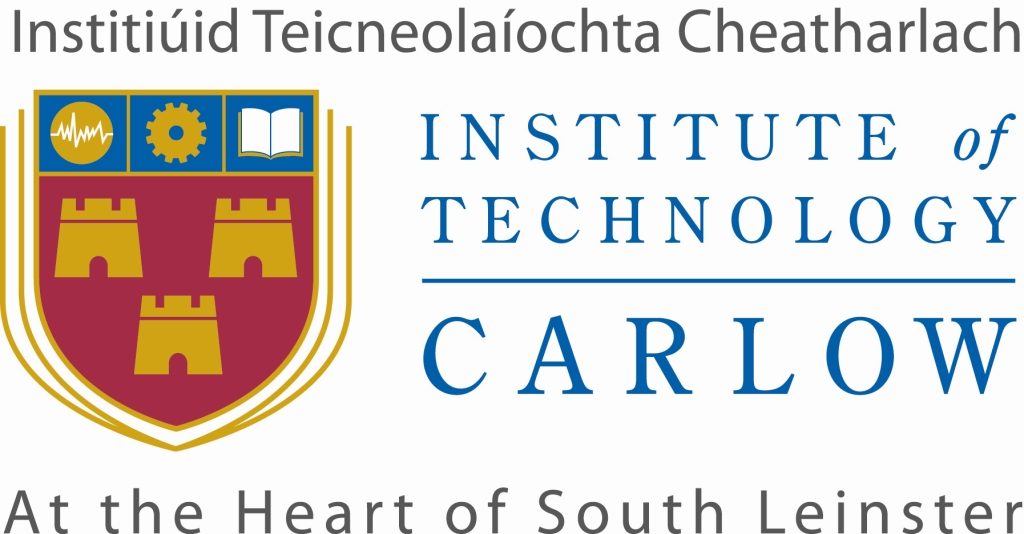
CAIRDE

**Car Sharing App**

Final Report



18th April 2018

BSc (Hons) Software Development

**Name:** Dylan Lawlor

**Year:** 4th year

**Student ID:** C00197013

**Supervisor:** Lei Shi

# Abstract

The purpose of this document is to provide an report on the final project including the functionality that is present in the finished product. It will also include sections that provide the authors reflection on their overall project experience, and their learning outcome.

Contents

[Abstract 1](#_Toc511800651)

[Introduction 3](#_Toc511800652)

[Project Description 4](#_Toc511800653)

[CAIRDE & Car Sharing 4](#_Toc511800654)

[Features and Functionality 4](#_Toc511800655)

[Front End: Mobile Application CAIRDE 4](#_Toc511800656)

[Back End: Azure App Services & SQL Server 18](#_Toc511800657)

[Conformance to Specification and Design 19](#_Toc511800658)

[Functional Specification Conformance 19](#_Toc511800659)

[Design Document Conformance 19](#_Toc511800660)

[Design Updates 19](#_Toc511800661)

[Overall Design 19](#_Toc511800662)

[UI Flow 24](#_Toc511800663)

[Description of Learning 25](#_Toc511800664)

[Technical 25](#_Toc511800665)

[Personal 26](#_Toc511800666)

[Review of Project 27](#_Toc511800667)

[What went right 27](#_Toc511800668)

[What went wrong 27](#_Toc511800669)

[Outstanding/Missing Work 27](#_Toc511800670)

[Technology Changes 28](#_Toc511800671)

[What the author would do differently 28](#_Toc511800672)

[Future Features 28](#_Toc511800673)

[Acknowledgements 29](#_Toc511800674)

[References 30](#_Toc511800675)

# Introduction

The purpose of this document is to detail the results of the authors final year project, the Car Sharing App CAIRDE. The document will be split into several sections and will include a description of the final project, an in-depth look into the functionality present in the app, and a section on the author thoughts and reflections on their experience during the course of this project.

The first section – Project Description – will give a detailed overlook of what CAIRDE is, its main functionality and how this was implemented, as well as provide screenshots from the completed app running on a mobile device. It will also detail any functionality added to the app later in the development that was not covered in earlier documents.

The next section – Conformance to Specification & Design – will examine the earlier documents (Functional Specification & Design Document) and compare them with the completed project. This section will highlight any changes or differences that were made during development and explain the reasons as to why these decisions were made.

This will be followed by a section describing the learning outcomes for the author from both a technical point of view in new skills gained or experience of new technologies, an from a personal point of view in what personal skills were gained.

Finally the author will provide a full review of the project. This section will describe the authors thoughts on what they feel went well, what didn’t go to plan, things they would have done differently and things they would like to have added. It will also give their opinion on the technologies used in the development of the project.

# Project Description

## CAIRDE & Car Sharing

CAIRDE is a car sharing app developed for Android designed to facilitate and encourage car sharing. Car sharing is the act of several people travelling together in one vehicle to a mutual destination. The benefits of car sharing are multiple and include being environmentally cleaner by reducing the number of cars on the road, increased safety for the same reason, and greater social inclusion by encouraging people to communicate and share long commutes with other members of society, as well as providing an opportunity to those who don’t or can’t drive to have the same opportunities as those who do.

The app achieves this by allowing users to register with the service as a user and allow these users to locate each other and request or offer a lift on the journey. In this way a user may be a Passenger or a Driver. The user will be able to use one account to access both sides of the app, depending on their needs.

The driver route will allow the user to create journeys, advertising a number of spaces in the vehicle for each particular journey, as well as the date and departure time for the trip. They can also view passenger applications for lifts and accept or reject applications.

The passenger route will allow the user to search for lifts or create a request for a lift. They can also view details about their upcoming journeys. They can apply for seats on a journey, and they can review completed journeys.

This application is aimed primarily at those groups of people that commute regularly to work or school.

## Features and Functionality

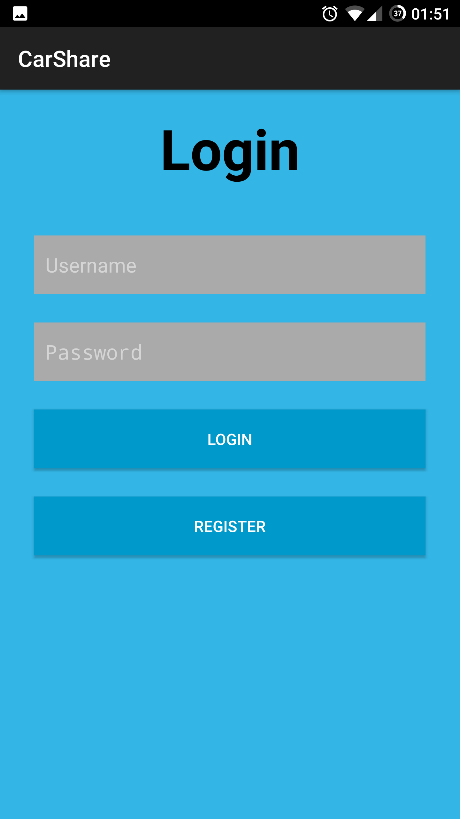
CAIRDE is a mobile application written in C# for Android using Xamarin, and the backend server is an Azure App Service and SQL server hosted on Microsofts Azure cloud platform. This section will describe both ends of the project, the client side frontend and the cloud backend.

### Front End: Mobile Application CAIRDE

#### Overview

The application was created with Xamarin for Microsoft’s Visual Studio and written primarily in C# using the .NET framework. XML is used for the layout of the UI and some other configuration files. The app is developed specifically for the Android platform, but by nature of Xamarin’s push towards cross-platform compatibility with Xamarin.Forms much of the code-base is readily transferable to other platforms. The app also required the use of maps and location services to provide the bulk of the service features and for this Google Maps was used. Maps was directly implemented into the application through use of the Google Maps API, which required getting an API key from Google and associating it with the application. From there several packages were added to the project to allow the app to use maps functionality, including the Google Play SDK which controls all access to Google services within the app. Reverse geo-location was also done in the app, that is the process of taking a co-ordinates value and translating it into a real-world address, and this was achieved by using the public maps API over Http using a custom method.

