MemberMe
Outdoor Activity Membership System & Application

Scott Coyne

Yvonne Grealy

B.Sc.(Hons) in Software Development

April 17, 2017

Final Year Project

Advised by: Mr. Martin Hynes

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

Table of Illustrations

Chapter 1

About this project

Abstract

There are two main problems associated with outdoor activity centres. The first is the lengthy process of check-ins at these businesses that require customers to sign indemnity forms upon each arrival. The second problem is the high element of danger associated with certain activities and members involved in accidents not being able to communicate efficiently with staff of the business.

This project sets out to address these issues by creating a membership system for outdoor activity centre businesses. The system is broken up into two main components, a web application for the business and a mobile application for the members.

The aim in solving the first problem is to have a system where members can use their mobile phone to act as a membership card. Members would be required to sign up only once at the activity centre and then download the Mobile Application. From that point onward, they would only be required to present the Mobile Application on check in, which the business can scan into the Web Application upon arrival.

The second problem will be addressed by providing SOS functionality in the Mobile Application that will allow members to send their geolocation, emergency status and message to the associated business via the Web Application, alerting staff to where and how the member is injured.

During the design and implementation of this system, this project seeks to create a robust, well thought-out, simple and effective system that will help resolve the two problems mentioned above.

Authors The authors of this project are Scott Coyne and Yvonne Grealy, two fourth year students studying for a Bachelors of Science Honours Degree in Software Development at Galway-Mayo Institute of Technology.

Acknowledgements We would like to acknowledge and thank our supervisor Mr Martin Hynes for the time and effort he put into helping us throughout this project. We would also like to thank all our lectures at Galway-Mayo Institute of Technology for equipping us with the skills and knowledge in the past 4 years that has enabled us to complete this project.

Finally, we would like to thank Bike Park Ireland for testing the software and for the kind feedback we received from them.

Chapter 2

Introduction

This chapter will outline the context and scope of the project. It will explain the objectives for the project and the specifications for the software. A summary of each of the following chapters is provided. It also contains a section on the version control repository including descriptions and resource URLs.

The idea for this project started off as solution to a problem posed at a specific outdoor activity business. The issue was that each customer was required to fill in an indemnity form upon arrival before the activity started. Busiest check in times were early morning and lunch time, which was when sessions would start and finish. Staff were overwhelmed with the number of customers arriving at once and the time taken to process all customers was lengthy.

The solution proposed to solve this problem was the creation of a membership system. This system would mean that customers would only be required to sign up once and from that point on would be fast-tracked on arrival at the business using a check-in system. This would allow activities to get underway in an efficient manner and free up staff for other work.

Another problem encountered at the business was as a result of the high risk activities members participated in. Members were often getting hurt and the only way of communicating back to staff was if someone in the surrounding area travelled back to base and reported the accident. The solution to this was to add SOS functionality to the Mobile App that can send messages to the related business. The manner in which this solution was approached is described in the Objectives section with specific functionality detailed in the Specifications section.

Context

This section discusses the objectives set out in this project and also the different components of the project.

2.1 Objectives

The objectives of this project are:

- To create a membership system consisting of:
  - An administrative Web Application (Web App) for business users
  - A Mobile Application (Mobile App) for customer users
  - A Database for storing business and customer users data
  - A Server to act as an intermediate between the applications and Database
  - A messaging service to handle communication between business and members

2.2 Specifications

The specifications for the system components are outlined below:

2.2.1 Web App Specifications

The Web App should contain the following features:

- Create new business user
- Log in
- Create new members
- Generate temporary passwords
- Update existing members
- Reset member passwords
- Delete members
- Check-in members

- Receive SOS messages from members
- Display analytics
- Logout

2.2.2 Mobile App Specifications

The Mobile App should contain the following features:

- Log in
- Generate unique QR code
- View personal details
- Edit personal details
- Get current geolocation
- Send SOS messages to business
- Logout

2.2.3 Database Specifications

The Database should:

- Store business user data
- Store customer user data
- Store relationships between business and customers

2.2.4 Server Specifications

The Server should:

- Provide secure routes
- Authenticate business and customer users
- Send valid requests to the database
- Receive and process database responses
- Receive and process application requests
- Return responses to applications

2.2.5 Message Server Specifications

The Message Server should:

- Receive and process requests from members
- Receive ”subscription” request from businesses
- Return notifications to business upon receiving messages

2.3 Chapter Summaries

The following is a summary of each chapter in this report.

Introduction

This chapter contains the context for the entire project covering what the project is about, what the objectives are and the location and different elements of the GitHub Repository.

Methodology

This chapter describes the way the project was approached and managed. It also gives a description of how the project was researched, developed and tested.

Technology Review

This chapter discusses the technologies that were researched and used in the project. It gives an insight into why the technologies used were chosen and what alternatives could have been used.

System Design

This chapter gives an in-depth explanation of how the entire system architecture was designed and how it all connects together. Diagrams and screenshots are provided to further explain each individual element of the system.

System Evaluation

This chapter evaluates the project upon completion and highlights where the project could be improved upon. It discusses the testing that was carried out throughout the project and the impact that had on the development.

Conclusion

The conclusion gives a summary of the findings, outcomes and experiences having completed the project.

2.4 GitHub Repository

The GitHub repository for this project can be found at https://github.com/codevonnie/fourthyearproject. The sections below describe the different components in the repository and a link to each part.

ReadMe

This contains a brief introduction and a description of each component of the system with links to each.[3] This section can be found here.

Main Server

This contains a description of the Main Server and instructions on how to run it.[4] This section can be found here.

Message Server

This contains a description of the Message Server and instructions on how to run it.[5] This section can be found here.

Web App

This contains a description of the Web App and instructions on both how to navigate to the live site and to run it locally.[6] It also provides test log in credentials. This section can be found here.

Mobile App

This contains a description of the Mobile App and instructions to run it locally as well as a downloadable APK.[7] It also provides test log in credentials. This section can be found here.

Chapter 3

Methodology

This chapter will review the approach that was taken to develop this project. The methodology that was adopted will be described including how it was implemented and how it shaped the development process. It will also cover information on the structure and frequency of meetings. The reader will be given a good overview of the entire development lifecycle of the project from the research stages right up to the completion of the software.

The methodology used for this project was Scrum which is an agile and iterative way of managing a project. This methodology involves sprints, a predetermined time period in which, planned tasks are carried out and ends with a review of the work that has been done. It is a continuous process where each iteration is followed by another.

The beginning of a sprint consists of a planning meeting where the team met to discuss what work needed to be done, what tasks each team member would carry out and a time frame of when the tasks would be completed. There were daily scrums where the team met to informally discuss the work being carried out on the sprint. There were also weekly meetings with the project supervisor who acted as a ScrumMaster and helped with the planning and time estimates for sprints. These meetings also acted as the sprint review at the end of each sprint where the team explained or demonstrated the new functionality added to the project in the sprint. The following sections outline the sprints that made up the development process for this project.

3.1 Sprint 1

The first sprint was used to decide on the architecture for the system and to research different technologies that could be used for each part. Each team member took a different part of the architecture to research and constructed a shortlist of possible technologies for each element of the system. From this shortlist, these technologies were researched more in-depth to try to decide on the technology that would ultimately be used. This in-depth research ensured that the possible technologies would be compatible to use together and that they would complement each other.

The research carried out in this sprint ranged from reading the documentation for each technology, reading reviews, articles and forum discussions. Tutorials were looked at to get an idea of what the technologies were like and how they would suit the scope of the project.

The different elements of the system that were researched were the Mobile Application, the Web Application, the Servers and the Database. At the review phase of this sprint the technologies that would be used were chosen. More information on why each technologies used in the project were chosen can be found under the Technology Review chapter.

3.2 Sprint 2

Tasks were divided between the team members in the second sprint. One of the first tasks was to set up a version control repository for the project which was done using GitHub. Each different part of the architecture was set up as a separate branch so that tasks could be carried out in parallel. The issue tracker was used to communicate bugs and things that needed to be changed between team members.

The project was divided out as follows:

- The web application was to be developed by Scott.
- The mobile application was to be developed by Yvonne.
- The server was to be worked on by both team members with Scott working on the routes for the businesses and Yvonne working on the routes for the users.

- The database was set up and maintained by both team members.

The requirements for the system were determined in this sprint including what information would be stored in the database for businesses and users. Also, an overall idea of what functionality each part of the system should have.

A free account was set up on Heroku for the project and a blank application. A GrapheneDB addon was added to the application for access to the database and Cloudinary for the image storage.

A basic Express.js server was built locally using npm to install dependencies. A Procfile was created for it and it was deployed to Heroku which was then connected to the database using a Neo4j bolt driver.

Simple routes were set up to create a new person and a new business on the database. These routes were tested using Postman which allowed the database to be queried by sending requests to the server and returning the responses. Test data was used without validation e.g. email addresses were not checked to be the correct format at this stage. The JavaScript console.log() method was also used to print some requests and responses to and from the database to the developer console. Once this testing proved successful more routes were created that would be required. New functionality for the server was first tested on localhost and when something new was tested successfully it was added to GitHub which automatically deployed the newest version of the server to Heroku.

3.3 Sprint 3

Storyboarding was done at this stage to design the pages of both the Mobile and Web Applications and the navigation between the pages. Once the storyboards were complete, the development of the applications began.

The Mobile App was created in Visual Studio Code and npm was used to install Ionic and all necessary dependencies. Following the design from the storyboard, the basic pages of the application were implemented and connected. Testing of the Mobile App was carried out iteratively, in the early stages, locally on Google Chrome browser. Controllers and Services were created to send HTTP requests to the server and to handle the responses and push data to the app pages.

The Web App was created in Visual Studio as a blank ASP.NET application. Once the basic setup and structure was in place, research was carried out to find the best approach to creating the MasterPage(Default) layout that the Web App would follow. Bootstrap was found to be an appropriate solution, providing a template that was amended as necessary to suit the scope of this project.

Log in pages for both the Mobile and Web App were developed. The Mobile App log in was for member users and the Web App log in was for business users. The Web App also allowed for the creation of new business users.

Initially, two separate routes were set up on the server to handle these log in requests. Once Auth0 JWT authentication was added to both applications and to the server, these two routes were amalgamated into one that would use one of two requests depending on the type of user that was logging in.

3.4 Sprint 4

An issue was realised that dates could not be saved properly into the database because Neo4j does not support them and so dates could not be correctly queried. It was decided to convert all dates to milliseconds prior to saving them to the database which could then be queried successfully. When dates were received from the database to either application, the dates were converted from milliseconds back to date format.

The profile page was set up on the Mobile App which was populated with user data once the person had logged in successfully. Certain fields of this page were made editable and a route was set up on the server to allow updates to the users entry on the database.

The Add new members page was also setup on the Web App. Here, businesses could add new members to their database by filling out the ”Create New Member” form. Once the form was validated and a profile image was chosen, the images were then sent to Cloudinary. When the images were successfully stored on Cloudinary, the Https URL was then stored in the database along with the users profile.

The Map page on the Web App was setup in this sprint, with an embedded Google map at the centre of the screen. Testing was carried out to evaluate the possible integration of Google Maps in an ASP.NET Application.

3.5 Sprint 5

In this sprint, the SOS system was developed for both applications. Google Maps was chosen as it is well known and would be familiar to both the Mobile and Web App users. Also, Google Maps was easy to implement in both applications using a JavaScript script which is declared in the HTML of both applications.

The message server and simple routes were set up for the Mobile App to send requests to, and for the Web App to Connect to. These routes were tested using Postman originally, which allowed testing of the routes Request/Responses from the server. Once Testing had finished, development began on the SOS functionality on both Applications.

The SOS page of the Mobile App was created which displays a Google Map with the users location highlighted with a marker. Other information is also gathered about the users situation and sent to the message server. To restrict users from sending messages to the system by accidentally, confirmations and button disabling was implemented.

The Map page on the Web App, which was created in a previous sprint, was modified to accept SOS messages from the Messaging server. Messages were displayed on the Google Map as a Pin using the geolocation provided in the message. Each Pin has Click/Tap functionality which displays the SOS message from the associated member.

In the original storyboarding, it had been decided that a Settings page would exist on the Mobile App which would include a Logout button. At this stage of development, it was realised that this page was unnecessary and was removed in favour of having a standalone Logout button that could be accessed on any page of the application.

3.6 Sprint 6

This sprint involved setting up the Arrivals and Dashboard pages in the Web App with all necessary functionality. Emphasis in this sprint was put on GUI design, splash screens and logos for the Mobile App.

The Arrivals page on the Web App was implemented in this sprint. This takes input from a QR code scanner or user email, querying the data base for a valid member and returning the members profile. Update and delete functionality of member profiles was also added.

The Dashboard in the Web App was the last page to be developed as this is where the analytics about members is displayed. Analytics include Top Visitors and Busiest Months that are obtained from the database. Test members were added to the database, all with different join/visited dates and data. With this test data, queries designed on the Web App were used.

3.7 Sprint 7

This was the final sprint in the development of the system and it concentrated on validation and testing. Validation was added for all inputs in both the Web and Mobile Apps to ensure only valid data can be inputted by the user. For the Web App, Bootstrap Validator was used to validate all client side input and ASP.NET built-in validation was used as a backup for both client and server validation. Input validation in the Mobile App was done using Angular validation directives.

Testing of authentication, routes and request/response parameters on the servers was carried out using Postman. On completion of testing, the servers were then then deployed to Heroku. The Web App was deployed at this point to Azure.

Testing then began on the Web/Mobile Apps communication to the servers. Even though empirical testing was performed throughout each sprint, testing was required once the technologies were deployed.

Testing on the Web App was done with multiple Web Browsers on laptops and mobile devices. Testing of the Mobile App was carried out using emulators for different devices and on physical devices.

The following chapter will look at the technologies that were implemented in the project throughout the process described in this chapter.

Chapter 4

Technology Review

This chapter will discuss the technologies that were utilised in the project in detail. It will provide the reader with examples of the research that was carried out into the different types of available technologies. It will also highlight the rationale behind certain technologies being chosen over others.

4.1 Web Application

In this section, the different technologies used in the Web App will be discussed and the reasons why they were chosen over other technologies.

4.1.1 Azure

Microsoft Azure[8] is a cloud computing platform and infrastructure created by Microsoft. It allows for building, deploying, and managing applications and services through a global network of Microsoft-managed data centres.

Azure was chosen as our hosting platform for the Web App mainly because development could be done using Visual Studio 2015 along with version control.
Another reason for deploying to Azure was because there is a free tier for hosting web applications with the option to add extra features such as Custom Domains, extra memory and SSL web pages. Azure also offers an extensive number of tutorials and documentation.

4.1.2 ASP.NET

ASP.NET[9] is an open source web framework for building dynamic modern web applications and services with .NET. It allows for the creation of scalable websites based on HTML5, CSS and JavaScript. ASP.Net offers resources such as validation tools, Security Control and Web Control templates.

4.1.3 LINQ

LINQ[10] (Language Integrated Query) is a Microsoft programming model and methodology that adds formal query capabilities into Microsoft's .NET-based programming languages. It offers a compact, expressive and intelligible syntax for manipulating data.

Below is an example LINQ Query written in C# used in the Web App. It first sorts a list in ascending order, from the highest value in the Array datesVisited, then re-sorts that list in descending order by the times visited amount. Results can be seen in Figure 4.1.

```
List<TempCustomer> SortedList = custObjList.OrderBy(lv =>
↪  lv.datesVisited.Max()).ThenByDescending(o =>
↪  Convert.ToInt32(o.visited)).ToList();
```

LINQ offers a fast and efficient way to sort/order lists without the need for nested for-loops and as a result made the code more concise and easier to read.

Figure 4.1: LINQ Result On the Web App

4.1.4 Regular Expressions

A Regular Expression or Regex[11] is an object that describes a pattern of characters. Used to perform pattern-matching and search-and-replace functions on text.

Research on user input validation, using Bootstrap Validator showed that custom regular expressions could be used to check the validity of input on the client side before the form is posted back to the server. This in turn would help lighten the workload of the server.

Regular Expression Example.
Below is an example of a regular expression that checks to see that an email is valid and the corresponding image showing the regular expression at work:

```
"^[^@\\s]+@([^@\\s]+\\.)+[^@\\s]+$"
```

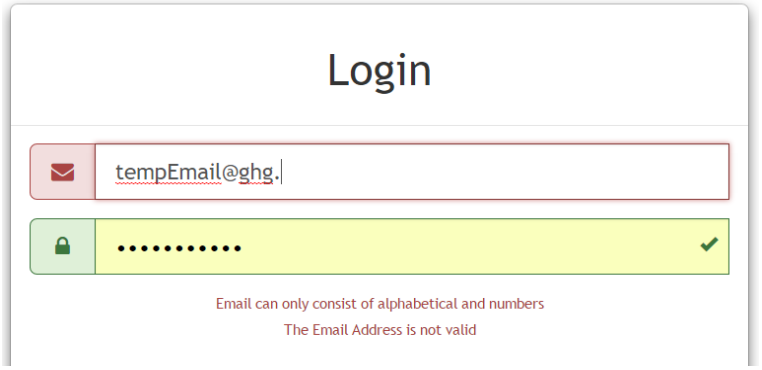

Figure 4.2: Regex Working on Email

Figure 4.2 shows a valid email must contain '@', '.com' and cannot contain spaces in between or characters like !,+,¬

This proved to be an streamlined way to validate user input as extra code did not have to be written to check user input.

