They can change the week view and view shifts for various weeks, both upcoming and past |
| Alternative Flows |  |
| Error Conditions | 1. If the API call to get the shifts the user fails an error message will show. |
| Post Conditions | User is presented will all upcoming or past shifts |

Table 7 - Flow of Events for viewing upcoming or past shifts

|  |  |
| --- | --- |
| **Flow of Events for viewing total hours for selected week** | |
| Actors | Admin/User |
| Objective | To view total hours to work for week |
| Precondition | Must be logged into the system and have been allocated shifts |
| Main Flow | 1. User logs into system and is present with weekly totals on user-dashboard based upon their allocated shifts for the week |
| Alternative Flows | - |
| Error Conditions | 1. If no shifts allocated, the total hours for week will be 0 |
| Post Conditions | User is present with total hours for selected week |

Table 8 - Flow of Events for viewing total hours for selected week

|  |  |
| --- | --- |
| **Flow of Events for adding a holiday** | |
| Actors | Admin/User |
| Objective | To add a holiday or date of unavailability |
| Precondition | 1. User must be logged into application 2. User must be on the profile page of the application |
| Main Flow | 1. User selects the + icon 2. Dropdown containing date input opens 3. User selects date of unavailability and adds holiday 4. The holiday is added to their upcoming holidays |
| Alternative Flows | 1. User cancels the addition of unavailability by returning not clicking the add holiday button |
| Error Conditions | 1. If user has already requested day off an error message will show indicating that they have already requested this day off and the day will not be added 2. If there is a server-side problem an error message will show and ask the user to try again later |
| Post Conditions | The requested day is added to the users upcoming holidays |

Table 9 - Flow of Events for adding a holiday

|  |  |
| --- | --- |
| **Flow of Events for deleting a holiday** | |
| Actors | Admin/User |
| Objective | To remove a holiday that has already been requested off |
| Precondition | 1. User must have already requested the day off 2. User is logged into system 3. User is on the profile page of the application |
| Main Flow | 1. User is presented with their upcoming holidays 2. User clicks the red trash can icon to remove selected holiday 3. Holiday is removed from their upcoming holidays |
| Alternative Flows | 1. User leaves profile page and does not click red trash can icon |
| Error Conditions | 1. Holiday is not removed, and error message will show indicating what went wrong to the user |
| Post Conditions | The user’s holiday has been removed from the upcoming holidays section |

Table 10 - Flow of Events for deleting a holiday

|  |  |
| --- | --- |
| **Flow of Events for viewing profile** | |
| Actors | Admin/User |
| Objective | For the logged in user to be able to view their profile containing their details |
| Precondition | 1. User must be logged into the system |
| Main Flow | 1. User navigates to the profile page 2. User is presented with all their information, e.g. personal information, holidays, training and shifts |
| Alternative Flows | - |
| Error Conditions | 1. User is presented with error message if their information fails to load |
| Post Conditions | User is presented with their profile |

Table 11 - Flow of Events for viewing profile

|  |  |
| --- | --- |
| **Flow of Events for editing general profile information** | |
| Actors | Admin/User |
| Objective | To edit general information such as edit address or change phone number |
| Precondition | 1. User must be logged into the system |
| Main Flow | 1. User navigates to profile page 2. User is presented with their information 3. User clicks edit button 4. Edit button opens modal containing form that shows user information 5. User edits the information 6. User clicks the save changes button 7. Changes are submitted and success message shows |
| Alternative Flows | 1. The user closes the modal and any changes made are not saved |
| Error Conditions | 1. If the user tries to submit the form with any mistakes the system shows the relevant error beside the incorrect input 2. The save changes button is disabled if any mistakes are made in the form 3. Error message will show if form is correct, but server-side has not updated changes. User will be asked to try again later |
| Post Conditions | Selected user information is updated, and user is presented with those changes |

Table 12 - Flow of Events for editing general profile information

|  |  |
| --- | --- |
| **Flow of Events for changing a password** | |
| Actors | Admin/User |
| Objective | To change the user’s password |
| Precondition | 1. User must be logged in 2. User must be on the profile page |
| Main Flow | 1. User clicks the edit button located next to the password on the profile page 2. Upon clicking the edit button an edit password modal opens 3. User enters current password and new password into respective input fields 4. User clicks save password button 5. Success message shows |
| Alternative Flows | 1. The user closes the modal and any changes made are not saved |
| Error Conditions | 1. Current password does not match entered current password error message shows and new password is not saved |
| Post Conditions | Users password is updated |