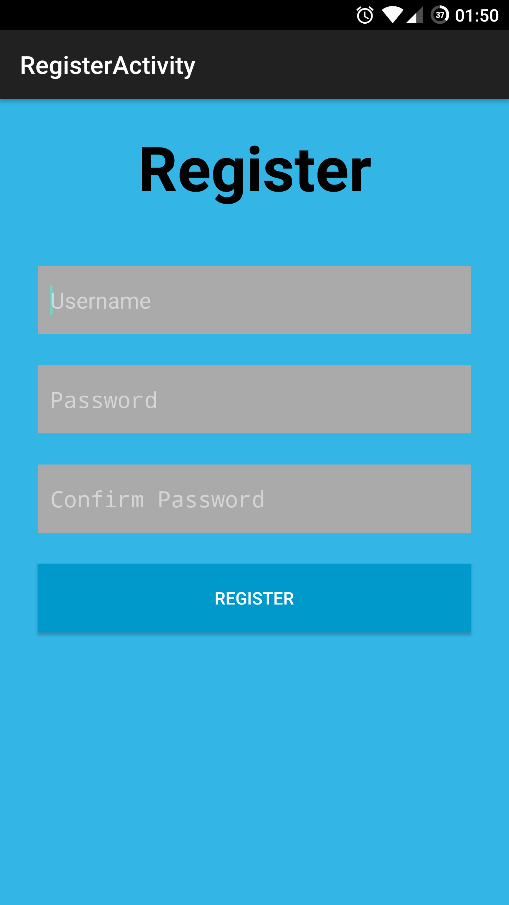
#### Login



The login screen is the start-up screen for when the user starts up the application and from here they have the choice to register to CAIRDE if they have not done so before or having previously registered can login with their credentials. The credentials that are entered are checked by the application against the database through the Azure App Service (more on this later, but it will be mentioned a lot in the following screens). Valid credentials will log the user in and present them the next screen, either the Set Up Profile screen if this is the first time logging in, or the main page if a returning user.

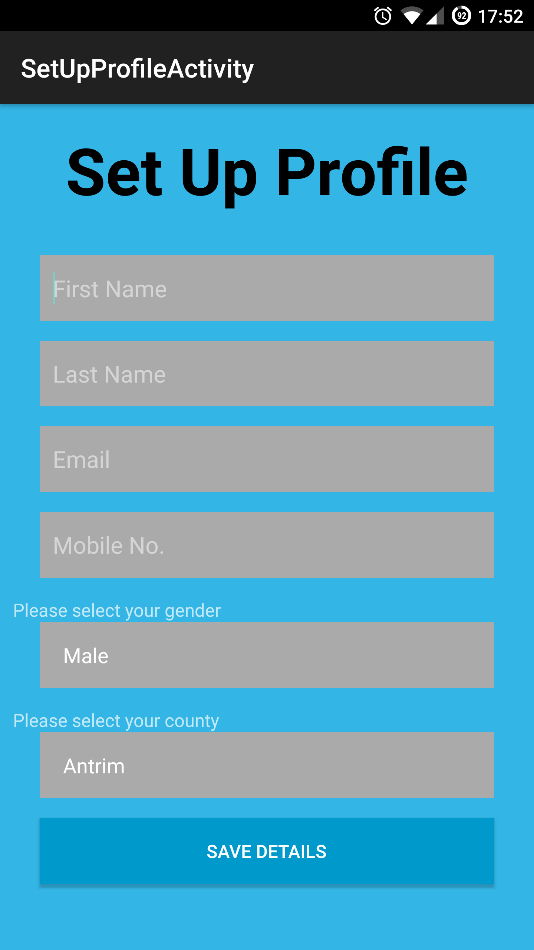
For the user’s convenience their login details are saved upon closing the app so that restarting the app they do not need to re-enter their details. This small cost to security (another person who gains possession of the phone could log into their account) is offset by the increase in convenience, and the fact that even malicious access to the app will not reveal any valuable information (Financial or identifiable information, other than name) about the user, since the app does not store any.

#### Register



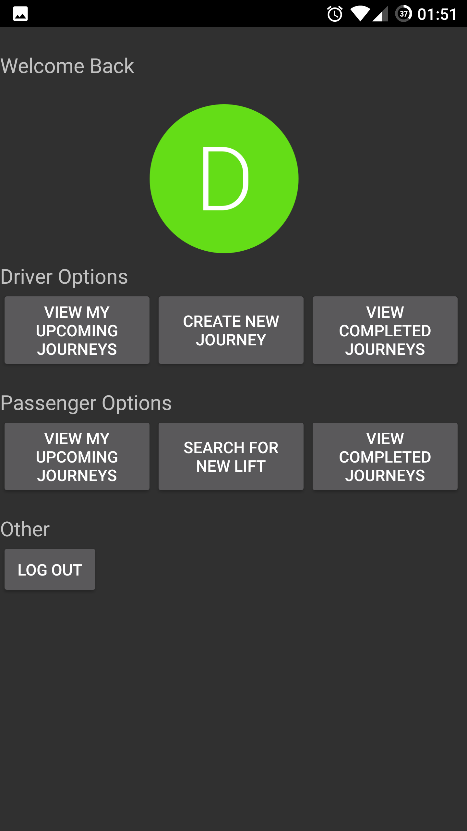
The registration screen is reached from the login screen and provides a form the user can fill out to register to the service. Username is unique to the system and is used solely for login into the app, therefore it is not necessary for the user to use their name for the username (as it will most likely be taken already) but rather something more unique yet memorable like their email address. The username is never shared with other users so it is private. The user will also register a password to the account, this password is encrypted by the app using a salted hash. This is implemented through an adaption of a public method hosted on GitHub that the author was recommended in lieu of creating his own method, as this is generally seen as unsafe behaviour for non-crypto security experts. Providing the username is not already taken, and the two password entries match, the username and password are then passed to Azure where they are stored in the database. The user is returned to the login screen where they can now log-in for the first time.

#### Set Up Profile



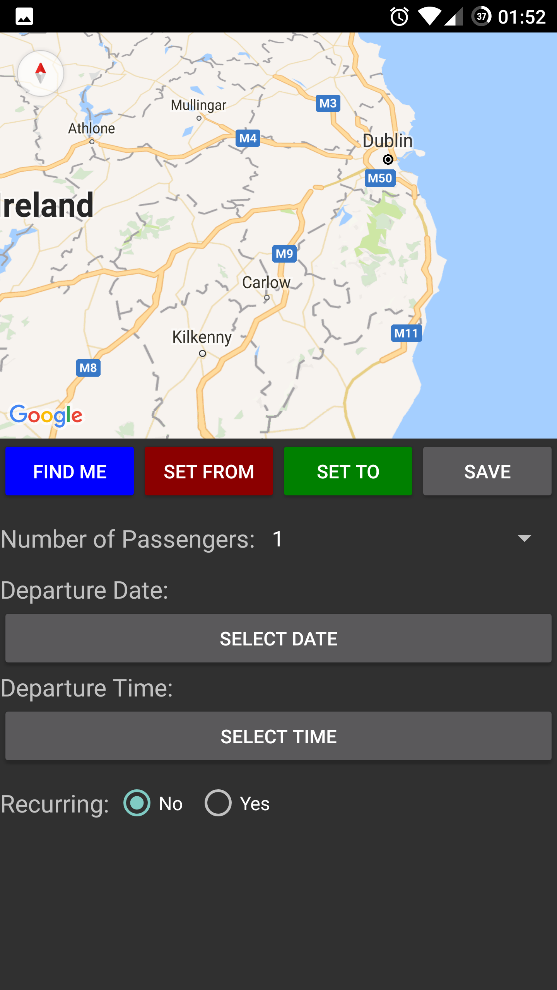
This screen is typically shown the first time a user logs into the app, it may also be shown in situations where the user failed to complete this step on previous log-ins. This screen requests the user fill in some basic details about themselves, and submit these details using the button at the bottom of the page. Gender and County are chosen from drop downs, with an option to not specify gender for those who would prefer this. Upon submitting the information the user is taken to the main menu.

