By: Kevin Dawson

Supervisor: Joseph Kehoe



Final Report

Abstract

The purpose of this document is to provide an overview of the project including functionality that is present in the finished product. It also includes sections that provide the authors reflection on their overall project experience.

[1. Introduction 5](#_Toc479085594)

[2. Project Description 6](#_Toc479085595)

[2.1 What is Looprac? 6](#_Toc479085596)

[2.2 Features and Functionality 6](#_Toc479085597)

[2.2.1 Front End: Mobile Application 6](#_Toc479085598)

[2.2.1.1 Overview 6](#_Toc479085599)

[2.2.1.2 Register 7](#_Toc479085600)

[2.2.1.3 Login 8](#_Toc479085601)

[2.2.1.4 Home Page 8](#_Toc479085602)

[2.2.1.5 Offering a Lift 10](#_Toc479085603)

[2.2.1.6 Search for a Lift 11](#_Toc479085604)

[2.2.1.7 Requesting Seat on Lift 12](#_Toc479085605)

[2.2.1.8 Viewing My Groups 13](#_Toc479085606)

[2.2.1.9 Viewing My Lifts 14](#_Toc479085607)

[2.2.1.10 Deleting a Lift 16](#_Toc479085608)

[2.2.1.11 Starting a Lift - Driver 17](#_Toc479085609)

[2.2.1.12 Starting a Lift – Passenger 19](#_Toc479085610)

[2.2.1.13 Viewing Completed Lifts 20](#_Toc479085611)

[2.2.1.14 Viewing Completed Groups 21](#_Toc479085612)

[2.2.1.15 Viewing User Profile 22](#_Toc479085613)

[2.2.1.16 Viewing Leader Board 23](#_Toc479085614)

[2.2.2 Back End: Looprac API 23](#_Toc479085615)

[2.2.2.1 Overview 23](#_Toc479085616)

[2.2.2.2 Password Hashing 23](#_Toc479085617)

[2.2.2.3 User Overall Rating Algorithm 24](#_Toc479085618)

[2.2.2.4 DBcm – Data Base Context Manager 24](#_Toc479085619)

[3. Conformance to Specification and Design 25](#_Toc479085620)

[3.1 Functional Specification Conformance 25](#_Toc479085621)

[3.1.1 Functional Updates 25](#_Toc479085622)

[3.1.1.1 Delete a Lift 25](#_Toc479085623)

[3.1.1.1.1. Detailed Use Case 25](#_Toc479085624)

[3.1.1.1.2 System Sequence Diagram 26](#_Toc479085625)

[3.1.1.2 Begin Lift Instance -Driver 26](#_Toc479085626)

[3.1.1.2.1 Detailed Use Case 26](#_Toc479085627)

[3.1.1.2.2 System Sequence Diagram 28](#_Toc479085628)

[3.1.1.3 Begin Lift – Passenger 28](#_Toc479085629)

[3.1.1.3.1 Detailed Use Case 28](#_Toc479085630)

[3.1.1.3.2 System Sequence Diagram 29](#_Toc479085631)

[3.1.2 Updated Main Use Case 30](#_Toc479085632)

[3.2 Design Document Conformance 30](#_Toc479085633)

[3.2.1 Design Updates 31](#_Toc479085634)

[3.2.1.1 Overall Design 31](#_Toc479085635)

[3.2.1.1.1 Login 31](#_Toc479085636)

[3.2.1.1.2 Register 32](#_Toc479085637)

[3.2.1.1.3 Car Register 32](#_Toc479085638)

[3.2.1.1.4 Main Page 33](#_Toc479085639)

[3.2.1.1.5 Search for a Lift 33](#_Toc479085640)

[3.2.1.1.6 Request a Lift 34](#_Toc479085641)

[3.2.1.1.7 Profile 35](#_Toc479085642)

[3.2.1.1.8 Offer a Lift 35](#_Toc479085643)