4.1.5 EventSource

The EventSource[12] interface was used to receive server-sent events. It connects to a server over HTTP and receives events in text/event-stream format without closing the connection.

The Web App ”subscribes” to a stream of updates generated by the server and whenever a new event occurs, a notification is sent to the Web App.

Some of the reasons on why EventSource was used are:

1. SSEs are sent over traditional HTTP which means they do not require a special protocol or server implementation to get working
2. The Client does not have to continuously poll for new messages because once connected to the server, the messages are sent to the Client when received
3. Setup was simple as demonstrated in the code below

EventSource Connection

```
//Create an EventSource object and pass it the URL of the
↪  stream:
var source = new
↪  EventSource('https://membermemessageserver.herokuapp.com/stream')

source.addEventListener('message', function (e) {
// Message Received.
    //Parse the JSON data
}

source.addEventListener('open', function(e) {
  // Connection was opened.
}, false);

source.addEventListener('error', function(e) {
  if (e.readyState == EventSource.CLOSED) {
    // Connection was closed.
  }
}, false);
```

4.1.6 Cloudinary

Cloudinary[13] is an image management back-end that uses Amazons S3 service. This is where all member images are stored.

The biggest benefit to using Cloudinary was that once the images were uploaded, they could be accessed through HTTPS and would not need to be downloaded or converted back into an image.

The reason Cloudinary was used as the image database was because Neo4j did not provide a way to store images efficiently. The alternative was to convert the image into a byte array and store that in the database. The drawback to that was the large amounts of memory and time it would take to store and retrieve the images.

The images were uploaded via HTTPS, server side, using the package CouldinaryDotNet.

Some of the reasons for using Cloudinary to store images:

1. Image backups
2. Good documentation
3. Integration with multiple languages
4. Free tier that covered the requirements for the project
5. Secure image hosting

Request Example:
The following code example shows how to upload to Cloudinary via CloudinaryDotNet wrapper on ASP.NET

```
//Create New Cloudinary ImageUploadParams object
ImageUploadParams uploadParams = new ImageUploadParams()
{
    File = new FileDescription("FileName")),
    Invalidate = true
};

//Upload The Image
ImageUploadResult uploadResult =
↪  cloudinary.Upload(uploadParams);
```

Response Example:

Json Object sent back from Cloudinary upon successful upload.


```
{
 "public_id":"tquyfignx5bxcbsupr6a",
 "version":1375302801,
 "signature":"52ecf23eeb987b3b5a72fa4ade51b1c7a1426a97",
 "width":1920,
 "height":1200,
 "format":"jpg",
 "resource_type":"image",
 "created_at":"2017-07-31T20:33:21Z",
 "bytes":737633,
 "type":"upload",
 "url":

   ↪  "http://res.cloudinary.com/demo/image/upload/v1375302801/tquyfignx5bxcbsu
 "secure_url":

   ↪  "https://res.cloudinary.com/demo/image/upload/v1375302801/tquyfignx5bxcbs
}
```


4.1.7 RestSharp

RestSharp[14] is an open source library that provides a developer-friendly way of consuming a RESTful API.

The reason RestSharp was chosen was because the library is easy to use and streamlines the process of serialization and deserialization of JSON into objects in C#. It is also widely known and utilised in industry.

RestSharp POST Example:

```
var client = new RestClient("port");
var request = new RestRequest("resource/{id}",
 ↪  Method.POST);
request.AddHeader("Authorization", "Token");
request.AddParameter("name", "name");

//Execute the Request
IRestResponse response = client.Execute(request);
```

RestSharp Deserialization:

```
//Deserialize the result into the class provided
dynamic jsonObject =
 ↪  JsonConvert.DeserializeObject<ResponseMessage>(response.Content);
var resObj = jsonObject as ResponseMessage;
```

4.1.8 Bootstrap

Bootstrap[15] is a front-end framework designed to aid in the development of web apps and sites. Among other things, it includes base CSS and HTML for typography, icons, forms, buttons, tables, layout grids and navigation. It also provides custom-built jQuery-plugins and support for responsive layouts.

For that reason, Bootstrap was the best choice when it came to developing the front end of the Web App. It saves development time as the CSS, HTML and JavaScript is already provided.

Bootstrap provides comprehensive documentation and an abundance of examples are available on the Internet to do everything required.

It allowed for the creation of a simple, modern, responsive Web App using Bootstrap as a template to build on. See Figure 4.3

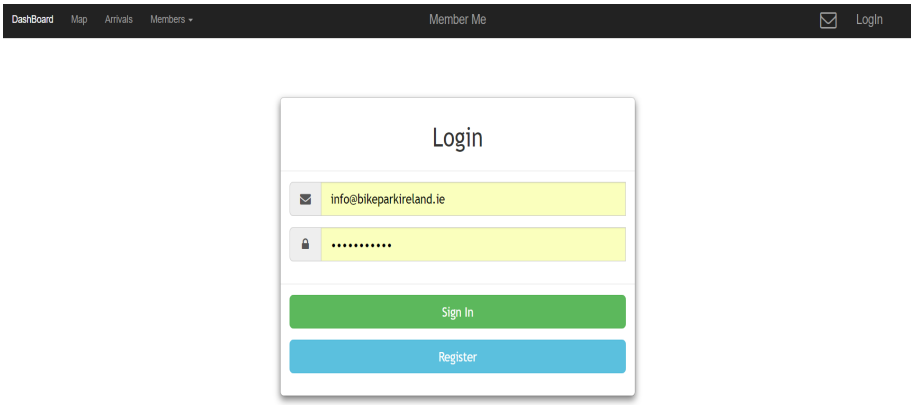

Figure 4.3: Bootstrap Login Page

See Figure 4.4

```
<!-- Modal -->
<div class="modal" id="exampleModal" tabindex="-1"
 ↪  role="dialog" aria-labelledby="exampleModalLabel"
 ↪  aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title"
         ↪  id="exampleModalLabel">Modal title</h5>
        <button type="button" class="close"
         ↪  data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary"
         ↪  data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save
         ↪  changes</button>
      </div>
    </div>
  </div>
</div>
```

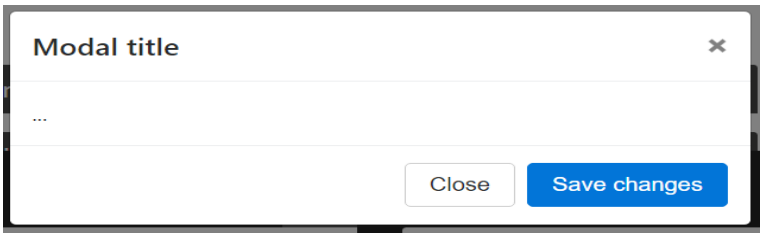

Figure 4.4: Bootstrap Example Modal

4.1.9 D3.js

D3.[16] is a JavaScript library that uses SVG, HTML5, and CSS standards for manipulating documents based on data into dynamic and interactive data visualisations for Web applications.

D3.js was elected for the project as an added technology to showcase some of the analytics gathered from the database visually in a bar chart rather than text format. D3 was the best alternative to convert the data as it is essentially JavaScript with added features. This meant that extra syntax or learning a whole new programming language was not required.

The bar chart used in the Web App was created using D3.js and can be seen in Figure 4.5.

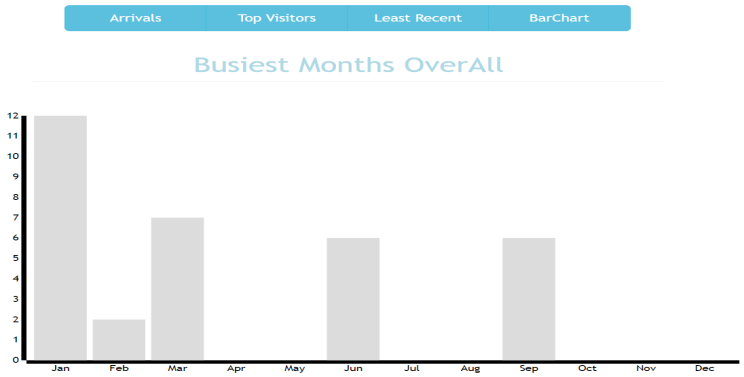

Figure 4.5: D3.JS Bar Chart

The main code for parsing the data into the bar chart:

```
 // Pass in the Data Array and create the Chart
x.domain(data.map(function (d) { return d.Month; }));
y.domain([0, d3.max(data, function (d) { return d.Visits;
↪  })]);

svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
  .append("text")
    .attr("transform", "rotate(-90)")
    .attr("y", 10)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text("Visits");

svg.selectAll(".bar")
    .data(data)
  .enter().append("rect")
    .attr("class", "bar")
    .attr("x", function (d) { return x(d.Month); })
    .attr("width", x.rangeBand())
    .attr("y", function (d) { return y(d.Visits); })
    .attr("height", function (d) { return height -
    ↪  y(d.Visits); })
```

4.1.10 Google Maps API

Google Maps API[17] is a web mapping service developed by Google. It allows third party websites to embed interactive Google maps on websites and devices.

Features included in Google Maps API:

- Directions API: directions between multiple locations
- Geo-coding API: convert between addresses and geographic coordinates
- Places API Web Service: up-to-date information about millions of locations
- Roads API: Snap-to-road functionality to accurately trace GPS breadcrumbs

Google Maps was utilised on both the Web App (Figure 4.6) and Mobile App (Figure 4.7).

1. **Web App**
   For the Web App, the Google Maps API was employed to embed a Map on one of the Web Pages along with custom, click-able markers that display a message on the map based on the geolocation included in an SOS message. As seen in Figure 4.6

Figure 4.6: Embedded Google Maps API On Web App

2. **Mobile App**
   For the Mobile App, Google Maps API was used to get the person's geolocation while on the SOS Page. This was then displayed as a marker on an embedded Google Map at the top of the screen as seen in Figure 4.7.

   This geolocation was an important part of the Mobile app as it was sent along with the SOS message to the associated business, which in turn, showed the location of the person in distress.

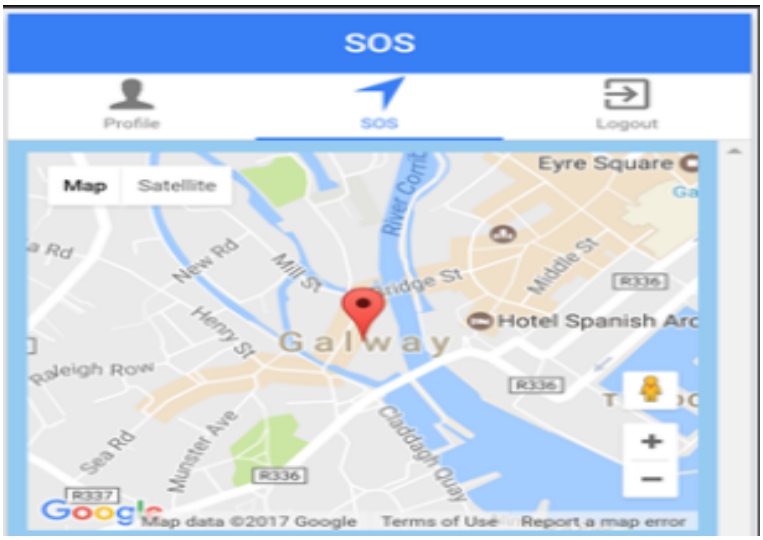

Figure 4.7: Embedded Google Maps API On Mobile App

4.1.11 Web Application Technology Alternatives

For the Web App, there was an abundance of technologies that could have been adopted during development. Research into the alternatives and examination of the scope of the project led to the utilisation of the technologies discussed above.

The following are some of the alternative technologies that were considered:

1. **Web Mapping**

   For the SOS aspect of the Web App, Google Maps and Bing Maps are the two main web mapping providers. Though research, it was found that each provider had similar features with more integration with .NET applications through Bing.

   Advantages of Bing:

   - Good Add-ons including travel and local result
   - Easy integration with ASP.NET
   - Lots of tutorials
   - Customisable

   Advantages of Google:

   - Enhanced and unrivalled features over competitors
   - Fast search engine
   - Continually evolves with improvements
   - Easy Integration on multiple platforms

   Google Maps was favoured as it provided all the requirements needed for the application with easy integration across multiple platforms.

2. **Web Application Platform:**

   One of the reasons for using ASP.NET over others such as Node.js/Angular or J2EE was because of experience using ASP.NET which would provide a good starting platform to build on. There was still a lot of learning involved which built upon and expanded this previous knowledge.

Another reason for using ASP.NET was that there was a number of new technologies being utilised in the project and using a familiar technology would allow for better features and functionality.

Visual Studio 2015, which was the platform on which the Web App was developed on was also a big deciding factor on what language to create the Web App in. The platform provided an easy way to debug, manage, test, deploy and develop code throughout the development.

3. **Hosting Provider:**

   An alternative to Azure is Heroku but after some research it was found that Heroku did not have any official build packs for compiling and deploying straight to Heroku. There were unofficial build packs but the documentation was not very good and deployment wasn't as straight forward as Azure.

   Advantages of Heroku:

   - Free web hosting with available add-ons
   - Lots of tutorials

   Advantages of Azure:

   - Enhanced features over competitors for windows development
   - Bug Tracking
   - Analytics
   - Easy integration with ASP.NET and Visual Studio

   Azure was the best choice because of the integration with Visual Studio, easy deployment and testing of the Web App and the extra features available using add-ons. There was not any advantage to deploying to Heroku other than already having the servers hosted there.

4.2 Mobile Application

The range of technologies used for the Mobile Application is quite extensive because Ionic uses a large number of libraries and dependencies. Each of the technologies used for the Mobile App will be reviewed in this section.

There are a number of different approaches that could be used to create a Mobile App for this project. Therefore, research was carried out to discern which approach would be most appropriate and best suited to the specific needs of the project scope.

4.2.1 npm

npm[18] is the default package manager for the JavaScript runtime environment Node.js. It was used from the very beginning of the Mobile App development and so it is appropriate to discuss it first.

npm comes as part of Node.js installation. It contains a command line client which connects to a remote registry that contains modules. These modules are essentially folders that consist of a number of JavaScript files. Once npm is installed, modules can be installed by opening the npm command line and entering in the following command:

```
npm install -packageName-
```

npm was used quite extensively in the Mobile App development to install modules used by the application. npm creates a file inside the application folders called package.json which contains all the installed modules. This file can then be used to install any module dependencies in the package.json and any available updates to the modules in the registry by using this command:

```
npm install
```

All modules installed from npm into the project are saved under the folder **node_modules** which is found in the applications root directory. When installing a module using npm it is possible to save the module to a global node_modules folder that can be accessed by any project on the system. It is also possible when installing to have the package.json automatically updated to include this module. These options are achieved by running the following command:

```
npm install -g -packageName- --save
```

The package.json file for this project is as follows:

```
{
  "name": "memberme",
  "version": "1.1.1",
  "private": true,
  "description": "memberme: An Ionic project",
  "dependencies": {
    "angular-mocks": "^1.5.8",
    "angular-utf8-base64": "0.0.5",
    "gulp": "^3.5.6",
    "gulp-concat": "^2.2.0",
    "gulp-minify-css": "^0.3.0",
    "gulp-rename": "^1.2.0",
    "gulp-sass": "^2.0.4"
  },
  "devDependencies": {
    "bower": "^1.3.3",
    "gulp-util": "^2.2.14",
    "shelljs": "^0.3.0"
  },
  "cordovaPlugins": [
    "cordova-plugin-device",
    "cordova-plugin-console",
    "cordova-plugin-whitelist",
    "cordova-plugin-splashscreen",
    "cordova-plugin-statusbar",
    "ionic-plugin-keyboard"
  ],
  "cordovaPlatforms": []
}
```

4.2.2 Ionic Framework

Ionic[19] is a framework for developing HTML5 hybrid mobile applications. Hybrid applications are web applications that run in a native browser on Android, iOS and Windows Phone platforms using just one code base.

Ionic is written using HTML, JavaScript, CSS and AngularJS and uses Cordova as a wrapper all of which will be described under individual headings.

Ionic must be installed to the computer system using npm as previously discussed. The command to install Ionic globally is:

```
npm install -g ionic
```

Ionic is run and built using the command line, examples of which is shown in the following Bash code snippets.

```
$ ionic start todo blank //create new application
$ ionic platform add ios //add iOS platform to project
$ ionic platform add android //add android platform to project
$ ionic serve //run application on web browser
$ ionic build ios //build executable for iOS
$ ionic emulate ios //run ios executable on emulator
$ ionic build android //build Android executable
$ ionic run android //run Android emulator
```

4.2.3 Cordova

Cordova[1] is an open-source mobile development framework from Apache. It applies a wrapper around an application (in this case Ionic) to allow the application to be used for different platforms using the same code base.