#### Main Menu



The Main Menu is typically the second screen the user will see and from here they can access all of the apps functionality within one button click. The profile icon is automatically assigned based on the users first name. The log out button will return the user to the login screen, and each of the other button will begin that function. This page is returned to automatically upon completing most activities. It is designed so that the user can get to where they want quickly and clearly. This screen is the beginning point of each of the next 6 screens.

#### Create a Journey



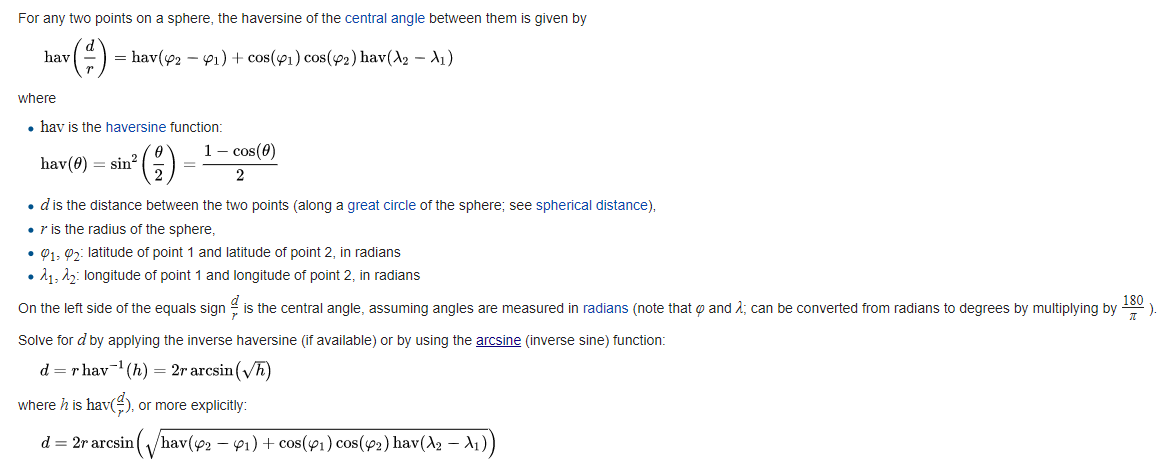
This screen allows the user to create a journey as a driver, that is to advertise space on a journey. By scrolling the map and setting markers for both the departure point and destination the user creates the route. This is an improvement on similar service which require a user to type a specific address. A user is under no obligation to be so precise if they do not need to be in this instance, as they can drop a marker near their locations and users will still be able to find the journey thanks to the range method of search for a lift (More on that in the Search for a Journey section). Markers are created at a certain Latitude and Longitude, and these values can be called from accessing the markers later in the code. Only one marker of each type can exist at once, clicking set from multiple times will use only the latest location. Then the user simple completes a few more controls and they can save the Journey. In the case of recurring journeys (Journeys along the same route at the same time but different days) users can hit the toggle to set recurring to yes, and enter as many dates as they wish.

Saving the Journey does a few things. First the Latitude and Longitude of both the departure location and destination are gotten from the marker, and then reverse geo-coded into an address to be saved in the Journey Table. The Latitude and Longitude values are also stored. This allows the app to quickly recreate the journey later including the markers on the map, as well as use the numerical values for latitude and longitude for comparison during searches. Once everything is set the record is added to the database through the Azure App Service.

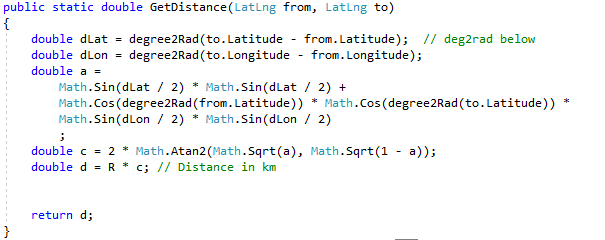
#### Searching for a Journey

This screen represents the main functionality of the application. It allow the user to setup a Journey much like in the previous Create a Journey screen, but this time includes a toggle for leaving time, allowing the user to find any journey that departs around their chosen time. For example a user chooses a departure time of 9.00am. The toggle Leaving Before would find all matching Journeys leaving between 8.00am and 9.00am, Leaving After would be between 9.00am and 10.00am, and +/- 30min does exactly that, between 8.30am and 9.30am. This functionality gives the user flexibility when searching, without having to navigate many menus or enter multiple times. Below the toggle is the range finder, which sets the maximum allowed variance in location for searching for a lift. By using this range finder the user can easily find a lift within their acceptable area. The map also displays the selected area, further increasing the usability.

The range finder works by calculating a minimun and maximum value that both the latitude and longitude have to fall within. These values are found by multiplying the selected range by a certain value that converts the distance into a latitude or longitude figure. The problem is that there is no set ratio as an increase of .5 latitude at the equator amounts to a greater distance in kilometres than the same .5 latitude in Ireland. To overcome this the author used the Haversine formula which determines the distance between any two points on a sphere, given their latitude and longitude. The haversine formula is shown in the following image.



Taking this formula and converting it into a method to find the distance between two points the author could now calculate the difference between two locations in kilometres, but this would not work quickly enough for a search, having to go through every journeys start and end point and calculate if the distance from the search journey is greater than the range. The author decided it would be better to find a flat ratio that could be used to directly compare the latitude and longitude values. Using the following method the author calculated two constant values for the application that are used when calculating what Journeys fall within the accepted range.



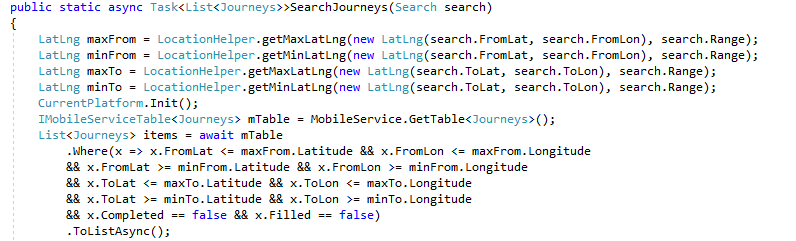
The method above was used to find first an increment in latitude (longitude remained constant) that equalled a distance of 1km. Then the same was done for longitude. This gave the following values used by the application.



This means one location (A) falls within the range (ex. 1km) of another (B) if:

* Latitude A >= Latitude B – (1\*0.008993215)
* Latitude A <= Latitude B + (1\*0.008993215)
* Longitude A >= Longitude B – (1\*0.014607411)
* Longitude A <= Longitude B + (1\*0.014607411)

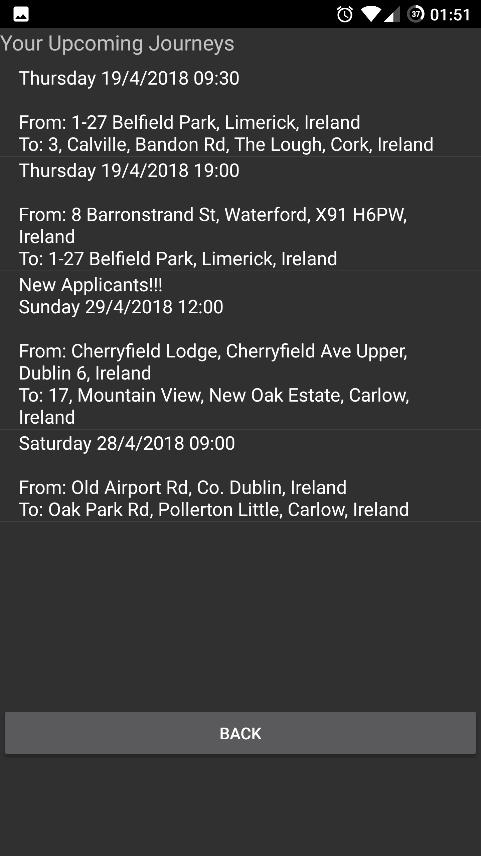
While this approach sounds complicated, it ultimately results in having a set of maximun and minimum figures that a prospective Journey has to fall within, and results can be queried directly to the App Service and database, which can compute the results in < 1 second, even for a large table of entries.