[3.2.1.2 UI Flow 36](#_Toc479085644)

[4. Description of Learning 38](#_Toc479085645)

[4.1 Technical 38](#_Toc479085646)

[4.2 Personal 38](#_Toc479085647)

[5. Review of Project 39](#_Toc479085648)

[5.1 What went right 39](#_Toc479085649)

[5.2 What went wrong 39](#_Toc479085650)

[5.3 Outstanding/Missing Work 40](#_Toc479085651)

[5.4 Technology Changes 40](#_Toc479085652)

[5.5 What the author would do differently 40](#_Toc479085653)

[5.6 Future Features 41](#_Toc479085654)

[6. Acknowledgements 41](#_Toc479085655)

[7. References 42](#_Toc479085656)

# Introduction

The purpose of this paper is to document the result of the authors final year project Looprac. This paper is split into several headings which will include a description of the submitted project, looking at the functionality that is present and the authors reflection on their experience throughout the project time.

In the project description section, it will look at a detailed description of Looprac and its functionality, this will include screenshots of pages from a mobile device. It will also talk about API’s that were used to help create some of the functionality which were unknown to the author at research phase.

The conformance to specification and design section will look at comparisons between the functional specification, design document and the finished project. This section will discuss difference outlined in the functional specification and any differences between the design document and the implemented design.

This will be followed by description of learning for the author which will reflect on two areas, what technical skills did the author gain from this project and what personal skills were gained.

Finally, the author will review the project. In this section, it will describe what the author felt went well, what didn’t go so well, their opinion on the technology choices that they made, and if they were to start again what would they do differently.

# Project Description

## What is Looprac?

Looprac is a car sharing or car pool application. The purpose of this application is to allow people to travel together to mutual destinations. This had aimed at the benefits of decreasing the number of cars on the road, its environmentally friendly and. The application provides two routes to registered users, to be a driver or a passenger.

The driver route allows the user to advertise a lift they are going to from one point to another on a particular date. They are able then to accept or deny requests from other users to join them on this trip.

The passenger route allows people that don’t drive, or just don’t want to drive to look for lifts in their area or further. It allows them to send requests to drivers asking permission to join them on trips.

The idea of this application is focused on colleges and organisations where people can group together to travel to and from college or work.

## 2.2 Features and Functionality

The Looprac system is hosted on a Flask server on python anywhere. This API bridges the gap between the application and the MySQL database. This section will firstly look at the mobile application side and then will look at the Looprac API system.

### Front End: Mobile Application

#### Overview

The application was created with Adobe PhoneGap which allowed it to be created using HTML, CSS and JavaScript. This gives it the advantage of being portable to different platforms with just a few lines of code. This project was developed and tested on Android. With it being a car sharing application, the use of maps was a huge part of it. For this Google Maps JavaScript API was used, alongside other Google API’s to compliment it including geocoding, geolocation, directions and distance matrix. This is done by getting an API key from Google by creating a project and requesting an API key to associate with that project. From there the author could choose from a list of API’s to enable for that API key.