To access Cordova, it must be installed on the computer system using npm. The command to install Cordova globally is:

```
npm install -g cordova
```

Cordova provides the application with access to native device capabilities such as the camera and Bluetooth using plugins. It also supports a wide range of third party plugins which can be added to the application using npm. Figure 4.8 shows the architecture and connectivity of a Cordova application.

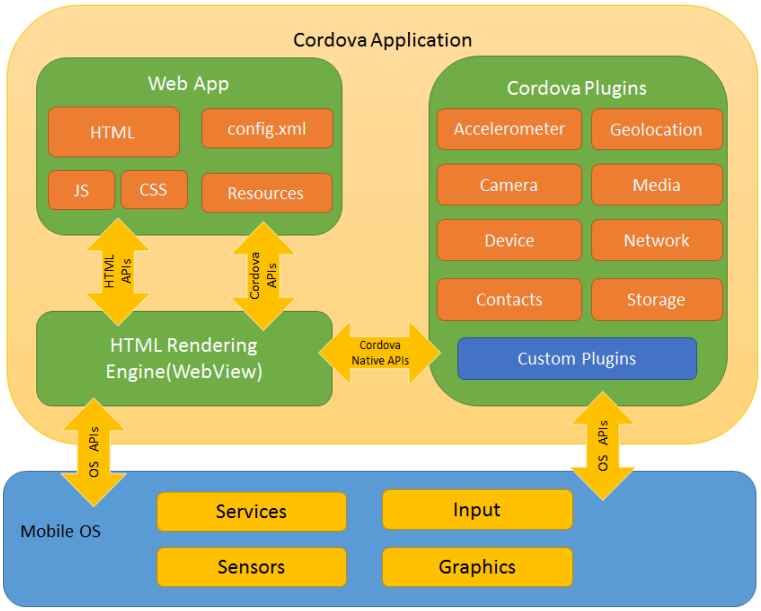

Figure 4.8: Cordova Application Architecture [1]

4.2.4 HTML/CSS

HTML (HyperText Markup Language)[20] is the foundation for webpages which describes the content of the webpage. Ionic uses HTML pages for the different pages or views in an application. Elements that exist on the page, such as input fields, images and script files are created using HTML tags such as the one below:

```
<script src="lib/ionic/js/ionic.bundle.js"></script>
```

The Mobile Application uses pure HTML for the index.html page which is the starting point of the application and all other pages are loaded from it. All other pages in the application use Ionic tags and Angular which will be discussed later.

CSS (Cascading Style Sheets)[21] is a language used to describe how the contents of a webpage look. The CSS code for a particular project is stored in a CSS file within the project directory. When an Ionic application is created it generates a CSS file.

Sass[22] (Syntactically Awesome Style Sheets) is a CSS extension language which supports customisation of CSS files and allows for the use of variables. Sass variables begin with a dollar sign and are initialised to a value. The variable can then be referred to in the properties. An example of Sass is seen below:

```
$energized:                    #ffc900 !default;
```

This can be referred to elsewhere:

```
.profile-list{
  color: $energized;
}
```

4.2.5 JavaScript

JavaScript[23] is a lightweight interpreted programming language that is used extensively with HTML and CSS to form web pages. JavaScript is based on the scripting language ECMAScript which is standardized by the ECMA International[24] standards organisation. ECMA-262 is the official standard name.

4.2.6 AngularJS

AngularJS[25] is a JavaScript framework that can be added to a HTML page. It extends HTML by using **ng-directives** which are HTML attributes with an ng-prefix. AngularJS is a core part of an Ionic application.

AngularJS is based on the software design pattern Model View Controller (MVC). This design pattern supports separation of concerns. The controller works on requests and data preparation with the model. The view is responsible for presenting the data given to it from the controller. Figure 4.9 shows a representation of this process:

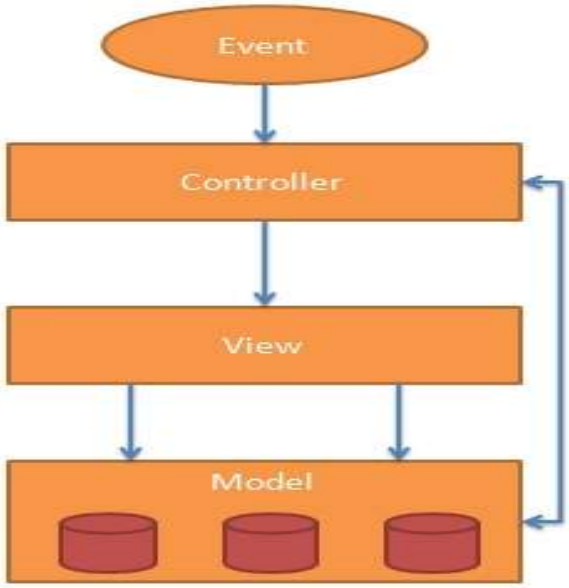

Figure 4.9: MVC Architecture [2]

An AngularJS module defines the application, names it and contains an array of AngularJS modules that should be created and injected into the application. Below is the module for this project which is called **starter** and the list of AngularJS modules injected into the application:

```
angular.module('starter', ['ionic', 'starter.controllers',
↪  'starter.services', 'starter.routes', 'ja.qr', 'ngCordova',
↪  'angular-jwt', 'ab-base64'])
```

The modules injected above begin by defining the module and more modules can be injected into the individual module. The controller module for this application is called **starter.controllers** and contains the following line at the start of the file:

```
angular.module('starter.controllers', [])
```

An Ionic application is defined using **ng-app** in the index.html file with the name given to the angular module above. As discussed in a previous section,

the index.html file is the entry point of the application when it is run. The ng-app directive in this project is shown below:

```
<body ng-app="starter">
```

Another Angular directive used frequently in the Mobile App is **ng-model** which binds the value on the HTML page to data in the controller. An example of this can be seen in the code snippet below from the SOS page of the Mobile App. Here, the user selects from a dropdown menu and the selection is stored in a value called sos.status which has been declared in the corresponding controller.

```
<select ng-model="sos.status" ng-change="emerSelect()">
  <option>Medical Emergency</option>
  <option>Other</option>
</select>
```

4.2.7 Bower

Bower[26] is a package manager similar to npm but there are some differences and it is not unusual to see both package managers used in one application. This is evident from the fact that Bower is installed in the project using npm. The main differences between Bower and npm are:

- npm is used for installing Node.js modules whereas Bower is used for managing front end components like HTML, CSS and JavaScript[27]
- npm maintains the modules it installs in a file called package.json. Bower maintains the packages it installs in a file called bower.json.

The bower.json file for this project is as follows:

```
{
  "name": "memberme",
  "private": "true",
  "devDependencies": {
    "ionic": "driftyco/ionic-bower#1.3.2"
  },
  "dependencies": {
    "angular-qr": "latest",
    "angular-jwt": "^0.1.9",
    "ngCordova": "^0.1.27-alpha"
  }
}
```

4.2.8 Gulp

Gulp[28] is used to automate tasks when developing an application. Ionic applications come pre-packaged with a gulpfile.js file which can be used to create and customise gulp tasks. In this project, gulp tasks were created to monitor Sass changes in the code during development and to reload the application automatically after these changes. This is called *watching*. It saves the developer from having to manually refresh the application after making CSS changes. The gulpfile.js sets the path to the Sass file of the project and uses this in each of the tasks. The tasks for this project are shown below:

```
var paths = {
  sass: ['./scss/**/*.scss']
};

gulp.task('default', ['sass']);

gulp.task('serve:before', ['sass', 'watch']);

gulp.task('sass', function(done) {
  gulp.src('./scss/ionic.app.scss')
    .pipe(sass())
    .on('error', sass.logError)
    .pipe(gulp.dest('./www/css/'))
    .pipe(minifyCss({
      keepSpecialComments: 0
    }))
    .pipe(rename({ extname: '.min.css' }))
    .pipe(gulp.dest('./www/css/'))
    .on('end', done);
});

gulp.task('watch', function() {
  gulp.watch(paths.sass, ['sass']);
});

gulp.task('install', ['git-check'], function() {
  return bower.commands.install()
    .on('log', function(data) {
      gutil.log('bower', gutil.colors.cyan(data.id),
      ↪  data.message);
```

```
    });
});

gulp.task('git-check', function(done) {
  if (!sh.which('git')) {
    console.log(
      '  ' + gutil.colors.red('Git is not installed.'),
      '\n  Git, the version control system, is required to
      ↪  download Ionic.',
      '\n  Download git here:',
      ↪  gutil.colors.cyan('http://git-scm.com/downloads') +
      ↪  '.',
      '\n  Once git is installed, run \'' +
      ↪  gutil.colors.cyan('gulp install') + '\' again.'
    );
    process.exit(1);
  }
  done();
});
```

4.2.9 Angular-jwt

Angular-jwt[29] is an Angular library from Auth0 which allows access to and functionality over JSON Web Tokens (JWT) from the application. The key features of the angular-jwt library are:

- The ability to decode a JWT from the application
- Automatically adding the JWT to the header of every request sent from the application
- Checking the expiration status of the JWT

angular-jwt was installed into the project using Bower and it is injected into the angular.module. The script files are also included in the index.html page as follows:

```
<!-- jwt dependency -->
<script src="lib/angular-jwt/dist/angular-jwt.js"></script>
```

When a user logs into the application, a request for a JWT is sent to the Auth0 server. The response JWT is saved to the local storage for calls to

the Main Server.

The library function jwtHelper is used in the services.js file to check whether the JWT that the application has stored locally has expired or is valid. If the JWT has expired, a new JWT is requested. The code for this function is seen below:

```
function checkAuthOnRefresh() {
  // get stored token from local storage
   var token = window.localStorage.getItem(LOCAL_TOKEN_KEY);
    //if token is not empty
    if (token) {
      //use jwtHelper method to determine if token has
       ↪  expired
      if (jwtHelper.isTokenExpired(token)) {
        //if token has expired, call login method to get new
         ↪  token
        login();
      }
    }
  }
```

4.2.10 Angular-qr : QR Codes

Angular-qr[30] is an Angular library that allows the creation of a QR code inside the application. The main feature of this library is to take a string and encode it into a valid QR code that can be scanned and read by a QR reader.

The package was installed into the project using Bower and it is injected into the angular.module. The script files are also included in the index.html page. The QR code appears in the profile.html page with the following code:

```
<qr text="qrcode" class="align-qr" size=size></qr>
```

The **qrcode** variable is set in the Controller for this page and takes the email address of the user once they have successfully logged in to generate the QR code as seen here:

```
$scope.qrcode = $scope.profile.email;
```

The generated QR code is for the user to check in at a business that they are a member of. The full functionality for this is discussed under the Web Application and Mobile Application sections of the System Design chapter of this report.

4.2.11 Mobile Application Technology Alternatives

There was a number of different development options that could have been used to create the Mobile Application. Ultimately, one had to be chosen over others and there was logic behind these decisions. The following are some of the alternatives that could have been used and why they were eventually decided against.

Ionic 2

The team behind Ionic released a new version of the app development framework, Ionic 2 in Alpha and then Beta form in 2016. This was to coincide with the release of Angular 2 which as discussed earlier, Ionic is built upon. In the research stage of this project, Ionic 2 was early in the Beta testing phase. This meant that there was the possibility of bugs/issues arising with the software. There was also a lot less documentation and examples available for this newer version. It made sense therefore to decide to favour the original, stable version of Ionic.

Native Application Development

There was also the option of developing the Mobile App for a specific platform such as Android[31] or iOS[32]. Developing an Android specific application involves programming in the Java programming language and using software such as Android Studio. For an iOS specific application, programming would be done in the Swift programming language using Xcode. Also, an iOS specific application requires the development to be done on a Mac computer.

The final decision to develop a hybrid application instead of a native application came about for a number of reasons:

- Developing a hybrid application would result in being able to use one source code to target multiple platforms
- The project team did not have access to a Mac computer to develop specifically for iOS
- There was previous knowledge and experience using Ionic which meant that the final application had more functionality and was more polished than starting to learn an entirely new application development process

QR Code Alternative

The original idea for the project was to use a membership card that a user could swipe when checking in at a business. Discussions and research were carried out about how to design and produce such cards. This would have required extra hardware e.g. a printer to produce the cards and a laminator to protect the cards.

The idea to generate a barcode or QR code that could be used in place of a physical card was then discussed. This proved to be a feasible idea with code generators already in existence. This also eradicated the need for extra physical hardware making it cheaper for a business to implement and use. Instead, a barcode/QR code scanner could be used, of which, there are numerous free applications already in the public domain. It also meant that a user would not have to take care of a physical card as all they would need is their mobile device.

4.3 Server Side Technologies

4.3.1 Heroku

Heroku[33] is a container-based cloud Platform as a Service (PaaS). For this project, Heroku was used to deploy and run both the Main Server and the Messaging Server using a free Heroku Hobby account.

Heroku allows the deployment of applications from the version control system, GitHub by associating a specified GitHub repository when creating new applications. This means that when changes in the source code of the servers are sent to GitHub, they are automatically updated, redeployed and run in Heroku.

Heroku builds the application the first time the repository is added and every subsequent time that a new version of the source code is added to GitHub.

Heroku uses the package.json file of the project to download any necessary dependencies. It also compiles the source code of the project and generates a **slug** which is the dependencies, compiled source code and language runtime all bundled together ready for execution.

A Procfile is a text file that is required to be present in the root directory of the application. It tells Heroku what language the application is written in

and what file should be called to start the application running.

The Procfile for the Main Server is as follows:

```
web: node server.js
```

Heroku also offers add-ons which are components or services maintained by a third-party provider or by Heroku itself. Add-ons are installed into the application by Heroku and are used to handle parts of the application such as data storage. This project uses two add-ons to the Main Server application, Cloudinary and GrapheneDB both of which are discussed in more detail in other sections of this chapter.

4.3.2 Node.js & Express.js

Node.js[34] is a open-source JavaScript runtime environment used to develop scalable network applications. Node.js has an event driven architecture which employs asynchronous events rather than a concurrent, thread-based model of networking which means it does not face issues such as deadlocking.

Express.js[35] is an application framework that is based on Node.js. It is seen as the de facto framework for Node.js[36] and provides a host of features and functionality to an application. It contains a large range of HTTP methods and middleware and makes it ideal for creating APIs.

Node.js uses npm (previously discussed under Mobile Technologies) to install packages which are declared in a package.json file. Express.js is added in this way and once declared in the main server page (server.js) it controls most elements of the server including routing, middleware for all requests and setting the port for the application to listen on.

Below are some examples of this from the Main Server server.js file:

```
var express = require('express');
var app = express();
```

In this code snippet, Express.js is declared as a required package for the file. The express function is called and assigned to the variable **app**.

```
app.all('*', function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header('Access-Control-Allow-Methods', 'PUT, GET, POST,
   ↪ DELETE, OPTIONS');
  res.header("Access-Control-Allow-Headers", "Content-Type,
   ↪ Accept");
  next();
});
```

Here, the express variable **app** is used to create middleware that all requests to the server must pass through and sets what methods, headers and request origins are allowed to access routes.

```
var router = express.Router();
app.use('/api', router);
.........
router.post('/authenticate', function (req, res)
```

Here, an instance of Express.router is created and set to the variable **router**. **router** is set to be used for all routes and prefix each route with **api**. The final line shows **router** being used to set up a route called **authenticate**.

4.3.3 Auth0

Auth0[37] is a service that allows user authentication for applications using JSON Web Tokens (JWT). This project uses Auth0 authentication for users of both the Mobile and Web Apps by designation of a JWT, the validity of which is checked on the Main Server.

A client application account has been set up with Auth0 to manage the authorisation for this project. This client account contains the configuration settings for the request authorisation method. These settings include the expiration time for the JWT and the token endpoint authentication method. The JWT is obtained by using a HTTP POST request to the client application.