Upon filling in the details and hitting search, the app queries the App Service to find any matching journeys, which are returned to the user on a new screen to view. On this screen the user can apply for a position on the Journey, view the Journey details, or return the search screen. If they decide to apply for a Journey the app updates the Journey to include the user as an applicant, and the driver is sent an email notification that the Journey has new applicants. The user is returned to the main screen.

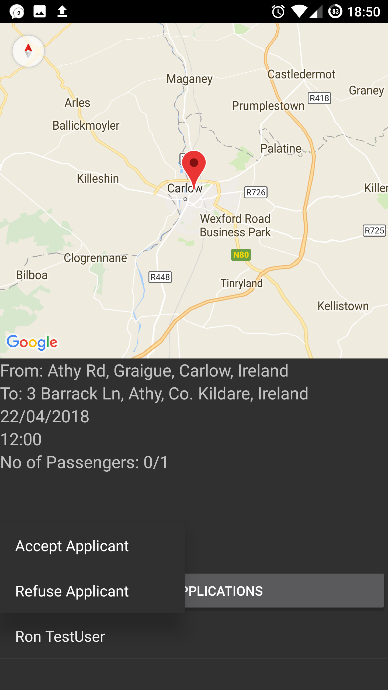
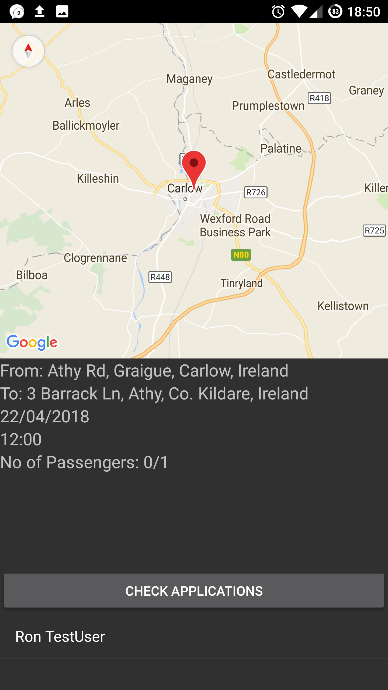
Emails are sent through a SendGrid account, which in this project was part of the Azure package and is registered to the same Azure account. A SendGrid API key is required, and allows the application to send various notifications to users, like in this case a notice to the driver to view applicants.

#### Viewing Upcoming Journeys - Driver



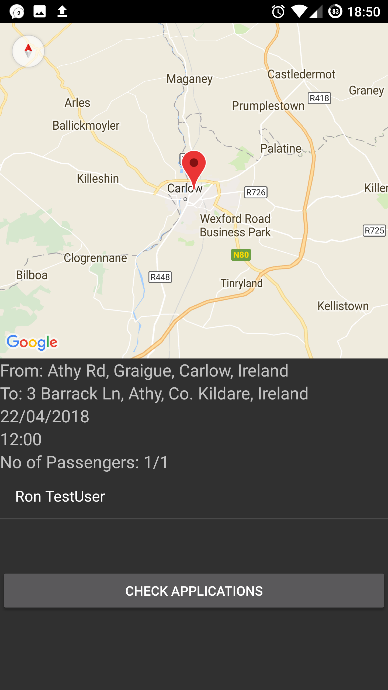
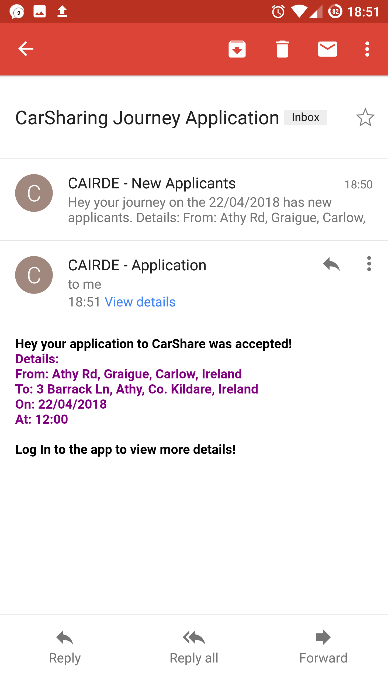
The View Upcoming Journeys button from the driver section of the main menu brings up this screen which is a list populated by journeys in which this user is a driver and which have not yet been completed. In this menu a user can select one of the Journeys to view details about it, like viewing it on the map and checking for applicants. Journeys which have new applicants are show in this list by the “New Applicants!!!” message, making the process of managing these Journeys easier for the user. The accepting of an applicant will be detailed in the next section.

#### Accepting an Applicant

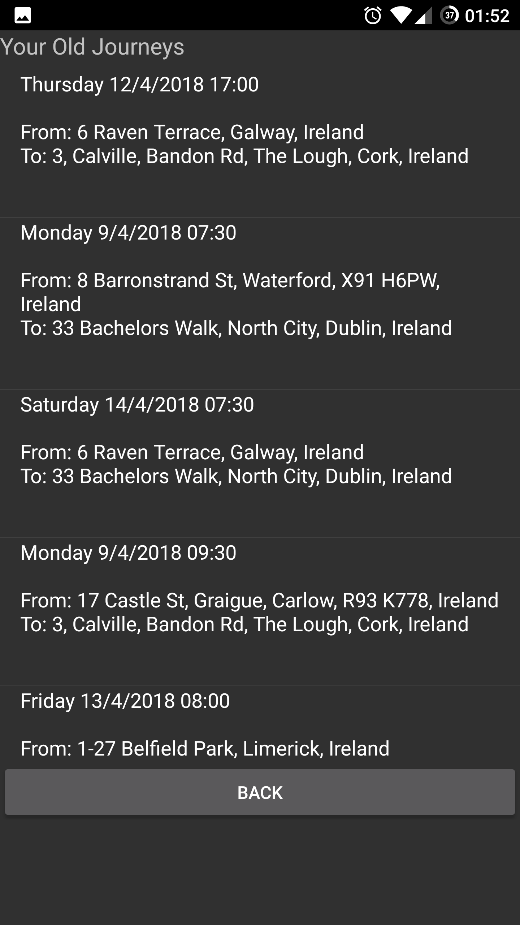


This function is reached by a driver selecting to view the details of an upcoming Journey as detailed in the previous section. This opens the view Journey screen, and by clicking the Check Applicants button the user can see the list of people who have applied for this Journey. The app does this by taking the comma delimited field of Applicants from the Journey object and splitting it into a list of User Id’s. The app then uses the App Service to look up the User Profiles of the applicants, and uses the returned data to populate the list of applicants with the applicants names.

Once an applicant is selected the user can decide to accept or refuse the applicant. Refusing the applicant removes their ID from the Applicants field and updates the record through Azure. Accepting the applicant adds the applicant to the Journey (provided there is a space remaining) and removes the applicant from the applicants field, adding them to passengers instead. The Journey record is then updated through Azure. An email notification is sent to the successful applicant to inform them the confirmation.