#### Register

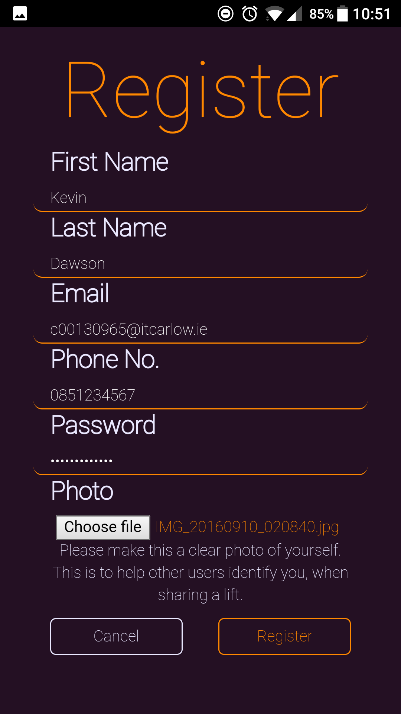
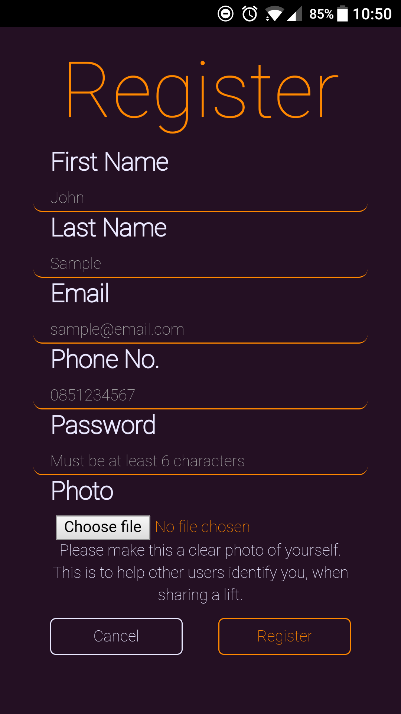


Figure 1

The registration page of the application is as shown in figure 1. It requests the users name, email, phone number for if they want to contact each other outside the application, a password which is as specified in figure 1 and the requirement of uploading an image of themselves for other users being able to identify the user. This also acts as a form of further verification of the user. The form contains relevant pattern matching to control the data which is entered. Then using JQuery and AJAX the data is sent to the Python backend which communicates the request and gets the data before inserting it into the database. At the backend, the Looprac API, the image is saved within a directory on Flask, and the path is inserted into the database. The password is hashed before being inserted into the database. This will be looked at in more detail in the Looprac API section.

#### Login

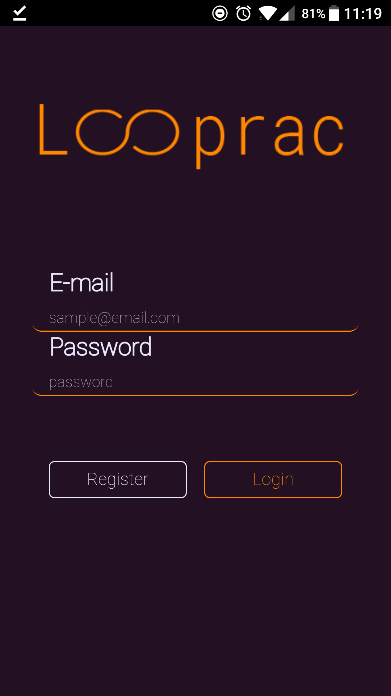


Figure 2

The login screen is the landing page when the user starts up the application, from here they have the choice to register to Looprac, or having previously registered can login with their credentials. The credentials that are entered are sent to the backend using JQuery and AJAX and processed. This will be explained in more detail in the Looprac API section.