While the project was in the development phase, Allowed Callback URLs were set to allow localhost requests for testing as seen below:

Figure 4.10: Auth0 Allowed Callback URLs

In order to allow requests from the Mobile and Web Apps, it was necessary to set the allowed origins (CORS) of where requests are made from. CORS or Cross-Origin Resource Sharing is when a resource makes a HTTP request from a different domain from the resource making the request. For security purposes, most browsers do not allow cross-origin requests and so it was necessary to set this to be allowed in this project. This was done by allowing this setting in the Auth0 client configurations seen in figure 4.11

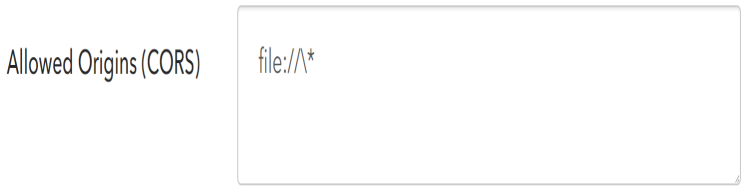

Figure 4.11: Auth0 Allowed Origins

Both the Mobile and the Web Apps send POST requests to the Auth0 client for JWT. Examples of both calls can be seen below starting with the Mobile Application:

```
//POST method for getting Auth0 token - used in login
↪  function
  var options = { method: 'POST',
  url: 'https://membermeauth.eu.auth0.com/oauth/token',
  headers: { 'Content-Type': 'application/json' },
  data:
   ↪  '{"client_id":"fXqMFIGFPGXAPLNm6ltd0NsGV6fWpvDM","client_secret":"HHnBRmKT
   ↪  };

  ......

  //use POST call options
  var login = function(){
    return $q(function(resolve, reject) {
      $http(options).then(function(result) {
        if (result.statusText=="OK") {
          //if successful send token to storeUserCredentials
          storeUserCredentials(result.data.access_token);
          resolve(result.data.message);
        } else {
          reject(result.data.success);
        }
      });
    })
  }
```

The call from the Web App is as follows:

```
var client = new RestClient(auth0EndPort);
var request = new RestRequest(Method.POST);
request.AddHeader("content-type", "application/json");
request.AddParameter("application/json",
↪  "{\"client_id\":\""+clientId+"\",\"client_secret\":\""+clientSecret+"\",\"au
↪  apiEndPort + "/\",\"grant_type\":\"client_credentials\"}",
↪  ParameterType.RequestBody);

IRestResponse response = client.Execute(request);

//Deserialize the result into the class provided
dynamic jsonObject =
↪  JsonConvert.DeserializeObject<Token>(response.Content);
var bizObj = jsonObject as Token;// ACCESS TOKEN USED TO GET
↪  AUTHENTICATED
```

The Main Server uses an Express.js *(see Express.js section)* function from Auth0 to check the validity of a JWT that is sent with application requests to the server. This server function is shown below:

```
var jwt = require('express-jwt');
.......
var jwtCheck = jwt({
  secret: rsaValidation(),
  audience: 'https://restapicust.herokuapp.com/api/',
  issuer: "https://membermeauth.eu.auth0.com/",
  algorithms: ['RS256']
});

app.use(jwtCheck);
```

4.3.4 JSON

JSON[38] (JavaScript Object Notation) is a lightweight data-interchange format. It uses human-readable text to send data objects consisting of name/value pairs. JSON is defined by ECMA-404[39] standards which describes the allowed syntax.

This project uses JSON extensively to send data from both the Web and Mobile Apps to the Main Server and responses are returned from the Main Server in JSON format. The Neo4j database also returns responses to the Main Server in JSON format. Examples of JSON responses can be seen under the System Design chapter of this report under the RESTful API Server section.

4.3.5 Neo4j

Neo4j[40] is a native, noSQL graph database that uses its own query language called Cypher[41]. It uses explicit connections to related entities which means that queries are faster because they do not process anything that's not connected and so do not have to query the entire data-set. This also means that the database is far more scalable than relational or object-oriented databases. This was a driving force behind the decision to use Neo4j for this project over other alternative database types.

The utilisation of the Neo4j database for this project can be seen in more detail in the System Design chapter under the Database section. Examples of Cypher queries can be seen in the same chapter under the section RESTful API Server.

The Main Server connected to the Neo4j database using the Neo4j Bolt driver for JavaScript. It is officially supported by Neo4j and connects to the database using the new binary Bolt protocol. The driver was installed using npm and uses Heroku environment variables to connect to the database via the GrapheneDB addon.

4.3.6 Server-Sent Events

A Server-Sent Event (SSE)[42] is when a web page automatically gets updates from a server without the client being required to continuously poll a server to see if any updates are available.

An alternative to SSE is WebSockets. WebSockets provide bi-directional, full-duplex communication over a single TCP connection. This is beneficial for things such as messaging apps, games and anything that requires real-time updates in both directions.

SSE are sent over traditional HTTP, meaning they do not require any special protocol or server implementation to get working. This type of messaging

is beneficial when data does not need to be sent from the client back to the server e.g Twitter News Feed or Facebook News feed.

Some of the benefits of SSE:

- Automatic reconnection
- Event Ids
- Easy setup
- No special protocols needed

4.3.7 Postman

Postman[43] is a powerful GUI platform that makes API development faster and easier. It is a complete tool-chain for API Development.

Postman allows:

- Building of API requests including Post/Get/Update/Delete
- Documentation
- Testing
- Response viewing
- Authorisation
- Collection sharing

Figure 4.12 is an example of testing the authentication route for a business on the main server.

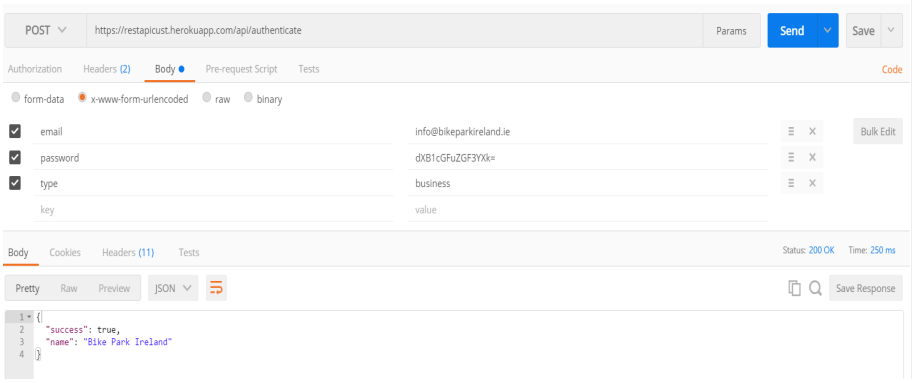

Figure 4.12: Postman Restful API Request

4.3.8 Server Side Technology Alternatives

Hosting Site for Main Server

In the research phase of this project, a location to host the servers needed to be chosen. It was found that there were a number of options for this. One of these options is OpenShift Online.

OpenShift Openshift Online[44] is a cloud based Platform as a Service (PaaS) that allows the deployment and management of applications. Some of the main differences between OpenShift Online and Heroku which influenced why Heroku was ultimately used for this project are as follows:

- Both Heroku and OpenShift offer free accounts but Heroku allows for an unlimited number of applications whereas OpenShift free accounts are restricted to 3 applications
- Heroku allows deployment and updating of the application through GitHub, DropBox and Git where OpenShift only supports Git
- Heroku has a large number of supported addons for applications. OpenShift only offers a small number of addons

Database Options

When choosing technologies for this project, there was no bigger selection to chose from than when it came to deciding on a database system. Not only are there a large range of database types such as relational, object-orientated, document-oriented and graph databases, there are a number of different options within each of these types.

After research and team discussion, the choice was narrowed down to two types of NoSQL databases. These were MongoDB[45] which is a document-oriented database and Neo4j which is a graph database. There was some experience of using both of these databases having used Neo4j in a 3rd year Graph Theory module and MongoDB as part of a project for a 4th year Emerging Technology module. Both were looked at in terms of suitability for this project.

MongoDB stores data as JSON-style documents rather than in tables and rows as in a relational database. Similar documents are stored together within a collection e.g. a user document would be stored in a collection of users. Relationships in MongoDB are stored as nested data within the document. In terms of complex queries or analytics this can make MongoDB searches more convoluted.

There were a number of reasons why Neo4j was chosen for this project over MongoDB some of which are discussed below:

1. One of the most important elements of the data stored in the database in this project is the relationships between users and businesses. For this purpose, Neo4j really excels as it automatically indexes nodes that are frequently used so queries made to the database will actually get faster the more time they are run.

2. Cypher queries are designed to be so easily read and written that even someone with no exposure to Cypher would be able to work out what the query was for. The simplicity of these queries was a strong point in Neo4j favour over MongoDB.

4.4 GitHub

GitHub[46] is a web-based version control repository. It was used for the development of this project with a number of different branches used so that the different elements (i.e. Mobile and Web Apps and servers) were separate and could be worked on at the same time without version conflicts.

Other features of GitHub that were utilised included the issue tracker and the wiki. The wiki was used to highlight resources that could prove to be useful for the project. The issue tracker was used to document and discuss issues and bugs that arose in the development process.

Now that all the technologies have been looked at in detail, the following chapter will look at the entire system architecture of the project and how these technologies have been implemented in the system.

Chapter 5

System Design

This chapter will provide a detailed explanation of the overall system architecture. It will show how the technologies that were discussed in the previous chapter have been coupled together to form the overall system.

Figure 5.1 shows the system architecture of the project. The image shows the main components of the system which are Auth0 for authentication, the Mobile and Web Apps, the Main and Message Servers that are hosted on Heroku and the database and image storage and how they all interact. These componants will be looked at comprehensively in this chapter.

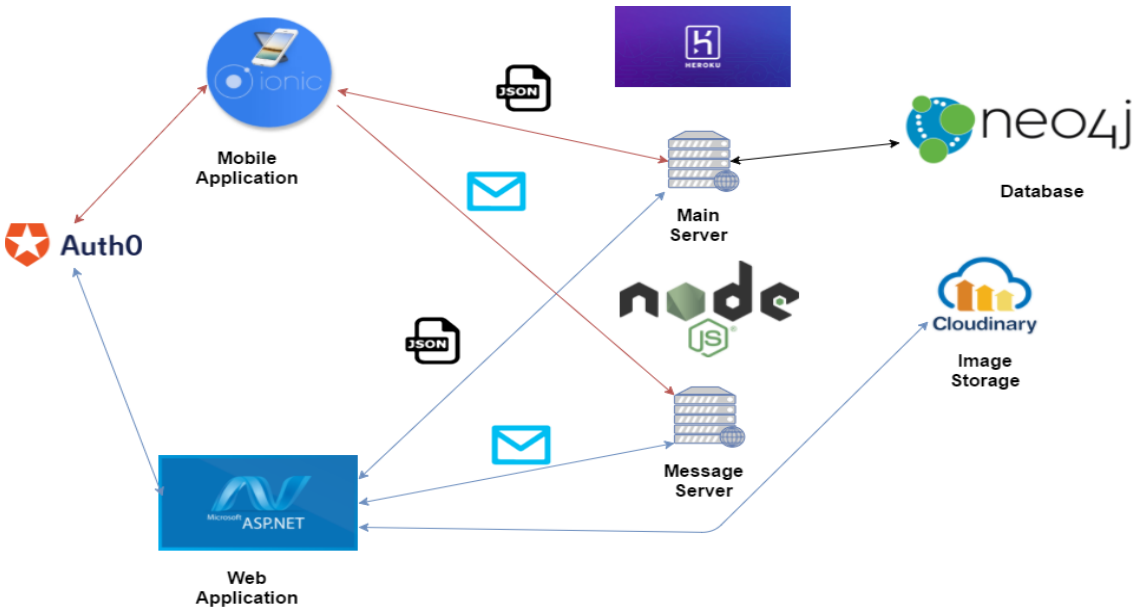

Figure 5.1: System Architecture

5.1 Database

5.1.1 Purpose

This section will describe the database and how it is used to support the membership system through the use of nodes and relationships.

There are 2 different types of nodes that are stored in the database:

- Person
- Business

The following tables 5.1 and 5.2 show the labels that make up the two types of nodes:

Name	Type
name	String
address	String
phone	String
iceName	String
icePhone	String
joined	String
dob	String
visited	String
membership	String
datesVisited	Array
publicImgId	String
imageUrl	String
email	String
guardianName	String
guardianNum	String
bEmail	String
password	String (base64 encoded)

Table 5.1: Person node

Name	Type
name	String
address	String
phone	String
email	String
password	String (base64 encoded)
emergencyNum	String

Table 5.2: Business node

The Person and Business nodes in the database are connected by relationships as seen in figure 5.2. A Person is connected to a business with an IS_A_MEMBER relationship and a Business is connected to a person with a HAS_A_MEMBER relationship. A Business node can have several Person members but a Person node can only only be a member of one Business.

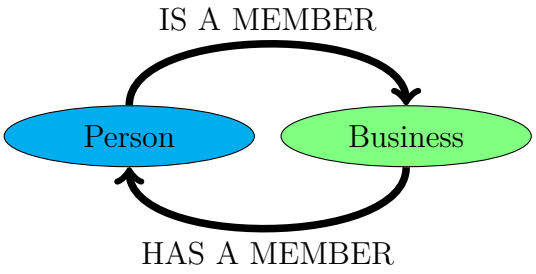

Figure 5.2: Node Relationships

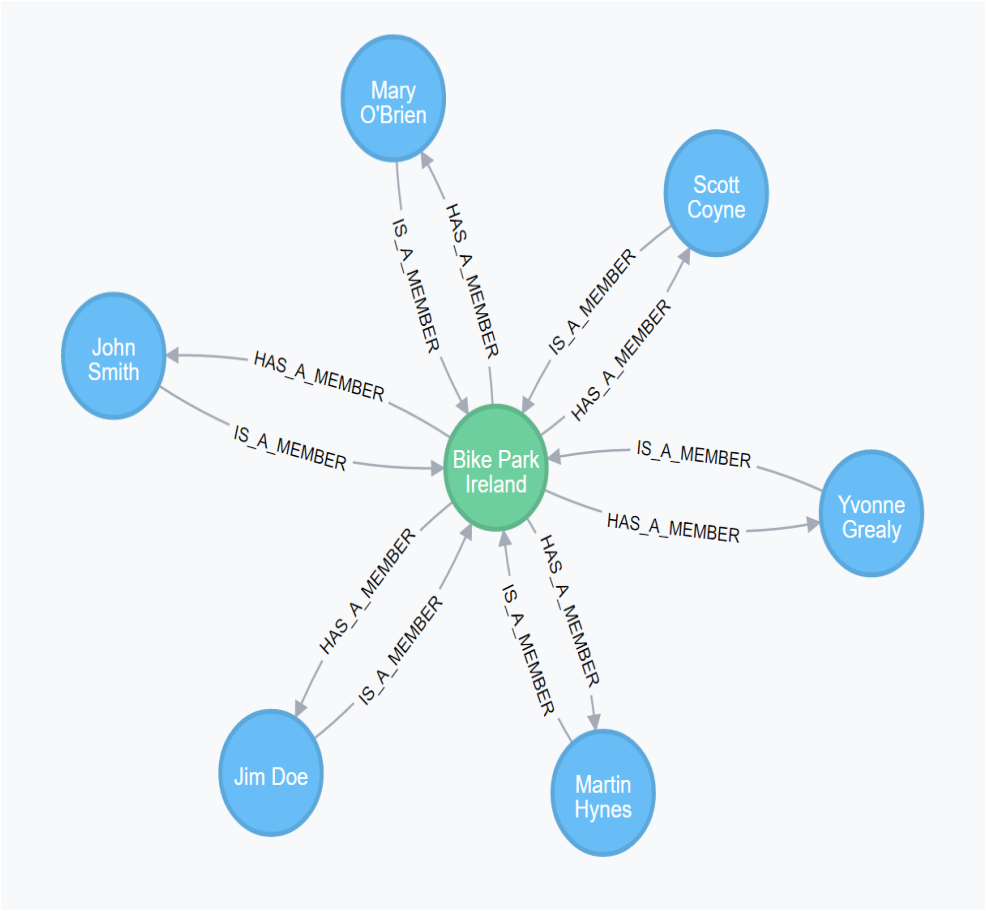

Figure 5.3: Example Business with Person members

The database uses the Cypher query language. The queries for this project will be discussed in the next section under the RESTful API Server queries. A feature of the Neo4j database that was used in this project was the use of unique node property constraints. This allowed the database to be set so that each node in the database had to have a unique email address. This meant that no two users or two businesses could have the same email address so each email was a unique identifier for that node. An example Cypher constraint query is as follows:

```
CREATE CONSTRAINT ON (a:Person) ASSERT a.email IS UNIQUE
```

5.2 RESTful API Server

This section will discuss the RESTful API Server (the server) including its purpose and routes.

5.2.1 Purpose

The purpose of the server is to act as an intermediate between the Mobile App and Web App, and the Database. Requests are sent to the server from both applications and the server processes the information and sends the appropriate queries to the Database. It then returns a response to the requesting application.

The server also acts as a security measure for the database. Each request sent to the server must contain a valid Auth0 security token. The following code shows how the server checks each request for a valid Auth0 token and if a valid token is not found, the server returns an error.

```
var jwtCheck = jwt({
  secret: rsaValidation(),
  audience: 'https://restapicust.herokuapp.com/api/',
  issuer: "https://membermeauth.eu.auth0.com/",
  algorithms: ['RS256']
});

app.use(jwtCheck);

app.use(function (err, req, res, next) {
  if (err.name === 'UnauthorizedError' || err.name ===
  ↪  'TypeError') {
    res.status(401).json({ message: 'Missing or invalid token'
    ↪  });
  }
});
```

This means that even if someone knows the routes on the server, they are not able to access any of the routes without a valid Auth0 token. The security function here is that there is no direct access to the database information. All queries from the Mobile and Web Apps must go through the server which makes the database information more secure.

5.2.2 HTTP RESTful Routes

The server contains a number of routes that can be sent requests. Some of the routes are only accessible by the Web App which has responsibility for the administration functions of the membership system. Other routes are available to the Mobile App which the customer has direct use of.