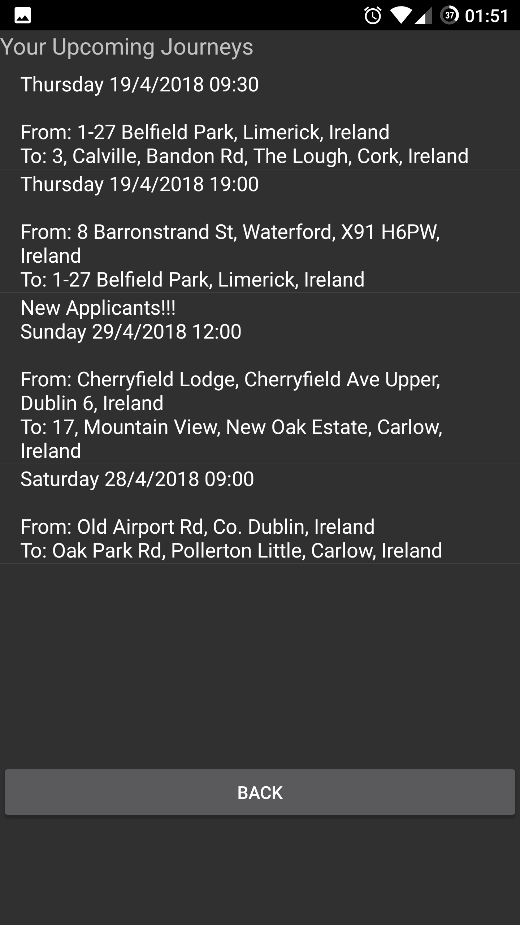


#### Viewing Old Journeys - Driver



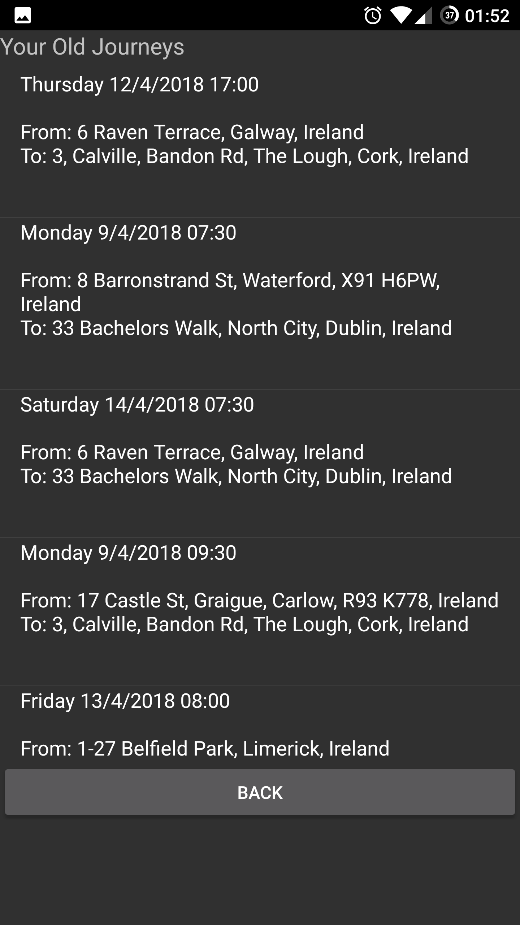
The View Old Journeys button from the driver section of the main menu brings up this screen which is a list populated by journeys in which this user is a driver and which have been completed. In this menu a user can select one of the Journeys to view details about it, like viewing it on the map.

#### Viewing Upcoming Journeys - Passenger



The View upcoming Journeys button from the passenger section of the main menu brings up this screen which is a list populated by journeys in which this user is a passenger and which have been not yet been completed. In this menu a user can select one of the Journeys to view details about it, like viewing it on the map.

#### Viewing Old Journeys - Passenger



The View Old Journeys button from the passenger section of the main menu brings up this screen which is a list populated by journeys in which this user is a passenger and which have been completed. In this menu a user can select one of the Journeys to view details about it, like viewing it on the map.

### Back End: Azure App Services & SQL Server

#### Overview

Microsoft Azure is Microsoft’s cloud computing service that provides a wide array of services and tools. It provides over 600 services including web hosting, database hosting, cloud storage, virtual machines, serverless computing and many more. They offer a range of pricing models for each service, with many services having a basic free tier. Azure provided a means to supply all the required backend functionality for the application in one place, almost for free, with the database the only service requiring an monthly fee of just over €4 to use. The database is a SQL database hosted in the cloud and always available. Creating a database is as easy as clicking a button and filling in some details thanks to Azure’s GUI Dashboard. In addition to the database Azure also provided a web app service which integrates with the application and database providing the link between the two. Once properly configured and running the web app made communicating with the database from the app safe, secure and easy. By posting C# objects to the service, the web app then translated the call into SQL for the database, reducing the programming load on the developer.

#### App Service

This service provided by Azure replaces the typical backend you might usually see in this type of project like a php server, instead integrating on top of the SQL database and being fully compatible with Xamarin for Android. By installing the required packages into the project (through nuget) the author has access to a library of methods from which they can access the database. Commands are sent from the mobile app to the App Service, which translates them into an SQL command and runs it on the database, then return the results as C# objects. It provides a high level of safety as it is not vulnerable to SQL injection or other malicious attempts to gain unauthorised access to the database. Access is granted through explicit methods such as InsertAsync(Object c), and no matter the contents of this operation only a valid insertion can take place, the App Service ensures this. The App Service also takes care of any filtering or searching through the results reducing the cpu time required by the mobile app.

#### SQL Database

This is a hosted database on Azure’s cloud platform and is the only paid service in the project. The SQL database holds all the User, User Profile, and Journey records created by the app. Connection to the SQL database is blocked for all incoming connections, except for traffic from the App Service, giving another layer of protection to the users data. The SQL database is a standard SQL style database, however used in conjunction with the App Service it behaves much like a NoSQL database like MongoDB, with tables not needing their schema to be defined, and the schema being changeable depending on the commands. This provides a level of abstraction from the coding to the data handling, allowing more time to be spent developing other functions.

# Conformance to Specification and Design

## Functional Specification Conformance

The system conforms to the functionality that was outlined by the author in the functional specification. All the use cases outlined in the functional specification were implemented, with no major changes. Each of the non-functional requirements were also implemented.

## 

## Design Document Conformance

The project conformed to the Design Document in most cases. The system architecture was implemented to the design spec and the SQL database was also implemented with no changes to its structure or design. The UI received some moderate updates, but this was to be expected given the prototype nature of the proposed UI in the design document. These changes will be shown in the following section.

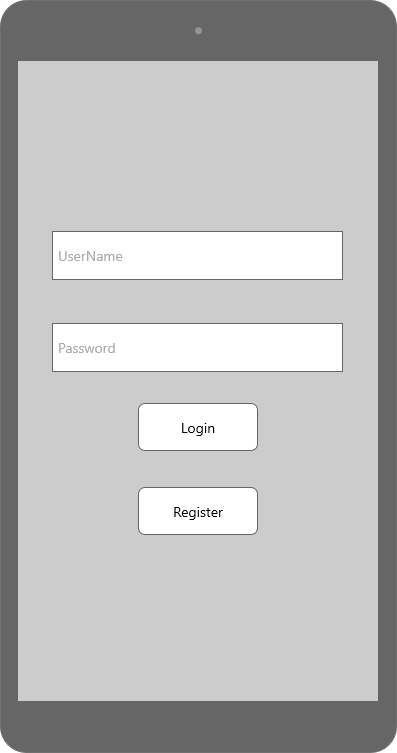
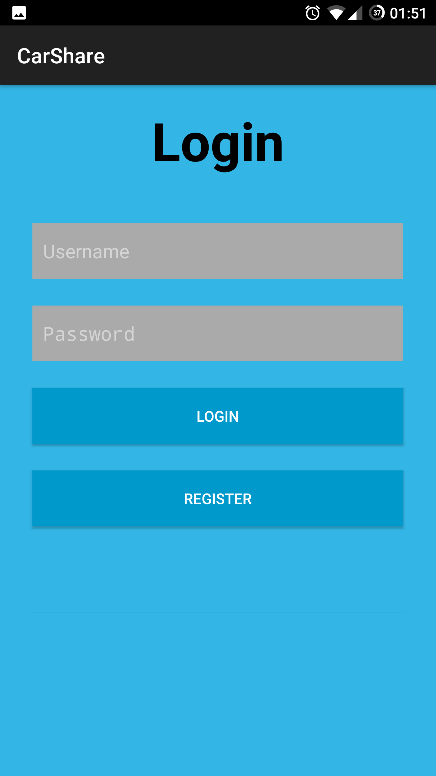
## Design Updates

### Overall Design

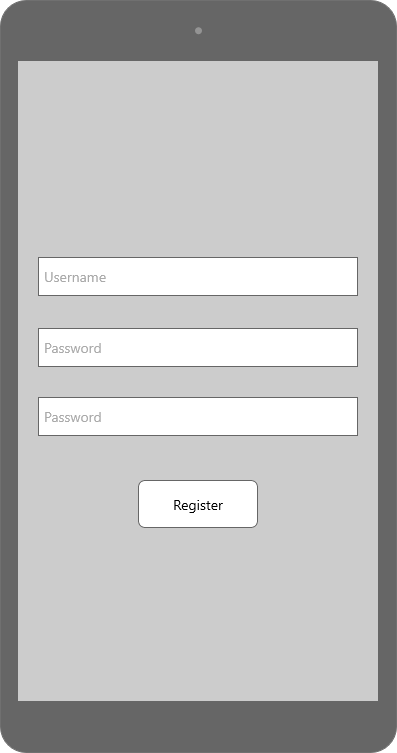
The overall user interface design that was implemented followed the guidelines set out in the design document, while making small changes as required either for a visual or functional reason. The following sections will display the prototype version and final version of the UI for each screen.