#### Home Page

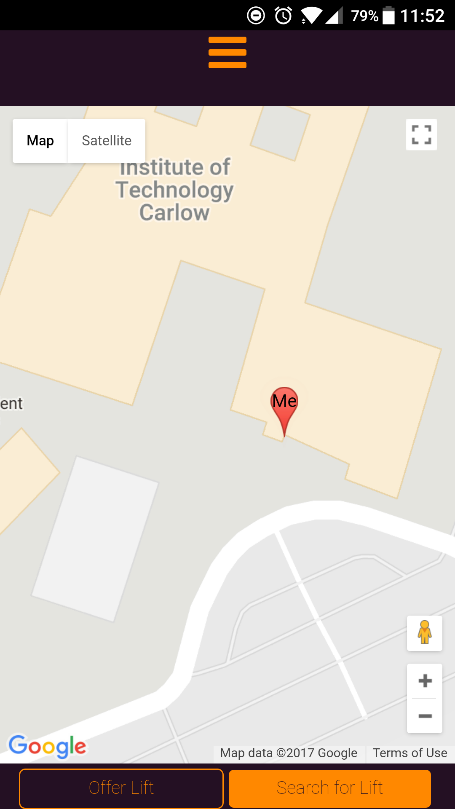
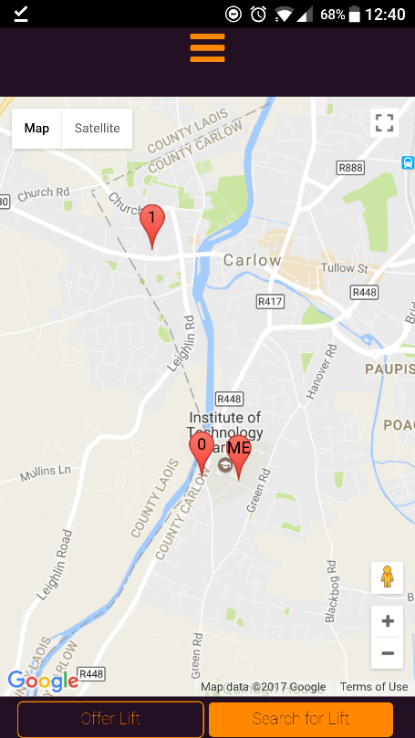


Figure 3 & 4

The home page is where the user is brought to when they login. Using the phones GPS and Google Maps JavaScript API along with Googles Geolocation API the application displays the user’s location on a map, as shown in figure 3. From this page the user can select to either offer or search for a lift. This page also displays to the logged in user any start locations of lifts that are available to them, this again uses the Geolocation API to display these. The user can select one of the markers and it will bring them to the lifts details page.

The functionality on this page is with the use of Googles Maps JavaScript API which displays the map seen in figures 3 and 4. This is done by referencing googleapis.com with the API key that is assigned to the project created by the author, as shown below:

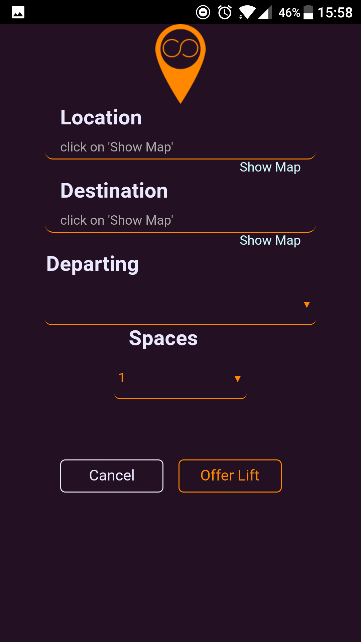
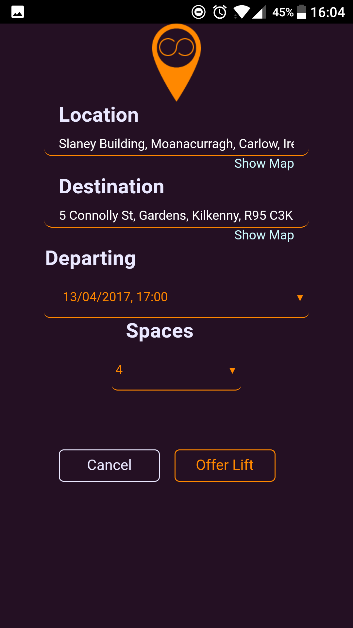
<script src="https://maps.googleapis.com/maps/api/js?key=***YOUR\_API\_KEY***" async defer></script>

The async attribute in the tag is a Boolean attribute, it means that the script will be executed asynchronously as soon as it is available. The defer attribute is also a Boolean attribute which specifies that the script is executed when the page has finished parsing. With async being present with defer it means that the script is executed asynchronously while the page continues the parsing.

The Geolocation API uses the same script reference, and uses coordinate variables to create an object which can be used and displayed on the map.