Common Routes

Authenticate
This route is used to authenticate a Person or Business logging into the system. The route is used by both the Web and Mobile Apps however the request and response are different depending on which application is accessing it. The route and an example request and response are shown below along with the Cypher query for both a business and person log in:

Business Login URL:
`authenticate[/:email][/:password]`
Request:
`authenticate/outdoor@email.com/password`
Response:

```
{"success": true, "name": "record._fields[0].properties.name"}
```

Cypher Query:

```
MATCH (a:Business)
WHERE a.email=email
AND a.password=pwd
RETURN a
LIMIT 1
```

Person Login URL:
`authenticate[/:email][/:password]`
Request:
`authenticate/joe@email.com/password`
Response:

```
{
  "success": true,
  "name": "Jim Doe",
  "dob": "318038400000",
  "address": "14 Main Street, Galway",
```

```
  "phone": "9876544567",
  "iceName": "John Smith",
  "icePhone": "0877779999",
  "joined": "1487621427870",
  "email": "jimbo@hotmail.com",
  "imgUrl":
   ↪  "http://res.cloudinary.com/hlqysoka2/image/upload/s--t8ju2ucu--/v148720656
  "guardianName": "null",
  "guardianNum": "null",
  "visited": "4",
  "membership": "0",
  "datesVisited": [
    "1452902400000",
    "1465772400000",
    "1452729600000",
    "1473807600000",
    "1487894400000"
  ],
  "bEmail": "info@bikeparkireland.ie"
}
```


Cypher Query:

```
MATCH (a:Person)-[r:IS_A_MEMBER]->(b:Business)
WHERE a.email=email
AND a.password=pwd
RETURN a, b
LIMIT 1
```

Web Application Only Routes

Create Business

This route is used to create a new Business node on the database. The route and an example request and response are shown below along with the Cypher query:

URL:
addCompany[/:name][/:address][/:phone][/:email][/:password][/:emergencyNum]
Request:
addCompany/Outdoor/address/0914859385/outdoor@email.com/password/0871234567
Response:

```
{"success": true, "message": "Business created!"}
```

Cypher Query:

```
CREATE (b:Business {name: business.name,
↪ address:business.address, phone:business.phone,
↪ emergencyNum:business.emergencyNum,  email:business.email,
↪ password:business.password})
```

Create Person with relationship to business
This route is used to create a new Person node on the database with a relationship to an existing Business. The route and an example request and response are shown below along with the Cypher query:

URL:
addPerson[/:name][/:address][/:phone][/:iceName][/:icePhone][/:dob][/:email][/:password][/:imgUrl][/:publicImgId][/:guardianName][/:guardianNum]
Request:
addPerson/Joe/address/0914859385/Mary/1234/09081992/joe@email.com/password/http://res.cloudinary.com/hlqysoka2/image/upload/v1485038988/sample.jpg/pdkj6e3qc4bt0ogxqv05/null/null
Response:

```
{"success": true, "message": "Person created!"}
```

Cypher Query:

```
MATCH (b:Business {email: bEmail})
CREATE (a:Person {name: person.name , address: person.address ,
↪ phone: person.phone , iceName: person.iceName , icePhone:
↪ person.icePhone , joined: person.joined , dob: person.dob ,
↪ email: person.email , imgUrl: person.imgUrl , password:
↪ person.password , visited: person.visited , membership:
↪ person.membership , publicImgId: person.publicImgId ,
↪ guardianName: person.guardianName , guardianNum:
↪ person.guardianNum})
CREATE (a)-[:IS_A_MEMBER]->(b)-[:HAS_A_MEMBER]->(a)
RETURN COUNT(*)
```

Find Person
This route is used to find a Person when they check in at the business. The route and an example request and response are as follows:

URL:
findPerson[/:email][/:bEmail]
Request:
findPerson/joe@email.com/outdoor@email.com
Response:


```
{
  "success": true,
  "name": "Jim Doe",
  "dob": "318038400000",
  "address": "14 Main Street, Galway",
  "phone": "9876544567",
  "iceName": "John Smith",
  "icePhone": "0877779999",
  "joined": "1487621427870",
  "email": "jimbo@hotmail.com",
  "imgUrl":
  ↪  "http://res.cloudinary.com/hlqysoka2/image/upload/s--t8ju2ucu--/v148720656
  "guardianName": "null",
  "guardianNum": "null",
  "visited": "4",
  "membership": "0",
  "publicImgId": "pdkj6e3qc4bt0ogxqv05",
  "datesVisited": [
    "1452902400000",
    "1465772400000",
    "1452729600000",
    "1473807600000",
    "1487894400000"
  ]
}
```


Cypher Query:

```
MATCH (a:Person)-[r:IS_A_MEMBER]->(b:Business)
WHERE a.email=email
AND b.email=bEmail
RETURN a, b
LIMIT 1
```

Update Person Details
This route is used to update a specific Person node on the database. The

route and an example request and response are shown below along with the Cypher query:

URL:
updatePerson[/:email][/:name][/:membership][/:guardianName][/:guardianNum][/:visited][/:datesVisited][/:tempEmail]
Request:
updatePerson/joe@email.com/address/null/null/null/1/12042016/joe@newemail.com
Response:

```
{"success": true, "message": "User Details Updated"}
```

Cypher Query:

```
MATCH (a:Person)
WHERE a.email= email
SET a.name=name, a.membership=membership,
 ↪  a.guardianName=guardianName, a.guardianNum=guardianNum,
 ↪  a.visited=visited, a.datesVisited=datesVisited, a.email=
 ↪  tempEmail
RETURN COUNT(*)
```

Delete a Business

This route is used to delete a Business node on the database and will also delete all nodes that have a relationship with it. The route and an example request and response are shown below along with the Cypher query:

URL:
deleteCompany[/:email]
Request:
deleteCompany/outdoor@email.com
Response:

```
{"success": true, "message": "Business Deleted"}
```

Cypher Query:

```
MATCH (b:Business {email:email})
OPTIONAL MATCH (b)-[r]-(p)
DETACH DELETE b, r, p
RETURN COUNT(*)
```

Delete a Person

This route is used to delete a Person node on the database and any relationships it has with other nodes. The route and an example request and response are shown below along with the Cypher query:

URL:
deletePerson[/:email][/:bEmail]
Request:
deletePerson/joe@email.com/outdoor@email.com
Response:

```
{"success": true, "message": "User Deleted"}
```

Cypher Query:

```
MATCH (a:Person)-[r:IS_A_MEMBER]->(b:Business)
WHERE a.email=email
AND b.email=bEmail
OPTIONAL MATCH (a)-[r]-(b)
DETACH DELETE a,r
RETURN a, b
LIMIT 1
```

Top Ten Visitors

This route is used to analyse the top ten visitors to the business. The route and an example request and response are shown below along with the Cypher query:

URL:
topTenVisited[/:bEmail]
Request:
topTenVisited/outdoor@email.com
Response:

```
{"success": true, "message": topTenList}
```

Cypher Query:

```
MATCH (a:Person)-[r:IS_A_MEMBER]->(b:Business)
WHERE b.email=bEmail
WITH a
ORDER BY a.visited
DESC
RETURN a
LIMIT 10
```

Mobile Application Only Routes

Update Person Details

This route is used to update a specific Person node on the database. This route does not allow as many details to be changed as the Web App route updatePerson. The route and an example request and response are shown below along with the Cypher query:

URL:
mobileUpdatePerson[/:email][/:name][/:address][/:phone][/:iceName][/:icePhone]
Request:
mobileUpdatePerson/joe@email.com/Joe/address/0861234567/John/0871234567
Response:

```
{"success": true, "message": "User Details Updated"}
```

Cypher Query:

```
MATCH (a:Person)
WHERE a.email= email
SET a.name=name ,a.address=address ,a.phone=phone
↪  ,a.iceName=iceName ,a.icePhone=icePhone
RETURN COUNT(a)
```

Set New Password

This route is used to set a new password on a Person node on the database. When a user first logs into the Mobile App they do so using a generated temporary password. On first login they must change their password using this route. The route and an example request and response are shown below along with the Cypher query:

URL:
newPassword[/:email][/:password]
Request:
newPassword/joe@email.com/myPassword
Response:

```
{"success": true, "message": "User Password Updated"}
```

Cypher Query:

```
MATCH (a:Person)
WHERE a.email=email
```

```
SET a.password=password
RETURN COUNT(*)
```

5.3 RESTful Messaging API Server

This section will discuss the RESTful Messaging API Server, its purpose and its routes.

For simplicity, in this section, the RESTful Messaging API Server will be referred to as the "Messaging API".

5.3.1 Purpose

The purpose of the Messaging API is to allow members to send SOS messages through HTTP, to the associated business, alerting the business that they are in difficulty.

Below is the message Object that is sent and received:

```
var messageObj = { status: "", message: "", latitude: "",
↪  longitude: "", email: "", business: "", name: "" }
```

5.3.2 Routes

The following are the different routes used in the messaging server which can broken up into two main sections:

1. **Connection/Stream**

   Request URL:
   https://membermemessageserver.herokuapp.com/stream
   Response Status:

   ```
   {"200"}
   ```

   Route:

   ```
   router.get('/stream', function (req, res) {
       res.sseSetup()//Setup of the Stream
       res.sseSend(messageObj)//Send the Object
       ↪  "Message""
       connections.push(res)//Push the New Connection To
       ↪  the global Array
   })
   ```

The code above takes in all the connection details of a business and saves its details to an array containing all connected business. This array **connections** is then used in the SendMessage route.

2. **SendMessage**

   This is the SendMesage Route, which is where the Customer sends the SOS messages using the Mobile App.

   Request URL:
   https://membermemessageserver.herokuapp.com/sendMessage
   Request:

```
{ "status:" "Medical Emergency", "message:" "Help!
↪  Broken my leg", "latitude:" "53.2793146",
↪  "longitude:" "-9.0878947", "email:"
↪  "scottcoyne@email.com", "business:" "bikeparkire",
↪  "name:" "scott" }
```

   Response Status:

```
{"200"}
```

   Route:

```
router.post('/sendMessage', function (req, res) {
  messageObj.message = req.body.message;
  messageObj.status = req.body.status;
  messageObj.business = req.body.business;
  messageObj.email = req.body.email;
  messageObj.latitude = req.body.latitude;
  messageObj.longitude = req.body.longitude;
  messageObj.name = req.body.name;
  //For each Connected Company, Send Message Object
  for (var i = 0; i < connections.length; i++) {
    connections[i].sseSend(messageObj)
  }
  messageObj = { status: "", message: "", latitude:
  ↪  "", longitude: "", email: "", business: "",
  ↪  name: "" }
  res.sendStatus(200)
})
```

   The code creates a new messageObj and sends it to all businesses that have a connection to the server.

5.4 Web Application

This section will detail the functionality and design of the Web App.

The Web App is currently hosted at MemberMe, and can be accessed with the following test log-in Credentials:

1. **UserName:** bikeparkireland@hotmail.com
2. **Password:** upupandaway

5.4.1 Purpose

The purpose of the Web App was to give businesses the ability to create a business account on which they could create and manage members.

Based on the information each customers provides, the business can query the database and retrieve/display analytics based on how many times/when a member visits.

Another feature of the Web App is the ability to receive SOS messages from members via the Message Server and display them in the application as they arrive.

5.4.2 Web App Architecture

Figure 5.4 shows the architecture of the web App and can be broken up into three different parts.

1. **Microsoft Azure:** The Web App is hosted on Microsoft Azure and can be accessed anywhere online.
2. **Heroku:** Main Server/Cloudinary/messaging server is hosted here.
    - The Main Server handles RESTful requests from the Web App to the Database
    - Cloudinary handles the saving and retrieval of customer images
    - The Messaging Server is where the SOS messages sent from the Mobile App are retrieved and passed to the Web App
3. **Auth0:** This is where authorisation access to the Main Server is handled.

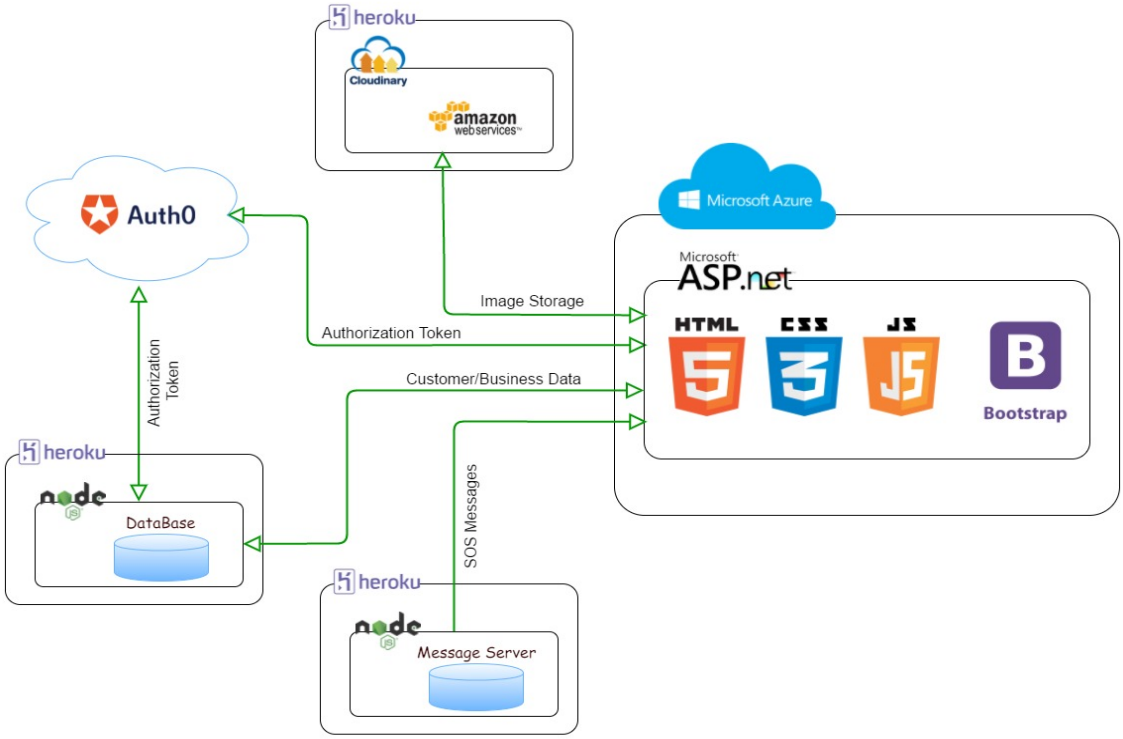

Figure 5.4: Web App Architecture

5.4.3 Web App Design

Below are some of the design choices considered when designing the Web App:

1. User friendly (Simple User Feedback)
2. Self-explanatory (No hidden meaning on pages)
3. Cutting edge (Trendy design)
4. Responsive (Web App Re-sizes on different screen sizes)
5. Secure (User authentication)
6. Clutter free (Simple design and layout)

Navigation

The navigation bar at the top of the Web App is for navigation to the different pages and to view any SOS messages received from members.

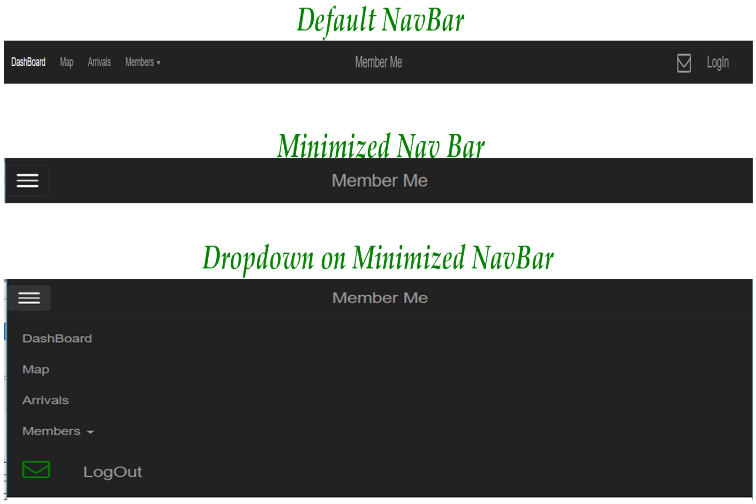

Figure 5.5: Navigation Bar

Figure 5.5 shows the navigation bar has a simple design, is intuitive and has been designed to be device responsive by resizing itself based on the current screen size.

Business Authentication

Access to the Web App requires authentication. Only valid user credentials grant access to the features of the Web App and access to the database.

Figure 5.6 shows sample data in the required fields. Each field is validated using JavaScript, Bootstrap and ASP.NET validation. This prevents the application from crashing and stops invalid data being entered. This provides a simple user friendly log in interface.

The log in works by sending off a Post request to the Auth0 API to get a JWT authorisation towken. Once the token has been received, a Post request is then sent to the Main Server which checks the validity of the token. If the token is valid then the Server sends a request to the database to check if the user is valid. A JSON response is returned with a message letting the Web App know whether authentication has been successful.