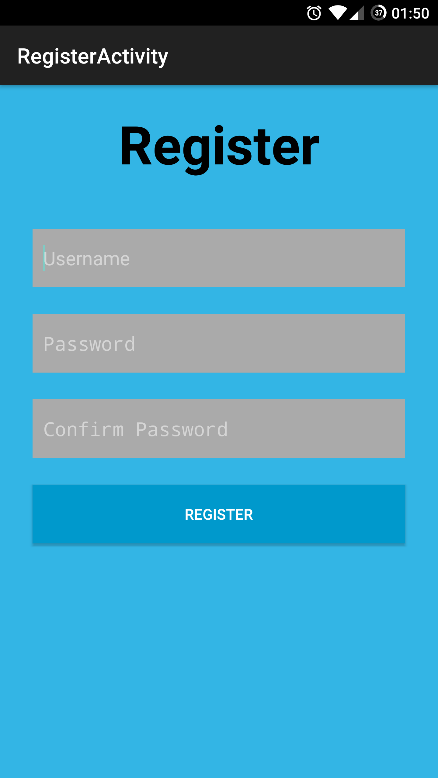
Comparison is only provided for what was documented in the design document.

#### Login

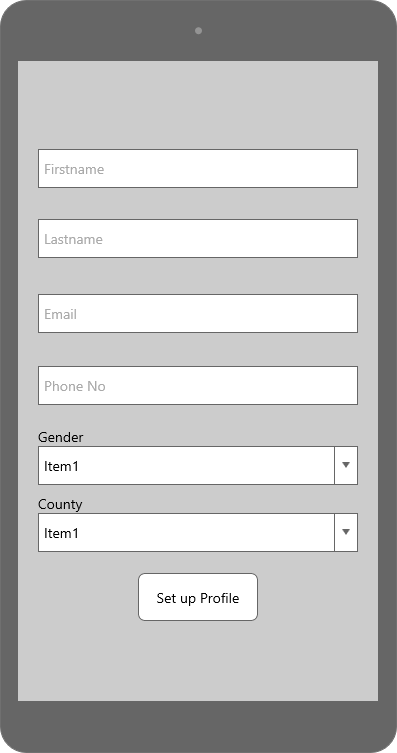
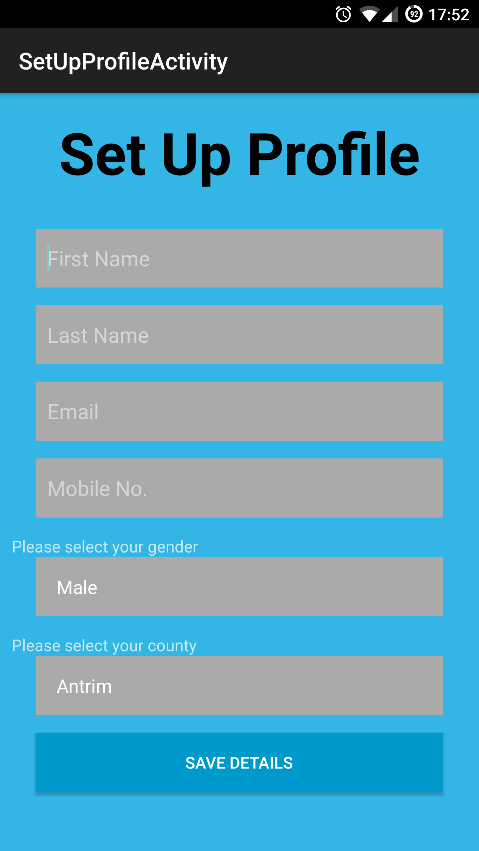


#### Register

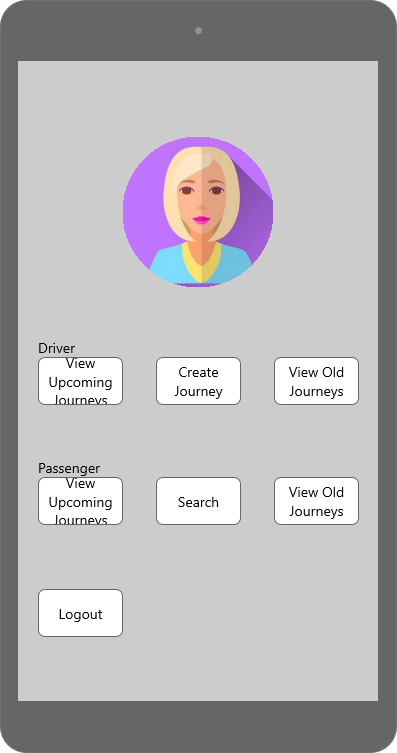
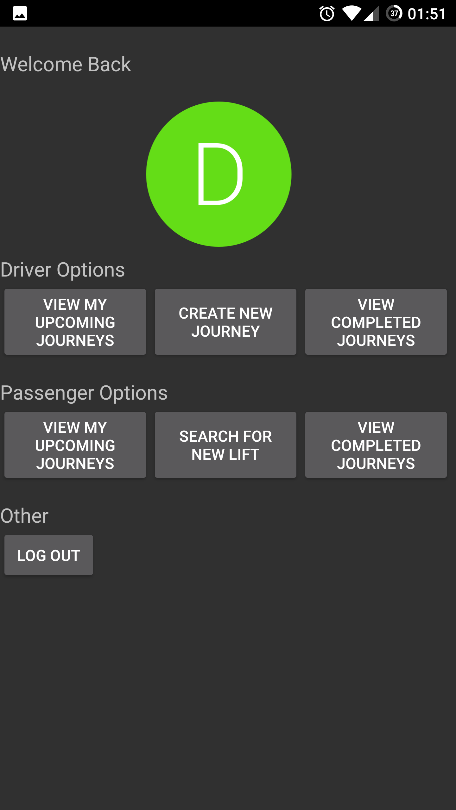




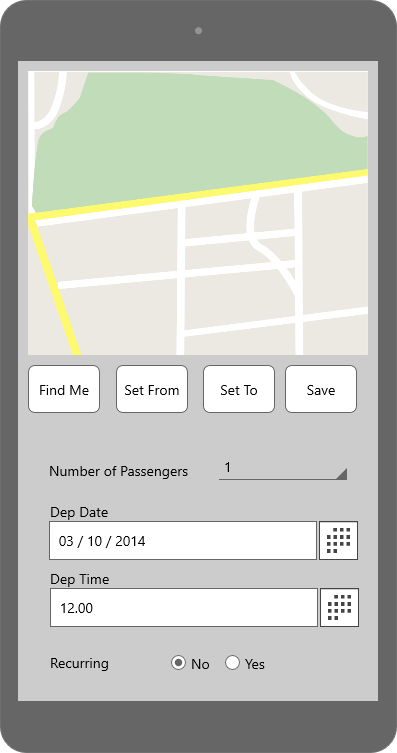
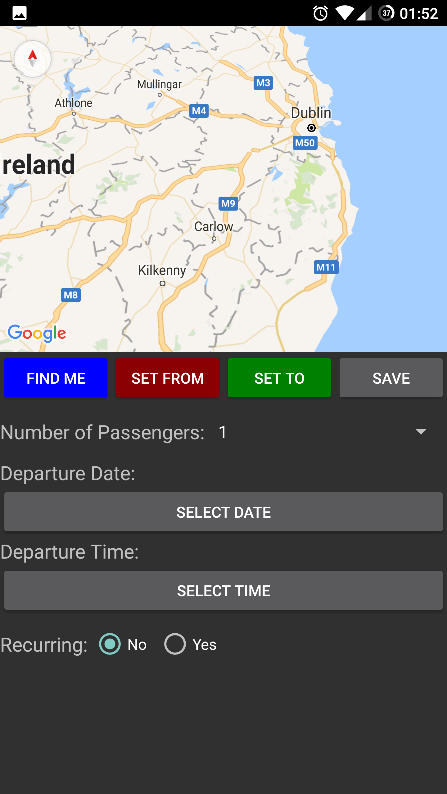
#### Set Up Profile