The Maps JavaScript API where a map is created along with any options such as zoom or where to centre the map, is used in conjunction with the Geolocation API where the application gets the users latitude and longitude coordinates, then creates an object with the Geolocation API and inserts it into the specified map.

#### Offering a Lift



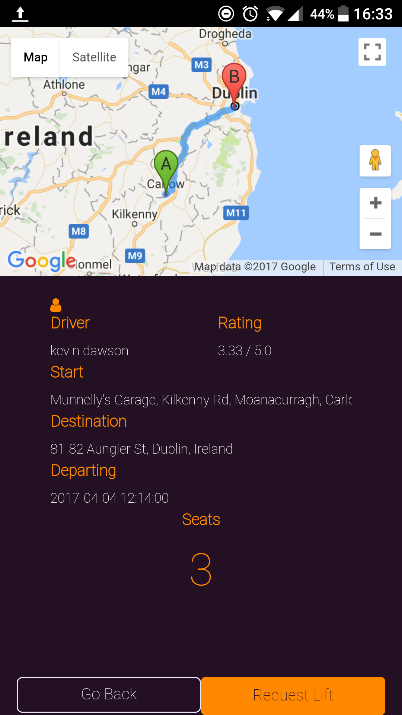
Figures 5, 6 & 7

The offer a lift feature is the main function for the drivers. It allows them to advertise a lift to every other user in the application allowing them to request a lift with the driver. As shown in figure 5 the author tried to keep the needed information to a minimum to not clutter up the page and to make the process of offering a lift quick and easy. For a driver to choose a start and end location they must click on the ‘Show Map’ button. This will present them with a map and a marker at their location. The driver can drag and drop the marker where ever they want and the information box and input field at the bottom will update with the address of the location that the marker is currently at. This is the process for picking a destination as well.

The functionality behind this feature involves the use of four Google API’s: JavaScript Maps, Geolocation, Geocoding and Distance Matrix. The JavaScript Maps is used to display the map’s; the Geolocation is used to display the user’s location on the map. Geocoding is the process of converting addresses into geographic coordinates which can be used to place markers on a map. However, for choosing a location for the start and destination the application is implementing reverse geocoding. This is, when the user drags the marker to a position on the map it gets the coordinates and returns from the API a human readable address. Then using JavaScript, the application displays the address in the input field at the bottom of the page and puts it in the relevant field on the offer lift form when the user clicks submit. The Distance Matrix API is used to calculate the distance between two location objects that are created by the geolocation API. It takes in options such as travel mode which is set to driving, the unit system which is set to metric and two arrays containing the start and destination location objects. It returns the distance between the points in kilometres.

Figure 7 is an example of a filled-out offer lift form. Then, like the register form, using JQuery and AJAX the data is sent to the Python backend which communicates the request and gets the data before inserting it into the database.

#### Search for a Lift



Figures 8 & 9

The features of this case are that it displays a list of all available lifts to the user, which if they are a driver, are not any of their own lifts and that they have not already requested a seat and lifts where there are available seats. The user is first presented with a list of available lifts as shown in figure 8. Upon selecting a lift from the list, the user is brought to the lifts detail page which is shown in figure 9. Here the route of the lift is displayed to the user on a map identifying the start and end locations. The user is also presented with details about the lift which includes details on the driver such as their name and rating. The icon above the driver title allows the user to view the drivers profile shown in figure 9, which will be looked at later in the document. The details presented on the lift include the start and end locations, when it departs and the number of available seats. The user can then request a seat in the driver’s car for the particular lift.

The functionality behind these screens are in figure 8 an AJAX request is sent to the backend with one of the user’s unique ID’s and is processed by the Looprac API which then returns the lifts that are available to the user. In figure 9, an AJAX request is sent to the backend the relevant lift ID and is processed by the Looprac API and returns data about the lift. This page implements the following Google API’s: JavaScript Maps to display the map, geocoding to present a readable address for the start and destination addresses and Directions API which is used to display the route between the two points.