Table 13 - Flow of Events for changing a password

|  |  |
| --- | --- |
| **Flow of Events for creating a shift** | |
| Actors | Admin |
| Objective | To create a shift for an employee |
| Precondition | 1. User must have admin status 2. User must be on the rota page of application |
| Main Flow | 1. User drags a selected employee into an area 2. User enters start and finish time into input fields 3. Allocate shift button is clicked and shift is allocated to the selected user |
| Alternative Flows | 1. User drags user but leaves page so user is removed from selected area |
| Error Conditions | 1. Start and finish times are not entered, error message below inputs are show and shift not inserted 2. Employee has requested the selected day off, error message will show, and employee is removed from area 3. Employee has already been allocated to work on the selected day off, error message will show, and employee is removed from area |
| Post Conditions | Shift is created and allocated for selected employee |

Table 14 - Flow of Events for creating a shift

|  |  |
| --- | --- |
| **Flow of Events for editing a shift** | |
| Actors | Admin |
| Objective | To edit an already existing and allocated shift |
| Precondition | 1. User must have admin status 2. User must be on the rota page of application 3. Shift must already be created and allocated |
| Main Flow | 1. User selects shift they want to edit by clicking the edit button 2. Start and finish time become editable 3. Times are changed and save changes button is clicked 4. Changes are saved, start and finish time are no longer editable |
| Alternative Flows | 1. User clicks the edit button but leaves page, no changes are made 2. User drags another employee into area. Content is still editable |
| Error Conditions | 1. Either start or finish time are not inserted, error message below inputs will show |
| Post Conditions | Created shift is edited |

Table 15 - Flow of Events for editing a shift

|  |  |
| --- | --- |
| **Flow of Events for deleting a shift** | |
| Actors | Admin |
| Objective | To remove a created shift |
| Precondition | 1. User is admin 2. Shift already been created |
| Main Flow | 1. User clicks the delete button beside the shift in which they would like to delete 2. Selected shift is removed |
| Alternative Flows | 1. User doesn’t click the delete button |
| Error Conditions |  |
| Post Conditions | Shift is removed |

Table 16 - Flow of Events for deleting a shift

|  |  |
| --- | --- |
| **Flow of Events for viewing all employees** | |
| Actors | Admin |
| Objective | To view all employees |
| Precondition | 1. User must be on employee page |
| Main Flow | 1. User navigates to the employee page 2. All employees are shown to user |
| Alternative Flows | - |
| Error Conditions | 1. Error message will show if no users are in the system or if the server-side does not return any |
| Post Conditions | User will be presented with a list of all employees |

Table 17 - Flow of Events for viewing all employees

|  |  |
| --- | --- |
| **Flow of Events for creating an employee** | |
| Actors | Admin |
| Objective | To create an employee |
| Precondition | 1. User has admin status 2. User is on employee page |
| Main Flow | 1. User clicks on the create employee button 2. Modal opens with form for creating user 3. User enters fields and clicks create employee button 4. Alert saying user has been created and returns to employee list 5. Employee added to the employees and total employee count updated |
| Alternative Flows | 1. User closes modal and any input fields are removed, and user is not created |
| Error Conditions | 1. Any fields inputted incorrectly or left blank will show corresponding error message and create employee button will be disabled – employee will not be created 2. Server-side error will show error and employee will not be created |
| Post Conditions | Employee is created and added to the employee list |

Table 18 - Flow of Events for creating an employee

|  |  |
| --- | --- |
| **Flow of Events for viewing employee information** | |
| Actors | Admin |
| Objective | To view an individual employee’s information |
| Precondition | 1. Must have the status of admin 2. Must be on the employee list page |
| Main Flow | 1. User clicks on view button located beside an employee name 2. System navigates to the specific employee page and loads employee’s general information, holidays, shits and training |
| Alternative Flows | 1. User navigates away from employee list page 2. User enters employee ID number in URL |
| Error Conditions | 1. If user with inputted ID does not exist an error message saying “sorry, a user for the selected ID does not exist |
| Post Conditions | All information regarding the selected user loads in view |