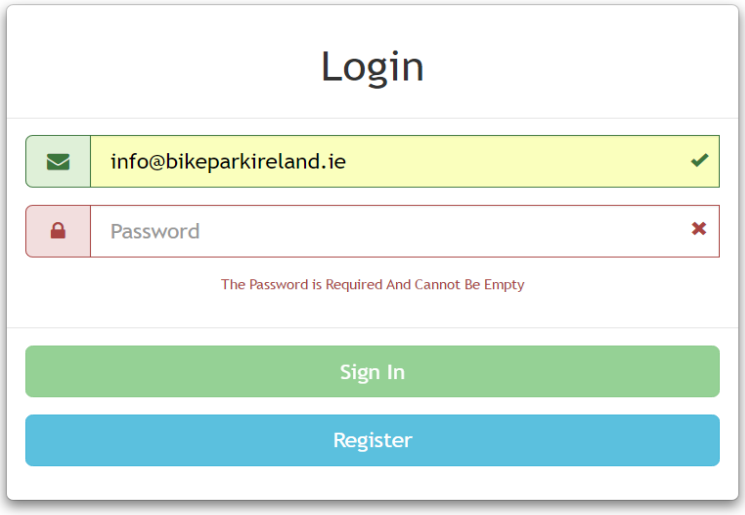

Figure 5.6: Log In Screen

Valid User Credentials/Authorisation Tokens are first encoded using Base64 and then are stored as an object in cache memory.

Even if someone managed to add fake user credentials to cache and gain access to the Web App, none of the features would work as the database requires valid user credentials to access data.

Business Registration

New businesses can register themselves using the Web App Registration Form.

As seen in Figure 5.7, the registration form is fully validated front and back-end using JavaScript, Bootstrap and ASP.NET validation. This prevents the application from crashing and stops invalid data being entered.

Once all fields have been filled in with valid input, the business will be informed that they have successfully registered and can log in.

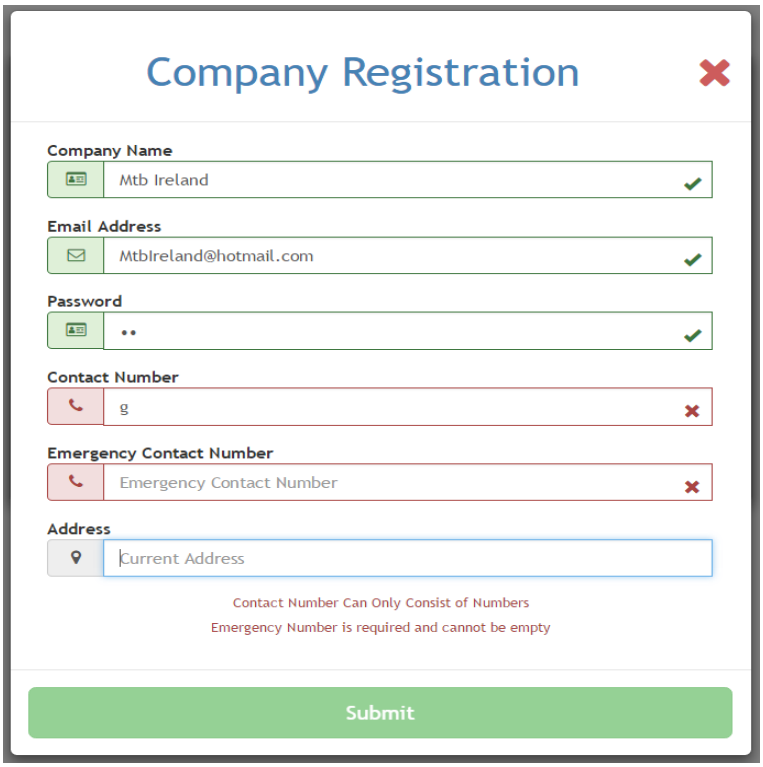

Figure 5.7: Registration Form

Dashboard

The Dashboard is where the analytics are displayed. These analytics can be seen by clicking on any of the four buttons displayed on the page.

1. **Arrivals**
   The "Arrivals" (Figure 5.8) button displays all members currently checked in. This feature only displays members that have checked in that day and will not display members that checked in on previous days.

   This information is vital because it not only shows all members currently checked in but, in the case of an accident, it will provide a fast look-up to identify members.

2. **Top Visitors**
   The "Top Visitors" (Figure 5.9) button displays the top 10 members that have the highest visit total overall. This information is beneficial to the business as it shows who visits the business the most.

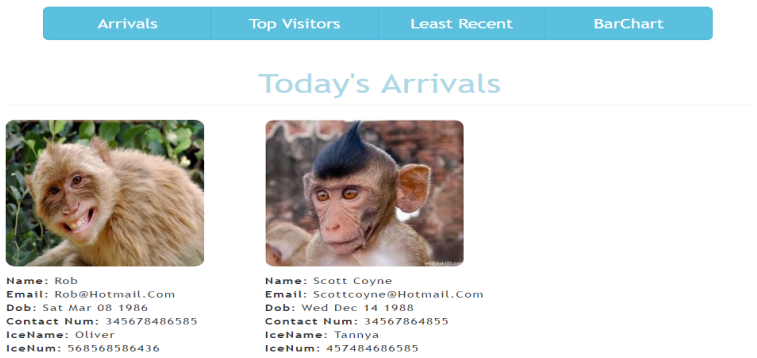

Figure 5.8: Arrivals

Figure 5.9: Top Visitors

3. **Least Recent**
   The "Least Recent" (Figure 5.10) button displays the top 10 members that haven't visited the business in the longest amount of time but have the highest visited total. This information is beneficial to the business as it lets the business target frequent members that have not visited in while with promotions, offers to try and draw them back in.

4. **Bar chart**
   The "Bar Chart" (Figure 5.11) Button displays the busiest months of the year by compiling and sorting all members by visited dates and using this information to get the total visited value for each month.

Figure 5.10: Least Recent

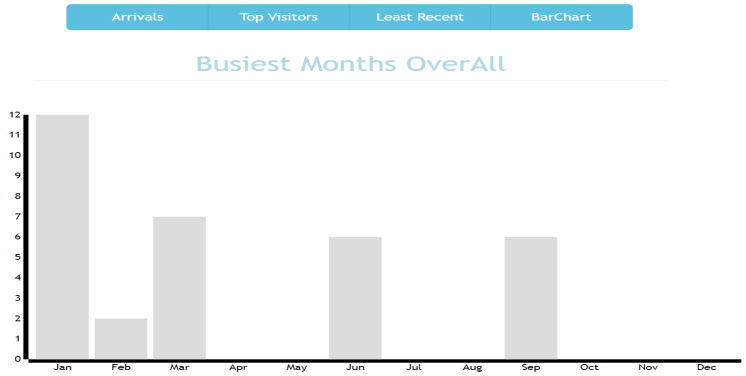

Figure 5.11: Bar chart

Map

The map is where SOS messages sent from members via the Mobile App gets displayed.

Only messages sent from associated members are received and can been seen on the header bar at the top of the Web App as soon as they are sent. This can be seen in the image below:
As seen in Figure 5.12, the number of SOS messages is in the top right hand corner. Also visible is the pin on the Google Map of where the member that sent the message is located.

To view the message sent from the member, the business can click on the pin on the map and can delete the message once read.

Figure 5.13 shows a sample message on the map.

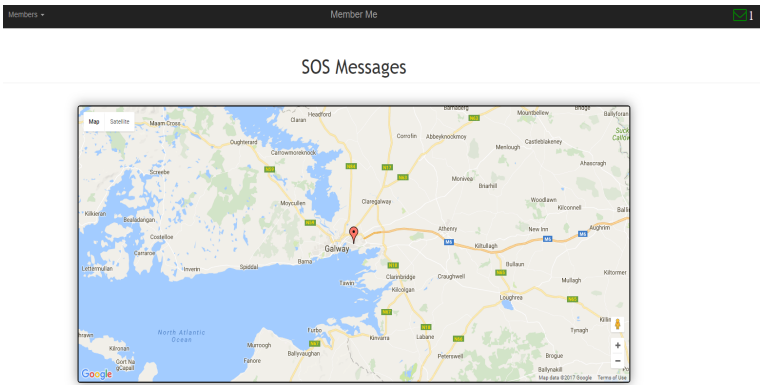

Figure 5.12: SOS Map Page

Figure 5.13: SOS Message's

Arrivals

The "Arrivals Page" (Figure 5.14) on the Web App is where the business can check members in and manage their account information.

This is achieved by scanning the members Mobile App Qr Code or entering the members email address into the input field as seen below:
If the customer is found on the database and is a member of the business, the customers profile information is displayed on the Arrivals page as a drop down modal for the business to validate the customer.

The design is simple as seen in Figure 5.15 and contains two main parts.

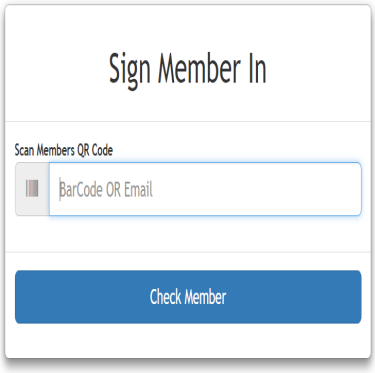

Figure 5.14: Customer Check In Box

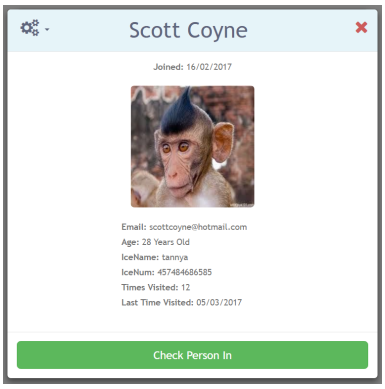

Figure 5.15: Valid Customer

1. **Customer Check In:**

   Once a customer is found and has been validated via there profile picture, the business is then able to use the button "Check Person In".

   This button updates a few different fields on the users profile:

   - **Last Time Visited:** Last date the customer visited the business
   - **Times Visited:** Total amount of times the customer has visited the business
   - **Dates Visited:** This is not displayed on the profile but is used for analytics to find bustiest/quietest months.

   All of the above can be seen in Figure 5.16, demonstrating the checking in of a customer.

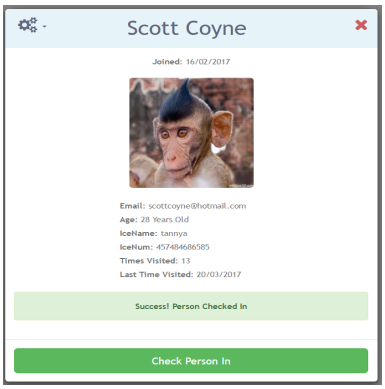

Figure 5.16: Customer Check In

2. **Customer Update/Delete:**

   This section of the Arrivals page is where the business can update and/or delete a customers profile information and membership. This is achieved by clicking the cog icon at the top left of the Customers Profile and choosing from the drop-down list.

   Once an item has been chosen, the modal displays an input box associated to that item with its own unique functionality and validation. The business can then click the Change button to update the customers profile and entry in the database.

   Here is the list of Update/Delete functionality the business has:

   - Name (Update Name)
   - Email (Update Email)
   - Guardian Name (Update or Add Guardian Name)
   - Guardian Number (Update or Add Guardian Phone Number)
   - Add New Membership (Date - 3,6,12 months Paid)
   - Remove Guardian (Removes Guardian name and number)
   - Remove Membership (Removes a paid Membership)
   - Reset Password (Creates a newly generatedm temporary password)
   - Delete Member (Deletes the Member)

The reason for only allowing the contents of the customers profile listed above to be updated/deleted on the business and not the Mobile App was to prevent customers changing important information.

All of the above can be seen in Figure 5.17, demonstrating the process of updating the customers name.

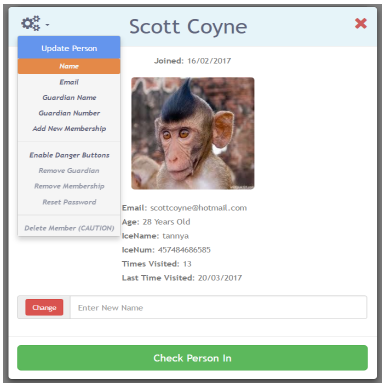

Figure 5.17: Customer Update/Delete

Members

The members page is where the business can add new members to the business as seen in Figure 5.18:

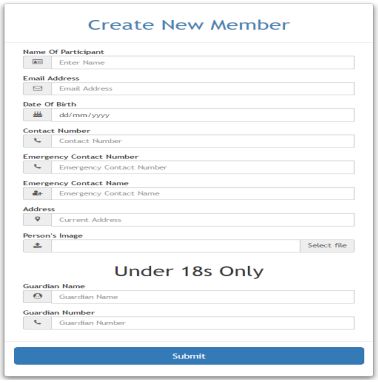

Figure 5.18: Add new Member Form

This is the following information required from the customer:

1. Name
2. Email
3. Date Of Birth
4. Contact Number
5. Emergency Contact Number
6. Emergency Contact Name
7. Address (Current Address)
8. Image (Current picture of the person)
9. Guardian Name (Under 18's Only)
10. Guardian Number (Under 18's Only)
11. Add New Membership (Date's, 3/6/12 Months Paid Membership)

Each of the inputs listed above are required, bar the guardian information depending on if the person is over 18.

As for the Image input box, clicking on the input box opens up file explorer and only accepts specific file formats such as Png, Jpg and Jpeg of max file size 10Mb.

5.5 Mobile Application

This section will discuss the details of the Mobile App. The views of the Mobile App will be referred to as **pages**.

5.5.1 App Architecture

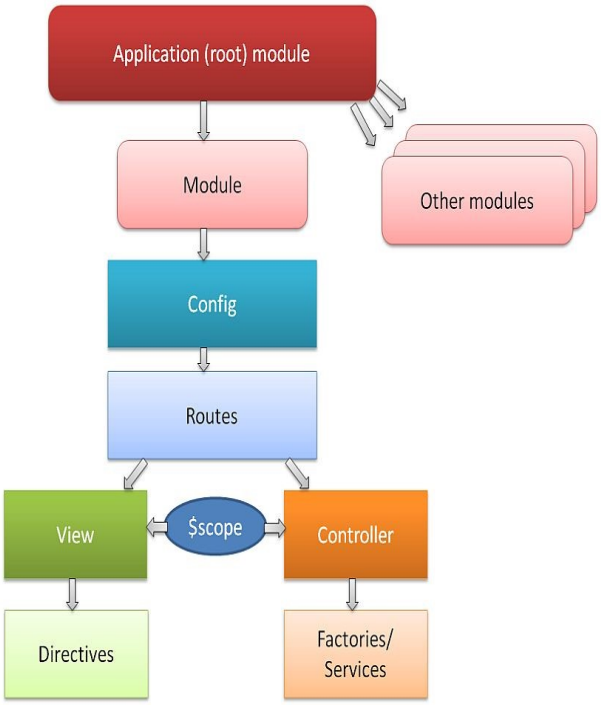

Figure 5.19: Ionic App Components

Ionic apps are made up of a number of interconnected components as described below.

- **Module:** the module is a container that contains the elements of the app.
- **Config:** configures the app and is used to specifiy the routes/states of the app by connecting them to controllers and setting defaults.
- **Routes:** controlled by Config, the routes set up each page of the app and how they can be navigated to.

- **View:** the presentation layer of the app. It is a HTML page which can display data provided by the controllers scope.
- **Controller:** passes data to the view using data binding. The data is passed using a scope variable defined in the controller. The data stored in the scope often comes from a service or factory called by the controller.
- **Factory/Service:** the data layer of an Ionic app which is called by the controller to retrieve data e.g. using a HTTP call.

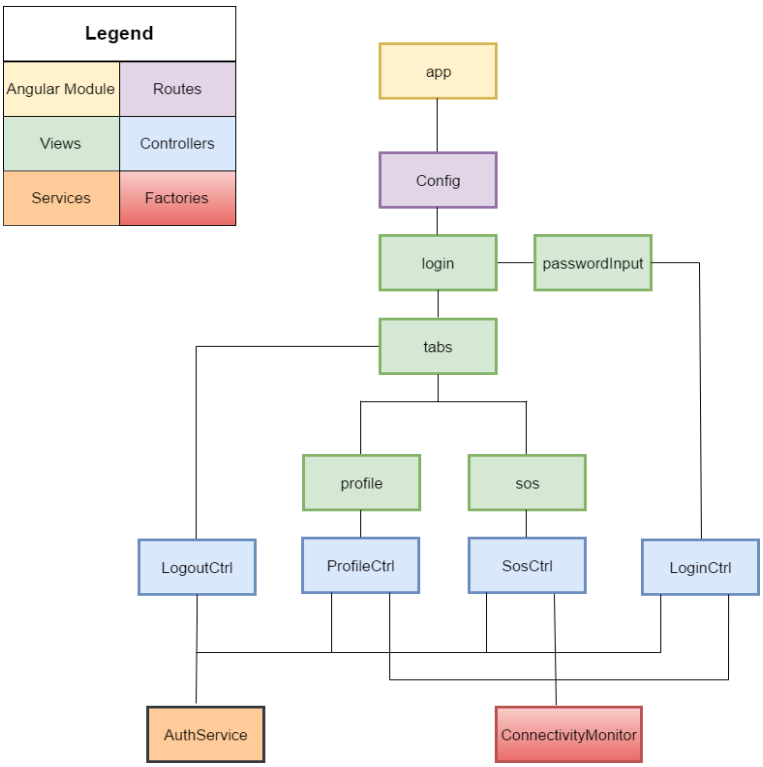

Figure 5.20: App Architecture

The architecture of the Mobile App is shown in figure 5.20. The app is made up of a number of pages which will be shown in screenshots in the next section. The tabs.html page is a type of master page that shows the tabs for navigation for the app on all pages. The Login and Password Input pages do not contain tabs as the user can not navigate the app until they have been signed in. The app also contains a number of controllers for each of the pages, a service which controls the authentication and a factory which checks whether or not the mobile device is connected to the Internet.