The Directions API works by taking geolocation objects, which are included in the details retrieved from the backend, along with other options in a request such as travel mode and unit system and sends these to the API which returns a route which is then applied to the map.

#### Requesting Seat on Lift

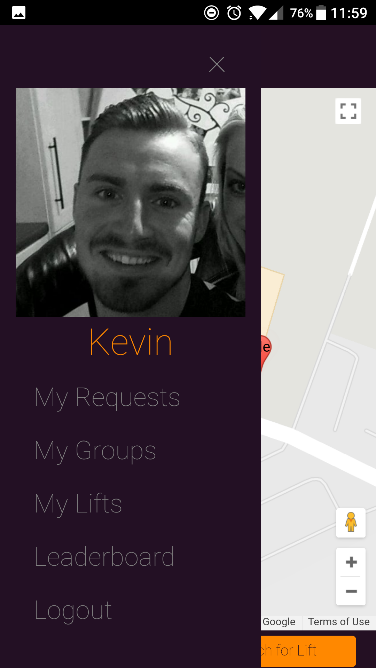
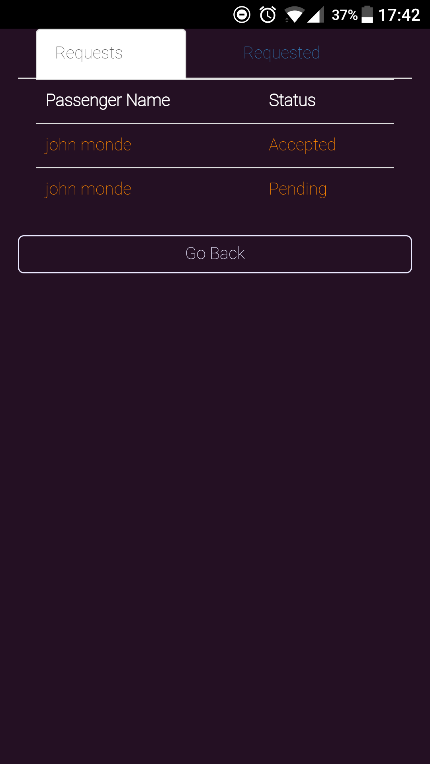
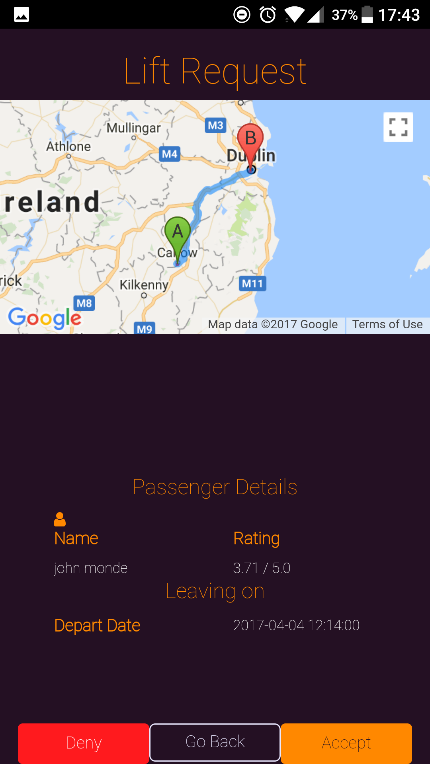


Figure 10 & 11

The feature of this part is that a user can request a spot in a driver’s car for a lift that they have advertised. Once the user has requested a lift they can keep an eye on the status of the request by clicking on the hamburger icon at the top of the landing page and then on ‘My Requests’ as shown in figure 10. This will present the user with a tabbed page as shown in figure 11 where they can view requests from people under the ‘Requests’ tab and their own requests under the ‘Requested’ tab. A request has three states: pending, accepted and denied.

For the driver, they can check requests in the same manner by selecting ‘My Request’s’ but viewing the ‘Requests’ tab as shown in figure 12.