Table 19 - Flow of Events for viewing employee information

|  |  |
| --- | --- |
| **Flow of Events for editing employee information** | |
| Actors | Admin |
| Objective | To edit an individual employee’s information |
| Precondition | 1. An individual employee must be selected |
| Main Flow | 1. User selects an employee 2. System navigates to employee information 3. User enters details into form in which they want to change 4. User clicks save button and changes are updated in the view 5. Success message will show |
| Alternative Flows | 1. User does not make any changes and navigates away from page 2. User does not click the save changes button and any changes made are not saved |
| Error Conditions | 1. Any input fields not filled in correctly or left blank will cause an error message corresponding to the error 2. Server-side error will show if any changes are not inserted to database |
| Post Conditions | Individual employee’s information is updated |

Table 20 - Flow of Events for editing employee information

|  |  |
| --- | --- |
| **Flow of Events for deleting an employee** | |
| Actors | Admin |
| Objective | To remove a selected employee |
| Precondition | 1. Employee must exist 2. User is on the selected employee view |
| Main Flow | 1. User clicks the delete button 2. Alert pop up that notifies the user that the selected employee has been deleted 3. System navigates back to employee list view where deleted employee has been removed from employee list |
| Alternative Flows |  |
| Error Conditions | - |
| Post Conditions | Employee has been deleted and user is returned to employee list |

Table 21 - Flow of Events for deleting an employee

|  |  |
| --- | --- |
| **Flow of Events for approving employee’s shift hours** | |
| Actors | Admin |
| Objective | To approve an employee’s hours |
| Precondition | 1. Employee must have worked a shift 2. User is on the timesheet page |
| Main Flow | 1. User clicks on an employee and shifts for selected week are shown 2. User clicks on approve hours and shift is turned green and added to the approve shift array |
| Alternative Flows | 1. User doesn’t approve the selected shift |
| Error Conditions | - |
| Post Conditions | Employee’s shift hours have been approved and counts updated |

Table 22 - Flow of Events for approving employee’s shift hours

|  |  |
| --- | --- |
| **Flow of Events for editing employee’s shift hours** | |
| Actors | Admin |
| Objective | To edit an employee’s shift hours |
| Precondition | 1. Employee must have worked a shift 2. User is on the timesheet page |
| Main Flow | 1. User selects employee from list and all shifts for selected week are shown 2. User clicks the edit button on the selected shift that needs editing 3. Modal is opened and changes to start or finish time are made 4. Save changes button is clicked and shift is updated with success message showing 5. Modal is closed and counts are updated based upon time edited |
| Alternative Flows | 1. User opens modal but doesn’t make any changes, last known times are saved |
| Error Conditions | 1. Any input fields not filled in correctly or left blank will cause an error message corresponding to the error |
| Post Conditions | Selected employee’s hours for selected shift are edited |

Table 23 - Flow of Events for editing employee’s shift hours

|  |  |
| --- | --- |
| **Flow of Events for deleting a user’s shift** | |
| Actors | Admin |
| Objective | To delete a selected employee’s shift |
| Precondition | 1. Employee must have worked a shift 2. User is on the timesheet page |
| Main Flow | 1. User selects employee from list and all shifts for selected week are shown 2. User clicks the edit button on the selected shift that needs deleted 3. Modal is opened and delete button is clicked 4. Shift is removed, modal closed and view updates |
| Alternative Flows | - |
| Error Conditions | - |
| Post Conditions | Selected employee’s shift is removed |

Table 24 - Flow of Events for deleting a user’s shift

|  |  |
| --- | --- |
| **Flow of Events for adding an unscheduled shift for employee** | |
| Actors | Admin |
| Objective | To add an unscheduled shift for a selected employee |
| Precondition | 1. User is on timesheet page and has selected an employee for which they want to add an unscheduled shift |
| Main Flow | 1. User selects employee and clicks add unscheduled shift 2. Modal containing form is opened 3. Form is completed with necessary information 4. User clicks the add button and shift is added to employee’s shift |
| Alternative Flows | 1. User clicks the close button on the modal any changes added are not saved and shift is not added |
| Error Conditions | 1. Any input fields not filled in correctly or left blank will cause an error message corresponding to the error |
| Post Conditions | Unscheduled shift is added for an employee |