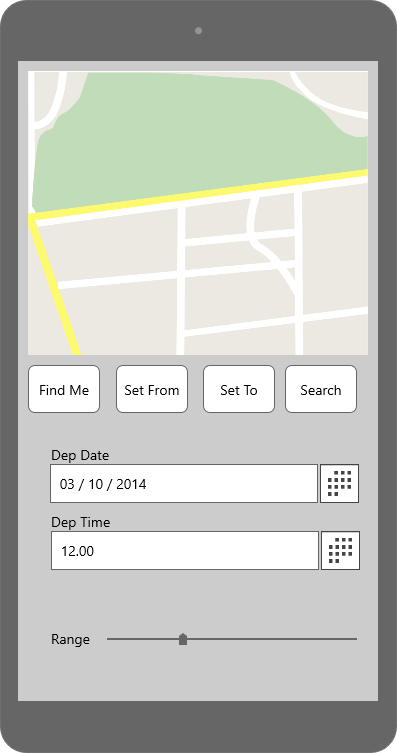
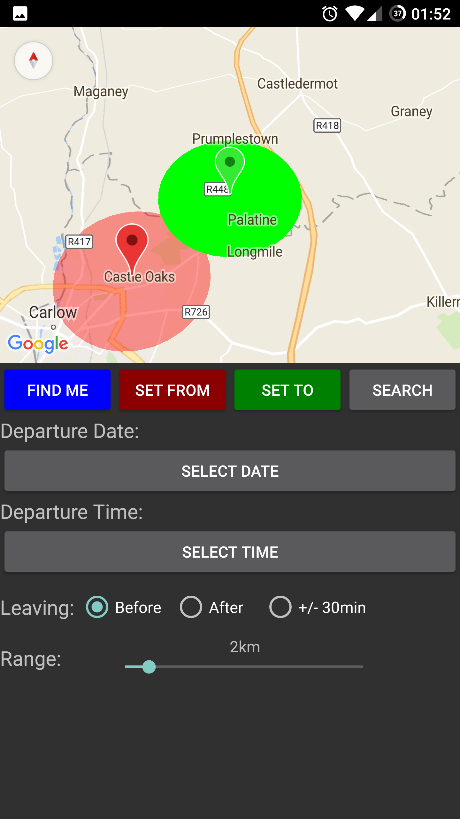
#### Main Menu



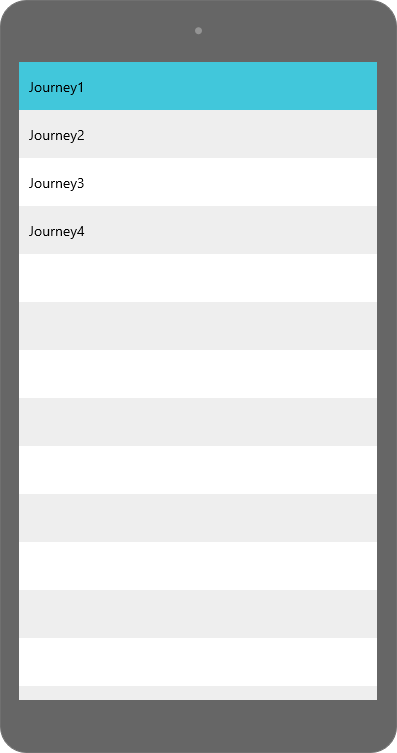
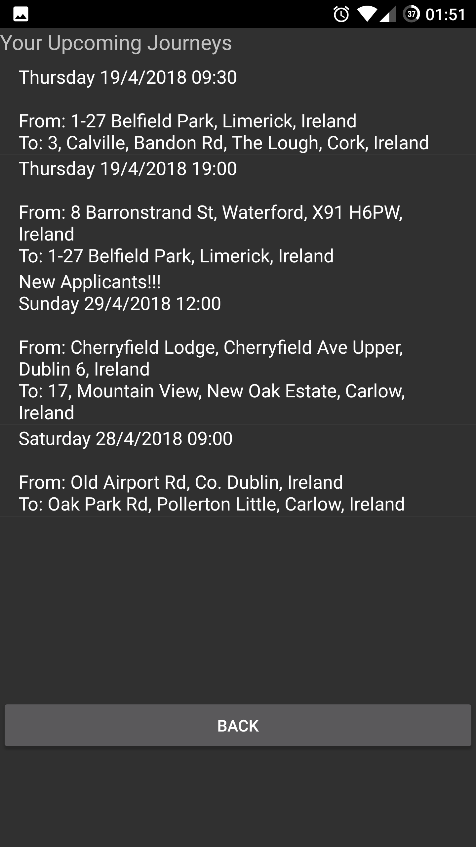
#### Create a Journey



#### Search Journeys



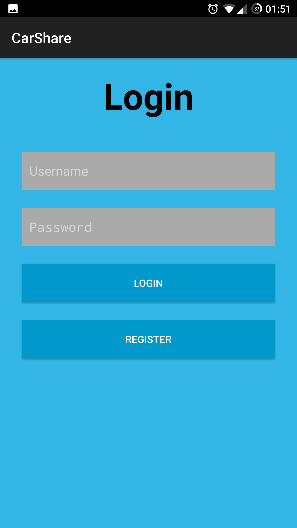
#### View Journeys



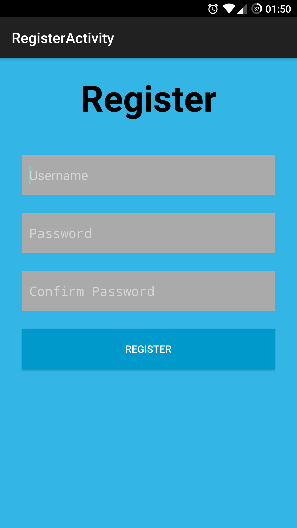
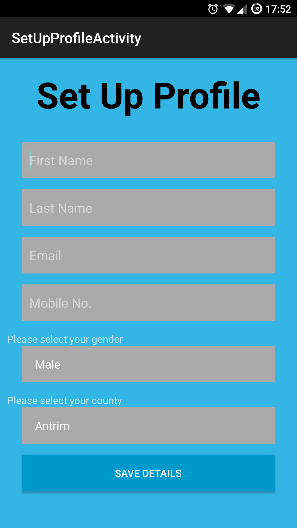
### 

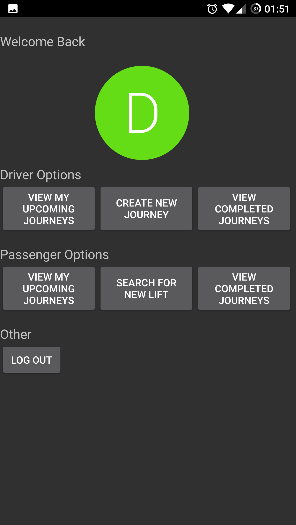
### UI Flow

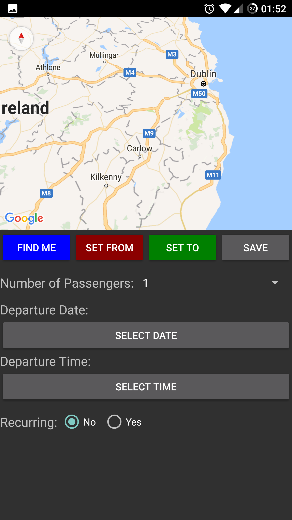
The UI flow again conforms with the originally specified flow in the design document. The flow is shown here once again for clarification with the update UI.