Figures 11 & 12

When the driver selects a request from the list they are presented with details of the request, that is, a map displaying the lift route, some details on the user requesting the lift including their name and rating and the departure time for the lift, as shown in figure 12. They can also click on the profile icon which allows them to view the users profile. On accepting a request, both the driver and the passengers request status are update to reflect and they are added into a group for the lift. If the driver denies the request, again the request status of both is updated to reflect the decision.

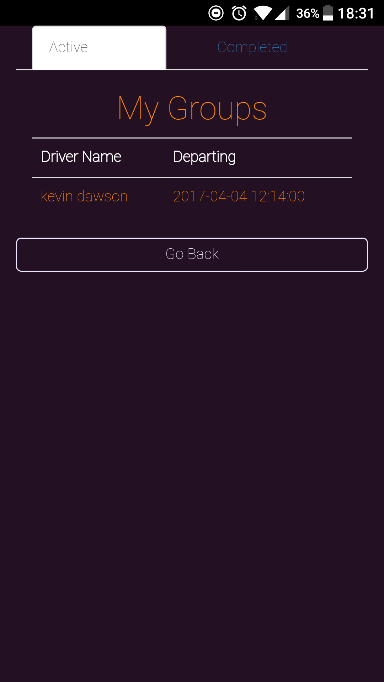
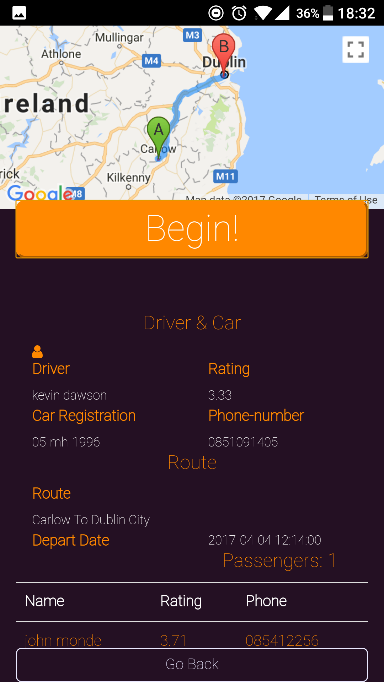
The functionality behind this for the request lists showing the status, an AJAX request is made to the Looprac API with the user’s unique ID. This is processed in the backend and returns a lift of all requests from the user and requests for the user.

With the request details page, shown in figure 12, is updated from an AJAX request to the backend which gets the relevant details associated with the request including lift details and the requesting user’s details. It also implements two Google API’s: JavaScript Map to display the map and Directions API to show the lift route on the map so that the driver knows which lift this is associated with.

When a request is accepted, an AJAX request is sent to the backend using the request ID and both user’s status in the lists are updated. The driver and passenger are put into a group which is created by the Looprac API, and any subsequent passengers are added to this group.

When a request is denied, an AJAX request is sent to the backend using the request ID and both user’s status in the lists are updated.

#### Viewing My Groups



Figures 13 & 14

The ‘My Groups’ option is available for passengers that have been accepted by a driver to join a lift. To get to this page a user must select ‘My Groups’ from the menu as shown in figure 10. They are presented with a tabbed page which allows them to view active and completed groups as shown in figure 13. When they select an active group, they are then presented with the group details page which includes a route of the lift, driver and car details and with themselves and the other passengers in the group. They can click on the users in the list to view their profiles as well as the profile icon above the ‘Driver’ label to view the driver’s profile. The begin button is used to activate the lift for the user when it is time for the lift to depart and they are at the start location.

The functionality behind these screens are for the list in figure 13, an AJAX request to the backend and it retrieves a list of groups that the user is a member of. In figure 14, there is an AJAX request made to the backend with the group ID which is processed by the Looprac API and returns all the relevant information for the group. This information is used to create a list of all passengers in the group as well as use lift details alongside two Google API’s: JavaScript Map and Directions API to display the lift and route for the user.