Table 25 - Flow of Events for adding an unscheduled shift for employee

|  |  |
| --- | --- |
| **Flow of Events for creating payroll reports** | |
| Actors | Admin |
| Objective | To create a payroll report |
| Precondition | 1. User must be of status admin 2. User must be on reports page |
| Main Flow | 1. User inputs date range using date inputs 2. User selects “all” or an individual employee 3. The button get reports is clicked 4. Information for each shift in date range is present in a table as user 5. Counts for date range are shown at the top of the screen |
| Alternative Flows | - |
| Error Conditions | - |
| Post Conditions | User is presented with shift summaries for selected date range and employees |

Table 26 - Flow of Events for creating payroll reports

# Appendix 3 - Testing.

Table 27 shows test carried for each use case. It should be noted that 27 tests where ran under conditions that were always correct, or a ‘happy path’ and would cause no errors to how. Table 28 conditions were inserted incorrectly to test if corresponding errors were shown.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Function to test | Pre-Conditions | Procedure | Expected Result | Status |
| 1 | Logging into system. | User is registered | From the Sign in page the user enters details and clicks sign in. | User will be logged into system and taken to the related dashboard. User ID and status will be stored in local storage. | Passed |
| 2 | Logging out of system. | User must be logged in to system. | User clicks the logout button located in the navbar. | User will be logged out of system and returned to *sign in screen.* All user details will be removed from local storage. | Passed |
| 3 | Reset Password. | User must be a registered user. | User navigates to *forgot password* screen and enters details and then clicks the reset password button. | Users password will be reset, and alert will show success message. User will be returned to *sign in* screen. | Passed |
| 4 | View upcoming shifts. | User is logged into system. | User navigates to *user-dashboard.* | Upcoming shifts for the current week will be presented to user after system makes API call to return all shifts for user. | Passed |
| 5 | View user profile. | User navigates to profile page. | User navigates to profile page which makes a call to the API to return user information based on the user id. | User is presented will all their information. | Passed |
| 6 | Edit user information. | User is on *profile page.* | User enters the details in which they want to change. | New information is stored in database and success alert is shown to user with new information being updated in the view. | Passed |
| 7 | Edit password. | User is on *profile page.* | User enters old and new password. | Password is changed and success message is shown to user. | Passed |
| 8 | Add holiday. | User is on *profile page.* | User selects date from date input and clicks add holiday. | Holiday is added for user and added to future holidays. | Passed |
| 9 | Remove holiday. | User has an upcoming holiday. | User clicks the ‘trash-can’ button to remove upcoming holiday. | Holiday is removed for user. | Passed |
| 10 | Insert shift for employee. | User is on the *rota* page. | User drags selected employee and places into area. Start and finish time entered, and allocated shift button clicked. | Employees shift is inserted and add to area array. | Passed |
| 11 | Edit a shift. | User is on the *rota* page and shift has already been created. | Users clicks edit button for selected shift and is presented with form for start time and finish time. User changes times and clicks save. | Time is changed and shift information is updated. | Passed |
| 12 | Create an employee. | User is on *employee* page. | User clicks create employee button and enters details for new employee and clicks create employee button. | Employee is created and added to employee list. Email is sent to new employees inserted email address with a generated password. | Passed |
| 13 | View individual employee. | User is on *employee* page. | User clicks view button beside employee name. | Individual employee information is loaded based upon employee id. | Passed |
| 14 | Edit employee information. | User is on an individual employee’s profile page. | User edits user’s information and clicks the save button. | Updated information is updated on view and stored in database. | Passed |
| 15 | Delete an employee. | User is on an individual employee’s profile page. | User clicks delete button. | Selected employee is removed, system shows an alert and returns the user to the employee list page were employee has been removed and total employees is updated. | Passed |