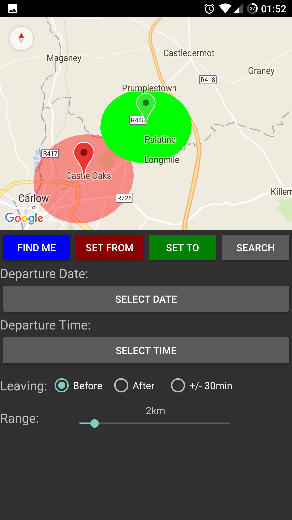


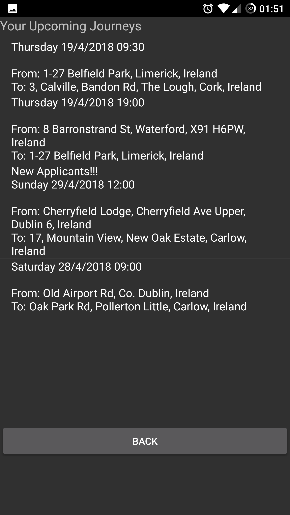
First Time











# Description of Learning

## Technical

Throughout the course of this project the author used various technologies, some of which the author was familiar with and others that required research and practice to get to a comfortable working level.

While the author had previous experience coding in C# using the .NET framework, he had zero experience in using Xamarin to build an Android application, and only one previous experience in building an Android app at all, which was in Java. This lead to an initial barrier to development of the app while the author learned some of the techniques and practises for using the new technology. After this period of learning the author grew to a more comfortable level in handling Xamarin, and could then begin to branch out his C# skills, developing a large project with multiple classes for the first time from scratch. Becoming familiar with the UI design mechanics which uses xml, was a particular challenge, and the author was improving this skill right up to the end of the project, as it is one area that is poorly documented by Xamarin. The xml side of the design was not an issue, but learning the various tags and attributes required and available was.

Working with Azure was another technology the author had some experience in, although not in the areas covered in this project. In this way it was a new challenge. Researching and studying the online documentation was key to developing the system architecture, and while it took a few attempts to get everything set up correctly, the experience gained in troubleshooting various problems was incredibly valuable. These are definitely skills the author will need again.

While the author was comfortable with SQL, using it in the way he did in this project was a different experience and brought with it its own challenges. By having the App Service communicate with the SQL Server the author had no way to structure SQL queries like he usually would, instead needing to use the tools available inside Xamarin and Azure to query the App Service, which took some adapting.

The maps integration with Google’s API’s was also new to the author and this presented a significant problem early in the project where it took a long time to become familiar with the various tools and requirements, as well as the limitations and constraints of the API. Ultimately the author came to a level of familiarity with the API, and could take better control of the maps integration in the project.

Finally using the SendGrid platform for managing the email notifications side of the application was entirely new, but fortunately it did not take long to learn as the documentation and libraries built are very easy to use. Again it was more of an issue configuring the service initially, than implementing it.

## Personal

Over the months spent developing the project the author gained many new personal skills, as well as strengthening existing ones. This level of personal development was perhaps the single biggest increase in the course of the authors four years studying the course, at the very least equal in experience to the internship undertaken the previous summer.

One skill that the author developed was the ability to design and plan a large scale project on their own. While the author had worked on large projects before, this was the first time they were tasked with building everything from the start, including picking all the tools. This was a new experience compared to the authors previous efforts which were always well defined and structured, now there were hundreds of choices that could be made and initially it was daunting. However the author quickly learned that with thorough research and a clear focus they could see through all the choices to the best option for their project, a valuable skill to have.

A skill that was improved upon was the ability to multi-task. Never before had the author such a large workload, with many projects in different subjects running concurrently, in conjunction with a professional career as a software engineer and personal life, often it seemed impossible to handle everything. The key to handling this came in the authors newly found organisational skill, previously hidden from him. The purchase of a large whiteboard which was used to keep track of tasks, dues dates, brainstorm and take notes meant the author could keep a visual overview of everything and plan accordingly. Unfortunately even the appearance of an organisational skill and improved time-keeping skill could not fully off-set the volume of work that was required, and in this regard the project suffered some times from a lack of development while other duties were accommodated.

Of course working under such time constraints did improve another of the authors skills, the ability to work under pressure. Always previously a strength of the author, this was never more apparent than the crunch before deadline days, diligently working without panic to finish one feature or another.

# Review of Project

## What went right

In general, the author feels like the project was a success in the regard that the goals set out in the functional specification were met. The application is in a fully functioning state and the backend fully hosted and set up.

The author is very pleased with his implementation of the car sharing app idea, particularly with how easy the app is to use and how effectively the matching function works by using a range finder. This the author feels is a unique feature and makes the entire experience much better. Furthermore the UI is well laid out and the user can navigate the app quickly and easily in very few button clicks.

Using Azure for the backend was another success, having a powerful hosted backend that can scale up and down as required means the user experience on the app is never degraded. The inclusion of SendGrid for email notifications gives the app a very nice professional feel.

## What went wrong

While the overall project was a success, it was not without its failures. During the course of development the author had several features he would liked to have added, but could not due to time constraints. These features would have really added to the app in a meaningful way and it is regretful the author could not showcase his new skills more in this regard.

In addition to the above the author is disappointed that the UI could not have been improved more. While it is clean and functional and user friendly, the author would have like the time to polish it a bit more and try out some different colour themes. Again time and not ability was the limiting factor here.

Another area that wasn’t a failure as much as a disappointment was notifications. During development the author attempted to implement push notifications into the app, going so far as to create the required classes, creating a Google FireBase account and an Azure Notification Hub, but despite his best efforts the author could not get notifications to come through. A full day was lost to this issue before the idea was scraped and email notifications only used. The author is sure that with more time he could have got this working.

## Outstanding/Missing Work

All the core functionality of the app is present, and while there is a large list of additional features planned for future versions, the author does not count this as missing work.

## Technology Changes

At the start of research phase of this project the author initially developed the app in both Java with Android Studio and HTML with Ionic. Both of these were abandoned at an early stage, Ionic due to a dislike of HTML and Android Studio due to a familiarity of Visual Studio. Other than this there were no other technology changes.

## What the author would do differently

If the author was to do the project again, the sole change he would make is to have better time-management, that is to say he would dedicate more time to the project which would allow for more features and improvements to the things that went wrong highlighted above.

## Future Features

Touched upon in a previous section is the list of additional features planned for future releases. These were always in the developers mind to add but time did not allow for it.

* A brief messaging service allowing people on the same journey to send messages in the app to each other
* A rating system for users where passengers can rate drivers and vice-versa, which would help to blacklist problem users and promote a safe environment for users
* A save search feature. Currently searches with no matching journey are not saved, this feature would save them and if a driver creates a journey that matches a saved search the search creator is added as an applicant on the Journey automatically.
* A tutorial for first-time users. This would involved having popups on the screen explaining how to use each page, which could be toggled off.
* A preference/options page, where users could manage their accounts, including adding profile pictures.
* A toggle to select whether your email or phone number can be shared with passengers. Not currently shared, this would allow users to choose their method of communication depending on their requirements.

# Acknowledgements

The author would like to sincerely thank his supervisor Lei Shi for his help, support and advise. He would also like to say a huge thank you to his girlfriend and family without whose support he would surely have failed. Lastly, he would like to thank his work colleagues and college friends, who were always available to bounce idea’s off and brainstorm solutions. I only hope I helped them as much as they did me.

# References

GitHub. (2018). defuse/password-hashing. [online] Available at: https://github.com/defuse/password-hashing/blob/master/PasswordStorage.cs [Accessed 17 Apr. 2018].