5.5.2 App Design

Figure 5.21: App Logo

Figure 5.21 shows the app appearing on a mobile device. A simplistic design was used for the app logo. It contains mountains to give the app an outdoor feel without tying it to a particular outdoor activity and the name of the app is located below it. When the app is started, a splashscreen appears until the app has loaded which is a bigger version of the app logo.

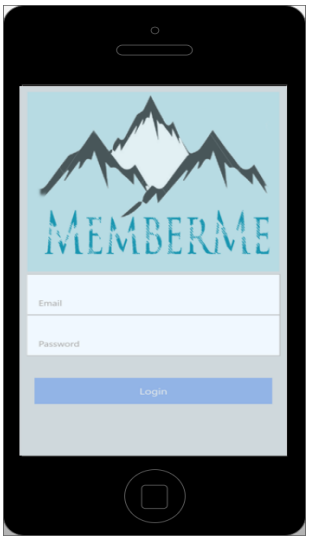

Figure 5.22: Login Page

Figure 5.23 shows the login page where the user is asked to enter their email and password. There is validation present on this page which checks that the user has entered a proper email type and that all fields have been filled in. The login button at the bottom of the page will not become usable until the fields have been filled in with valid values. The user also needs their device to be connected to the Internet in order to sign in. Some of the error screens for this page are shown in figure 5.23.

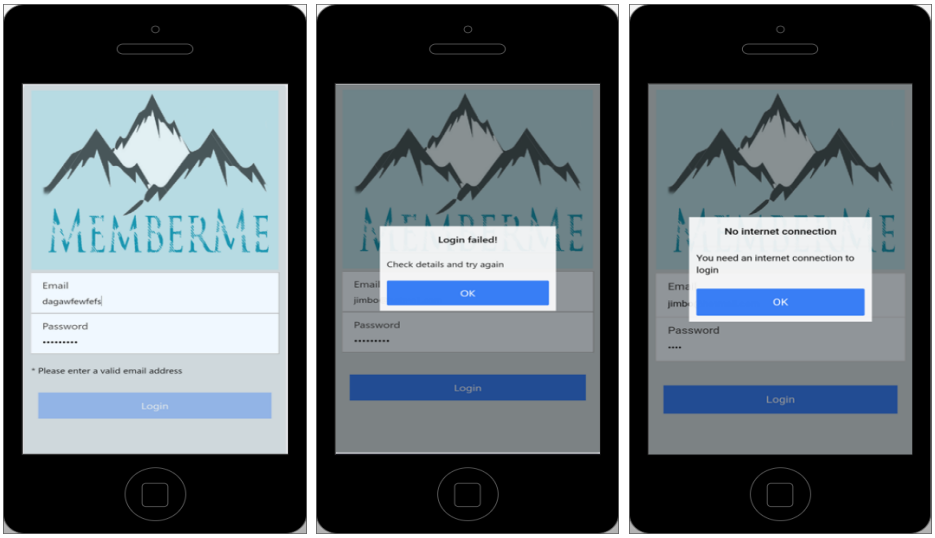

Figure 5.23: Login Error Screens

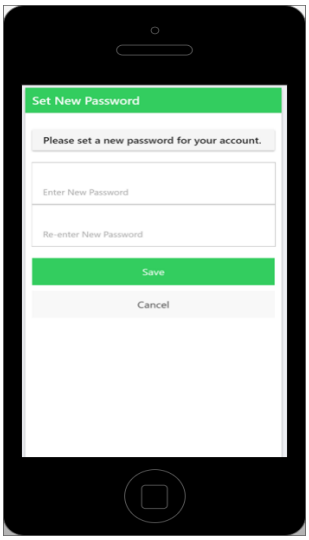

Figure 5.24: New Password Page

When a user signs up with a business they are given a temporary generated password. The first time they log into the app using the temporary password they are brought to the screen shown in figure 5.24 and asked to set a new password. The user is required to enter a new password and then to confirm that password. There is validation present on this page which ensures the passwords match, that a password has been entered and that no invalid password types have been entered (Figure 5.25).

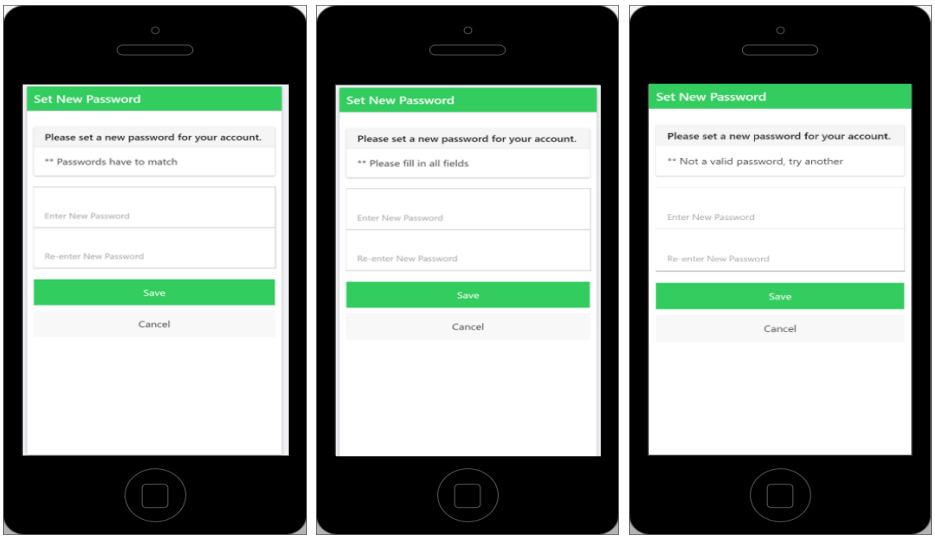

Figure 5.25: New Password Validation checks

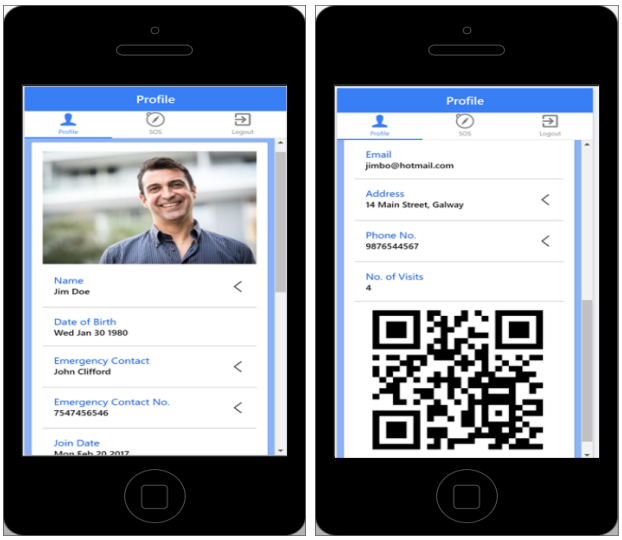

Figure 5.26: Profile Page

When a user successfully signs in they are brought to the profile page (Figure 5.26). This page contains information about the user, including their profile picture, which is retrieved from the database. It also contains the QR code (Figure 5.26) that the user will use to check in at the business they are a member of by scanning it.

Angular evaluated expressions were used to check whether certain properties were valid for a user and whether to show them on the profile page. For example, a user under 18 will have properties for a guardian including a name and contact number. Users over 18 will not have these details and so this information will only be shown for users who have values for this property.

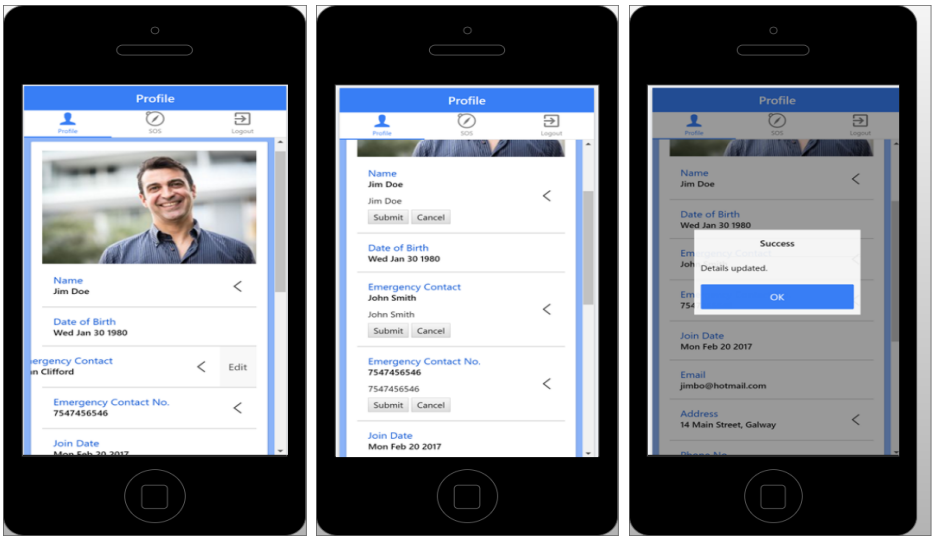

Figure 5.27: Editing the Profile Page

Certain fields of the profile are editable and they are shown with an arrow (Figure 5.27) to indicate that the user should swipe to edit. When the user swipes on applicable fields they are given the option to change details and submit or cancel the changes (Figure 5.27). If they submit, the changes are sent to the server and if successful, an message appears to alert the user that the changes have been applied (Figure 5.27). If the user selects to cancel, the field reverts to its original value. If the device is not connected to the Internet, the user is alerted that the changes were not saved (Figure 5.28).

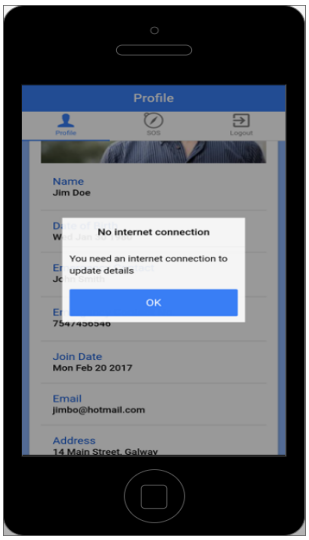

Figure 5.28: Updating without internet

Figure 5.29: SOS Page

Figure 5.29 shows the SOS page. This page of the app is for use in cases of emergency where the user needs to send a message back to the business while they are participating in an activity. The page shows a map with the users current location marked with a pin. The map requires the device to be both connected to the internet and to have GPS switched on. If either of these prerequisites are not met, a message alerts the user (Figure 5.30). Once both these connections are established, the user can pull from the top of the screen to refresh the page and the map will appear.

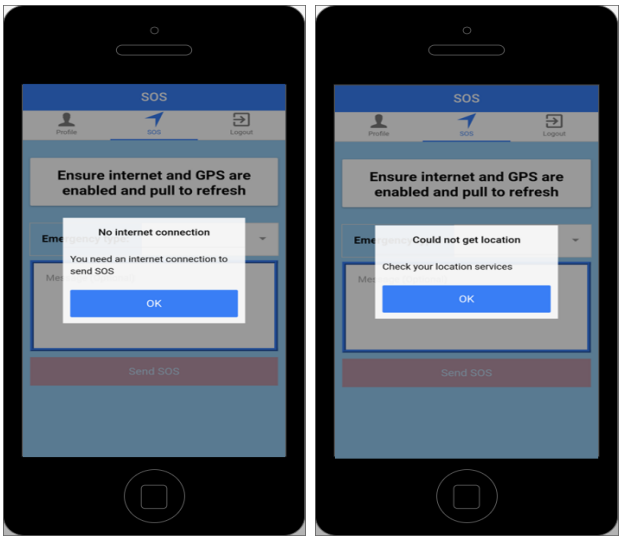

Figure 5.30: SOS Page Connection Alerts

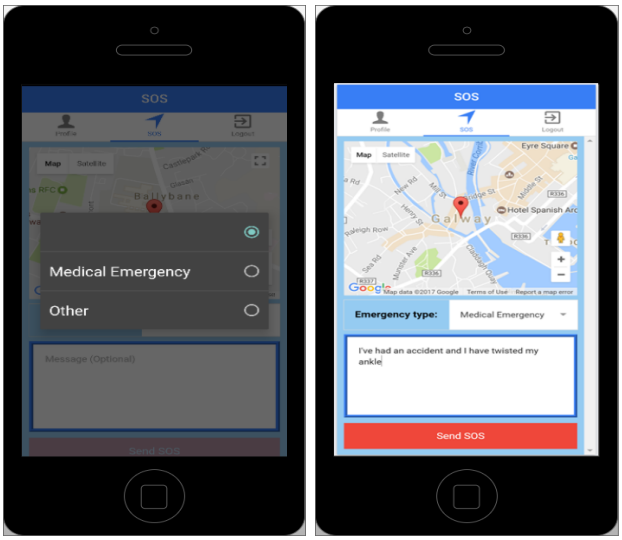

Figure 5.31: SOS Page - Emergency Type & SOS Message

The emergency type option box (Figure 5.31) gives the user the opportunity to enter whether their current emergency is medical in nature or something else. The submit button for this page will not become active until the user chooses an emergency type. They can also enter a message (Figure 5.31) to further explain the issue but that is optional. The user details and GPS location will be sent to the message server with or without a message.

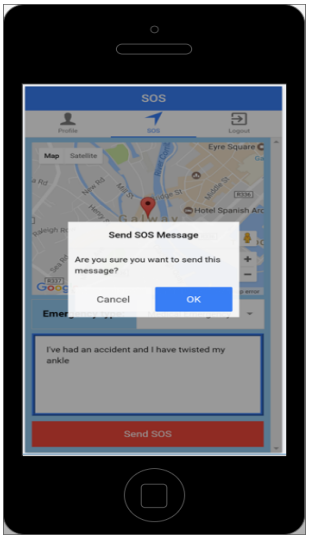

Figure 5.32: SOS Page - Send Message Confirmation

When the user selects to send the SOS message, a confirmation alert shows up on the screen (Figure 5.32). This is to reduce the risk of messages being sent by accident. If the user selects to send the message then an attempt is made to send it to the message server. If the message is successfully sent, a confirmation is shown to the user (Figure 5.33). If an error occurs, this is relayed to the user (Figure 5.33). SOS messages depend on the device being connected to the internet in order to send them and this is displayed to the user (Figure 5.33).

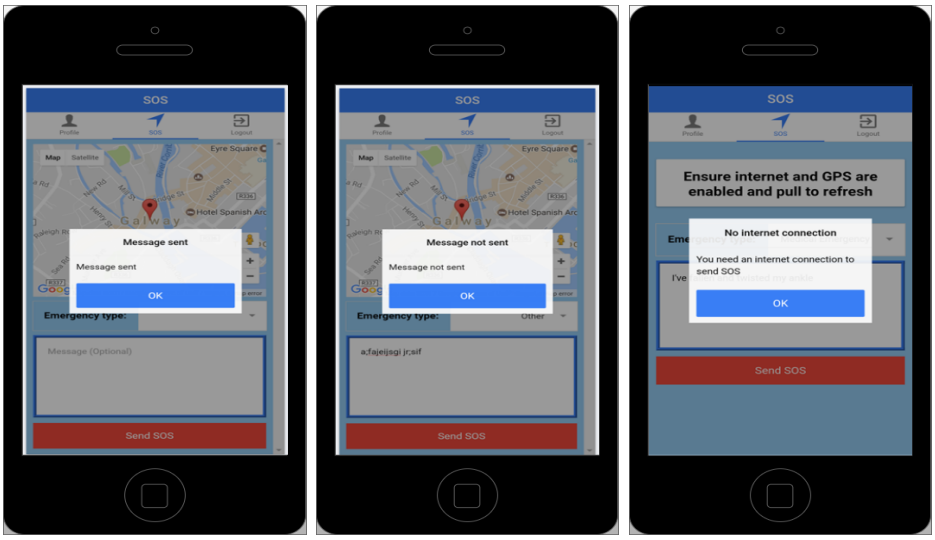

Figure 5.33: SOS Page - Message sending

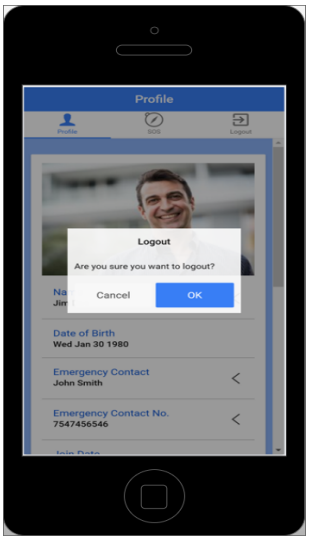

Figure 5.34: App Logout

The user can logout of the app from the Profile page or SOS page by clicking the Logout button on the tab bar. The user is asked to confirm whether they want to logout or cancel as seen in Figure 5.34. If they cancel they are returned to the current page they are on. If they confirm that they wish to logout, the app returns to the Login page.