Table 27 - Tests carried out for each user story

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Function to test | Pre-Conditions | Procedure | Expected Result | Status |
| 1 | Logging into system. | User is registered. | User enters incorrect log in details. | Error will show corresponding to which detail is incorrectly inputted. | Passed |
| 2 | Reset Password. | User must be a registered user. | User navigates to *forgot password* screen and enters details wrong details and clicks reset password button | Users password isn’t reset and error message will show. | Passed |
| 3 | Edit user information. | User is on *profile page.* | User enters details incorrectly or leaves out an input field | Save changes button is disabled and information is not updated | Passed |
| 4 | Add holiday user has already requested off. | Holiday date already requested off. | User selects date that they have already requested off. | Error message shows saying that they have already requested the selected day off and holiday is not added. | Passed |
| 5 | Error message showing user has requested day off. | Employee has requested day off. | Employee dragged into area on day they have requested off. | Error message will show saying they have requested the day off and shift will not be inserted. | Passed |
| 6 | Error message showing user is already allocated to work on selected day. | Employee is already allocated to work on the selected day | Employee dragged into area. | Error message will show saying the user has already been allocated to work on the selected day and shift will not be inserted. | Passed |
| 7 | Creation of employee error message. | User is creating an employee. | User enters details of new employee incorrectly or leaves input fields blank. | Create employee button is disabled and related error messages will show. | Passed |
| 8 | User training creation | User is on the training tab. | User enters input that doesn’t match created area. | Error message stating that the area is not a created area and does not insert training. | Passed |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Function to test | Pre-Conditions | Procedure | Expected Result | Status |
| 16 | Approve employee’s shift. | User is on timesheet page. | User selects employee and shifts for selected week will show. Any unapproved shifts will show. User clicks approve hours. | Employee’s shift will be approved, and details will be display in a green card. | Passed |
| 17 | Add unscheduled shift for employee. | User is on timesheet page. | User selects employee and clicks add unscheduled shift button. Adds the shift for employee. | Unscheduled shift will be added for the user and will show in the view. | Passed |
| 18 | Edit employees shift | User is on timesheet page. | User selects employee and clicks edit button for selected shift. User updates shift information. | Shift is edited and changes will be shown in the view. | Passed |
| 19 | Add employee training. | User is on individual employee’s profile. | User clicks add training. Inserts training for user. | Training is added to selected user and view updates to show that training has been added. | Passed |
| 20 | View pay-roll reports | User is on payroll reports page. | User selects date period and employees. | All shifts for selected employee/employees for date range will show with all information in table. | Passed |

Table 28 - Tests carried out that would show errors

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Function to test | Pre-Conditions | Procedure | Expected Result | Status |
| 9 | Inserting shift with no start or finish time. | Shift is inserted into area. | User tries to insert shift with no start or finish time. | Error will show corresponding to which detail is incorrectly inputted and shift will not be allocated. | Passed |
| 10 | Editing a shift with no start or finish time. | Shift has been allocated. | User tries to edit shift with no start or finish time. | Error will show corresponding to which detail is incorrectly inputted and shift will not be edited. | Passed |

# Appendix 4 – Supervisorial Meetings.

## Meeting 1 14/05/2020

**Meeting Time: 13:00**

**Other Attendees:** N/A

**Key Points Arising from Meeting:**

* Discussed the problem specification and key points regarding it.
* Looked at the timeline of the project organised key dates.
* Discussed the project requirements.
* Discussed the main features the system should include.
* Focused on what technologies could be used to create the application.
* The main focus should be on creating the problem specification.

**Next Action as a result of the Meeting:**

* Write the problem specification for submission on the 22nd May
* Continue to learn/improve skills with JavaScript, Angular, Node and Express in preparation for development.
* From the project requirements create a product backlog and organise into achievable sprints.
* Begin development of system starting with areas of a high priority level.

## Meeting 2 28/07/2020

**Meeting Time: 10:30**

**Other Attendees:** N/A

**Key Points Arising from Meeting:**

* Discussed the progress of the system so far based on timeline laid out in the problem specification.
* Discussed the how to complete the report and what needed to be included in key chapters.

**Next Action as a result of the Meeting:**

* Continue working to complete the working system so that it is at a standard to have a demo of the system.
* Continue to work on the report.

## Meeting 3 11/08/2020

**Meeting Time: 12:00**

**Other Attendees:** N/A

**Key Points Arising from Meeting:**

* Demo of completed software was shown and one point about creation of a payroll summary page was discussed.
* Discussed the current timeframe and what work still needed to be completed with regards both the application and the report before the deadline of 18th September.

**Next Action as a result of the Meeting:**

* Implement payroll summary reports for all and individual users.
* Start use case testing and user acceptance testing and use any feedback to fix any defects or improve system.
* Complete report.

# Appendix 5 – Login method.

A screenshot of a cell phone

Description automatically generated

# Appendix 6 - GitHub repo link.

<https://github.com/isaacbarr/Rota-App>