5.5.3 Code Snippets

In this section, some interesting snippets of code from the Mobile App will be looked at.

Below are some code snippets from the Profile page which is made up of HTML and Angular. The code below defines how the different profile details are displayed on the page. Any fields which are editable have extra code attached to them which allow the item to be swiped and have Submit and Cancel buttons. Some profile details are only visible if they have a value e.g. if the user is over 18 they do not have guardian details attached to their profile so if profile value is null it will not be shown. The visibility of these fields is controlled by **ng-show** and **ng-hide**.

```
<div class="card">
      <div class="item item-text-wrap">
        <img class="profile-image" ng-src="{{profile.imgUrl}}">
        <ion-list show-delete="shouldShowDelete"
        ↪  can-swipe="listCanSwipe">
          <ion-item class="item item-icon-right">
            <i class="icon ion-ios-arrow-left"></i>
            <h2 class="profile-list">Name</h2>
            <h3>{{profile.name}}</h3>
             <ion-option-button class="button-info"
             ↪  ng-click="edit()">
                Edit
             </ion-option-button>
             <div ng-show="toggle">
                    <input type="text" ng-placeholder="Enter
                    ↪  new details"
                    ↪  ng-model="profile.name"></input>
                    <button
                    ↪  ng-click="submit()">Submit</button>
                    <button
                    ↪  ng-click="cancel()">Cancel</button>
                    </div>
          </ion-item>

          <ion-item>
            <h2 class="profile-list">Email</h2>
            <h3>{{profile.email}}</h3>
```

```
        </ion-item>

        <ion-item class="item item-icon-right">
          <i class="icon ion-ios-arrow-left"></i>
          <h2 class="profile-list">Address</h2>
          <h3>{{profile.address}}</h3>
          <ion-option-button class="button-info"
           ↪  ng-click="edit()">
              Edit
            </ion-option-button>
            <div ng-show="toggle">
                  <input type="text"
                   ↪  ng-model="profile.address"
                   ↪  placeholder="Enter new details"
                   ↪  ></input>
                  <button
                   ↪  ng-click="submit()">Submit</button>
                  <button
                   ↪  ng-click="cancel()">Cancel</button>
                  </div>
        </ion-item>

        <ion-item ng-hide="!membership">
          <h2 class="profile-list">Membership End Date</h2>
          <h3>{{membership}}</h3>
        </ion-item>

        <ion-item ng-show="profile.guardianName!='null'">
          <h2 class="profile-list">Guardian Name</h2>
          <h3>{{profile.guardianName}}</h3>
        </ion-item>

        <ion-item ng-show="profile.guardianNum!='null'">
          <h2 class="profile-list">Guardian Number</h2>
          <h3>{{profile.guardianNum}}</h3>
        </ion-item>
        <ion-item><qr text="qrcode" class="align-qr"
         ↪  size=size></qr> </ion-item>
      </ion-list>
    </div>
  </div>
```

The Profile page calls three functions that are contained in the controller, **ProfileCtrl**, which is shown below.

- The **Edit** button calls the edit function which closes the slider on the page, allows input to the field and reveals the Submit and Cancel buttons.
- The **Cancel** button reverts the field to read-only and the field values to the original values i.e. ignoring any inputs before the cancel.
- The **Submit** button checks to ensure the device is connected to the Internet and if it is not, a popup alert will appear. The field values will revert to the original values. If an Internet connection is detected, the controller will call the **updateProfile** function in the service, **AuthService** and pass it an object containing the profile details. A popup will appear on screen telling the user whether the details were successfully saved or not.

```
$scope.edit = function() {
  $ionicListDelegate.closeOptionButtons(); //closes the
  ↪ slide effect
  $scope.toggle=!$scope.toggle; //turns toggle on/off
}
//cancel method on profile edit
$scope.cancel = function(){
  $scope.toggle=!$scope.toggle; //hide input box
  profileData=window.localStorage.getItem('profile');//get
  ↪ profile details from local storage
  profileData=JSON.parse(profileData); //parse JSON object
  $scope.profile=profileData; //save JSON object to $scope
  ↪ variable
  $scope.tempProfile.icePhone =
  ↪ parseInt(profileData.icePhone); //convert phone
  ↪ number string to number
  $scope.tempProfile.phone = parseInt(profileData.phone);
}
//submit function for profile edits
$scope.submit = function() {
  if(!ConnectivityMonitor.isOnline()){
    var alertPopup = $ionicPopup.alert({
        title: 'No internet connection',
```

```
            template: "You need an internet connection to
             ↪  update details"
            });

        $scope.toggle=!$scope.toggle; //hide input box
        profileData=window.localStorage.getItem('profile');
        profileData=JSON.parse(profileData); //parse JSON
         ↪  object
        $scope.profile=profileData; //save JSON object to
         ↪  $scope variable
        $scope.profile.icePhone =
         ↪  parseInt(profileData.icePhone);//convert phone
         ↪  number string to number
        $scope.profile.phone = parseInt(profileData.phone);
    }
    else{
      AuthService.checkAuthOnRefresh(); //check if token has
       ↪  expired
      $scope.toggle=!$scope.toggle; //hide input box
      $scope.profile.tempEmail=$scope.profile.email; //set
       ↪  tempEmail for updateMethod on server
      $scope.profile.icePhone =
       ↪  $scope.tempProfile.icePhone.toString();//convert
       ↪  phone number string to number
      $scope.profile.phone =
       ↪  $scope.tempProfile.phone.toString();
      profileData=$scope.profile;
      AuthService.updateProfile(profileData).then(onSuccess,
       ↪  onError); //call AuthService method updateProfile
       ↪  and pass profileData object
      }//end else
    }//end submit
    var onSuccess = function(){
        var alertPopup = $ionicPopup.alert({
            title: 'Success',
            template: "Details updated."
            });

        profileData=$scope.profile; //save updated $scope obj
         ↪  to profileData obj
```

```
    window.localStorage.setItem('profile',
    ↪ JSON.stringify(profileData)); //set updated
    ↪ profile details to local storage
}

var onError = function(){
    var alertPopup = $ionicPopup.alert({
        title: 'Could Not Update',
        template: "Details not updated. Please try again."
        });
}
```

Chapter 6

System Evaluation

This chapter will evaluate the software developed in the project and will describe the testing that was carried out in order to evaluate all of the system componants. It will also highlight the limitations of the software and analyse where there are opportunities to improve upon the way that the system has been developed.

6.1 Testing

6.1.1 System Testing

System testing on the Web App involved testing it locally on multiple Browsers and devices before deployment. It was also tested at different days and times to see if all the features were still working correctly after it had been deployed on Azure. Testing was also carried out on mobile devices through the browser.

System testing for the Mobile App was done using Android Studio[47] which allowed for testing on a variety of different devices using the in-built emulator. Devices could be created to represent a variety of screen sizes and all Android Operating Systems. This meant that the Mobile App could be tested on both older and the most up-to-date systems. When all functionality was confirmed to be working on all emulated devices, testing began on physical devices. This involved building an APK of the Mobile App and giving a copy to a group of classmates (with Android devices) to install. We gave a basic demonstration of the functionality of the Mobile App and asked each user to try out a list of instructions including logging in, editing profile details, sending an SOS message and logging out. Feedback was given on any issues and bugs that were noticed.

6.1.2 Usability Testing

For both the Web and Mobile App, the system was tested at an outdoor activity business. Given log in credentials, the owner of the business began using the system. He testing out the different functionality including updating, deleting and adding new members as well as the arrivals system and SOS messaging page. Feedback received from him was that he found the system simple and intuitive to use. He also had a positive reaction to the SOS messaging functionality and said it was something that he could see being utilised regularly.

As mentioned in System Testing, a group of our classmates were asked to install the Mobile App and carry out basic instructions. Included in the feedback was a discussion on usability of the Mobile App. The response was generally favourable but we also received some valuable critiques. One such insight was about editing profile details. We did not have any indicator that fields were editable which we as developers knew but had not considered it from the users perspective. We took this feedback on board and added a visual cue (a swipe arrow) to the editable fields.

6.1.3 Regression Testing

Regression Testing was done as part of the iterative approach we choose when developing the project. Regression testing of the Main Server was carried out continuously throughout development of the project. Whenever new functionality (such as new or amended routes) were added to the server, it was tested locally to ensure no bugs resulted from it and that it continued to work as before. Once the testing was successful, the Server would be deployed to Heroku.

6.2 Limitations

Encryption on personal information used in the project is base64 encoding. This was done to avoid sending or storing plain text personal data in the database. This is not the industry standard on encryption for storing personal data in databases but limited time resulted in this being the approach used. Other alternatives researched suggested using software like ASP.NET Identity to handle encryption of personal information.

Log in credentials for the Web App were stored as a base64 encoded object which was stored in cache and retrieved when navigating to each page. Although storing and retrieving data from cache is fast, the user has no control on when cache is cleared. When this happens, the user is required to log into the system again. An alternative to this would be to use local storage, session storage or cookies to store user data but more research on web credential storage and encryption best practice would be required.

The Server Sent Event (SSE) technology used to send and receive messages from the Mobile App to the Web App via the message server has limitations. The technology only allows for one way messaging which means that messages sent from the Mobile App to the message server would return a successful response. However, no response was given to the Mobile App user about whether the business received or opened the message. Another limitation of SSE is that there is no way to identify on the server what business a user is associated with. This means that messages that are sent from Mobile App users are received by all businesses that are currently logged in and have a connection to the message server. These messages are checked in the Web App to see whether the message sender is a member of the business so as not to display messages intended for a different business. An alternative method would be to use WebSockets to achieve full-duplex communication between Mobile App users and businesses.

The Web App is deployed on a free tier on Azure which means that SSL is not implemented on the website. This had an effect on the map page where the users geolocation is requested and because SSL is not present, the geolocation information could not be provided.

Having examined all parts of the system, the final chapter will summarise the conclusions of the project.

Chapter 7

Conclusion

This chapter will summarise the project in terms of the objectives of the initial proposal and will discuss the findings and outcomes of the finished project.

This project was proposed as a solution to the issue of a time-consuming check in procedure being used at an outdoor activity business. This issue led to delayed activity start times and took staff away from other tasks in order to get customers checked in.

It also attempted to resolve the difficult issue of customers being injured or in a situation where they needed help from staff.

The objectives set out in this project were to create a membership system for a business that would be efficient and easy to use for both the business and the members. The system that was built consisted of a Web App for the business users, a Mobile App for the customer users and a back-end system comprising of a Main Server, a Message Server and a Database.

The objectives set out in the first chapter were reached with additional functionality added throughout the project.

The following are the findings and outcomes from this project:

- Encryption and cryptography need to be researched more in-depth to improve security of the system
- Log in credentials should be stored in a more stable manner

- Websockets would provide a full-duplex messaging system with which push notifications could be utilised
- Scrum Agile Methodology provided structure and helped with the time management of the project
- GitHub issue tracker was an effect means of tracking and resolving bugs and issues during development
- GitHub made deployment and development easier by separating out the different components into different branches which allowed team members to work independently

All the objectives set out for the project were reached and specifications were met and adapted when necessary.

The whole development of the system from start to finish proved to be a challenging and rewarding experience. As a result of the large scope of project, it provided a good opportunity to explore and learn a range of different technologies. Though this did increase the learning curve during development, the team broke up each section into smaller parts until the features needed were implemented.

Another rewarding experience of the project was the teamwork aspect. The extent of the scope of this project far exceeded any other group projects carried out in the past. Each member of the team brought something different to the project ranging from different technology interests, ideas and talents during development. This resulted in the creation of a well thought-out, simple and unique membership system that not only functions well but that the team is proud of.

Chapter 8

Appendices

8.1 Source Code

The source code for the projects can be found on GitHub here or at https://github.com/codevonnie/fourthyearproject.

8.2 Installation Instructions

Installation instructions are contained in the ReadMe files on the GitHub repositories.

Bibliography

[1] Apache, "Cordova overview." https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[2] T. Point, "Angularjs - mvc architecture." https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm.

[3] S. Coyne and Y. Grealy, "Readme." https://github.com/codevonnie/fourthyearproject/blob/master/README.md.

[4] S. Coyne and Y. Grealy, "Main server." https://github.com/codevonnie/fourthyearproject/tree/Master-MainServer.

[5] S. Coyne and Y. Grealy, "Message server - api." https://github.com/codevonnie/fourthyearproject/tree/Master-MessageServer.

[6] S. Coyne and Y. Grealy, "Web application." https://github.com/codevonnie/fourthyearproject/tree/Master-WebApp.

[7] S. Coyne and Y. Grealy, "Mobile application." https://github.com/codevonnie/fourthyearproject/tree/Master-MobileApp.

[8] Microsoft, "Azure." https://docs.microsoft.com/en-us/azure/app-service-web/app-service-web-get-started-dotnet.

[9] Microsoft, "Asp.net." https://docs.microsoft.com/en-us/aspnet/overview.

[10] Microsoft, "Linq." https://msdn.microsoft.com/en-us/library/bb308959.aspx.

[11] S. C. Kleene, "Regular expressions." http://www.regular-expressions.info/.

[12] E. Source, "Event source." https://developer.mozilla.org/en-US/docs/Web/API/EventSource.

[13] N. S. Itai Lahan, Tal Lev-Ami, "Event source." http://cloudinary.com/documentation/dotnet_integration.

[14] RestSharp, "Restsharp." https://github.com/restsharp/RestSharp.

[15] J. T. Mark Otto, "Bootstrap." http://getbootstrap.com/getting-started/.

[16] V. O. Mike Bostock, Jeffrey Heer, "D3.js." https://d3js.org/.

[17] Google, "Google maps api." https://developers.google.com/maps/.

[18] npm, "What is npm?." https://docs.npmjs.com/getting-started/what-is-npm.

[19] Ionic, "All about ionic." http://ionicframework.com/docs/guide/preface.html.

[20] M. Foundation, "Html." https://developer.mozilla.org/en-US/docs/Web/HTML.

[21] M. Foundation, "Css." https://developer.mozilla.org/en-US/docs/Web/CSS.

[22] Sass, "Sass (syntactically awesome stylesheets)." http://sass-lang.com/.

[23] M. Foundation, "Javascript." https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[24] E. International, "Ecma international." http://www.ecma-international.org/.

[25] AngularJS, "Angularjs." https://angularjs.org/.

[26] Bower, "Bower - a package manager for the web." https://bower.io/.

[27] sachinjain024, "Difference between grunt, npm and bower (package.json vs bower.json)." http://stackoverflow.com/questions/21198977/difference-between-grunt-npm-and-bower-package-json-vs-bower-json/21199026#21199026.

[28] Gulp, "gulp." http://gulpjs.com/.

[29] R. Gontovnikas, M & Chenkie, "angular-jwt." https://github.com/auth0/angular-jwt.

[30] J. Antala, "angular-qr." https://github.com/janantala/angular-qr.

[31] Google, "Develop android." https://developer.android.com/develop/index.html.

[32] A. Inc, "Start developing ios apps (swift)." https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/.

[33] Salesforce, "Heroku." https://www.heroku.com/home.

[34] N. Foundation, "Node.js." https://nodejs.org/en/.

[35] N. Foundation, "Express." https://expressjs.com/.

[36] M. Buttigieg, S & Jevdjenic, *Learning Node.js for Mobile Application Development*. Packt Publishing Ltd, 2015.

[37] Auth0, "Auth0." https://auth0.com/.

[38] D. Crockford, "Introducing json." http://www.json.org/.

[39] E. Internations, "The json data interchange format." http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[40] I. Neo4j Technology, "Neo4j." https://neo4j.com/.

[41] I. Neo4j Technology, "Intro to cypher." https://neo4j.com/developer/cypher-query-language/.

[42] S. S. Events, "Server side events." https://www.w3schools.com/html/html5_serversentevents.asp.

[43] I. Postdot Technologies, "Postman." https://www.getpostman.com/.

[44] R. Hat, "What is openshift?." https://developers.openshift.com/.

[45] I. MongoDB, "Mongodb — for giant ideas." https://www.mongodb.com/.

[46] Github, "Github." https://github.com.

[47] Google, "Android studio." https://developer.android.com/studio/index.html.