#### Viewing My Lifts

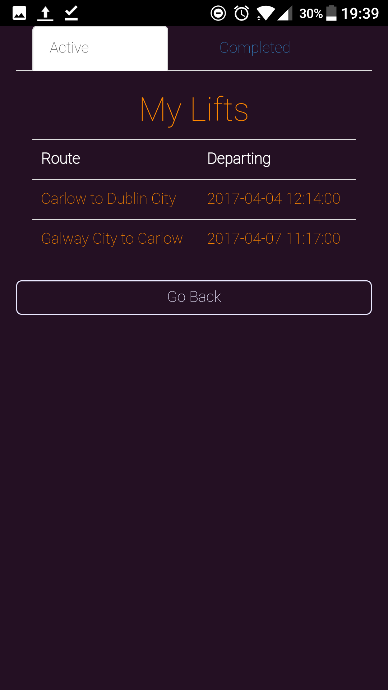
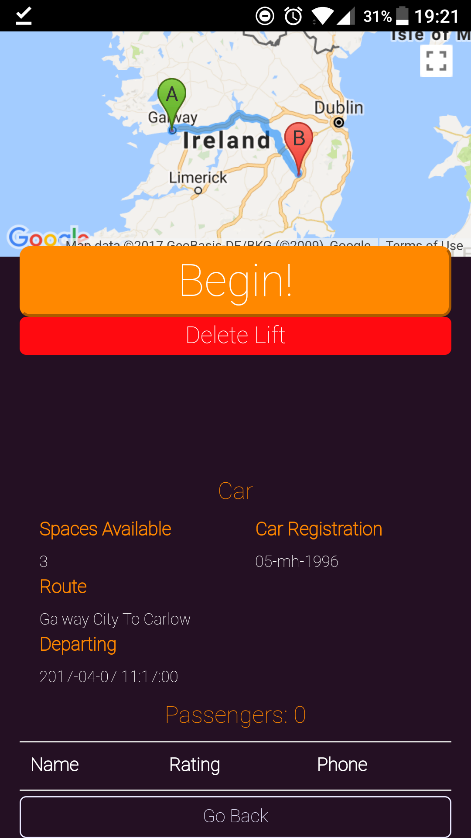
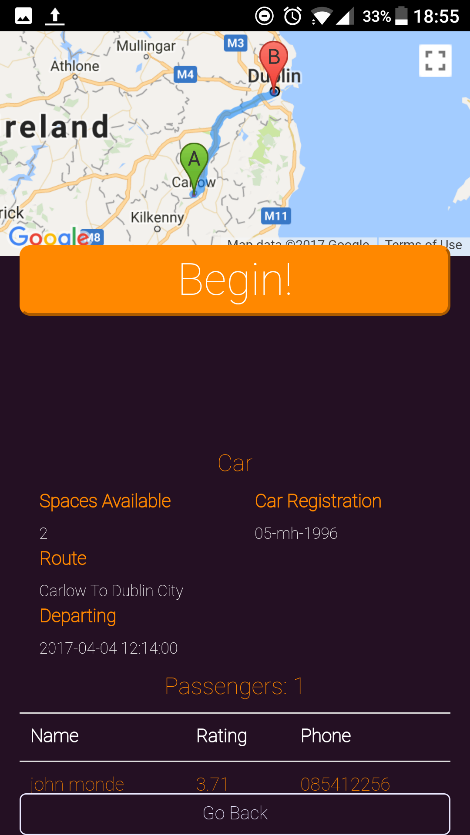


Figure 15

‘My Lifts’ allows drivers to keep track of their lifts. It is displayed by selecting ‘My Lifts’ from the menu that is shown in figure 10. It presents the user with a tabbed page which allows them to look at active and completed lifts, as shown in figure 15.



Figures 16 & 17

When a user select one of the lifts from the list, they are presented with a details page on the lift which will provide a map with the lift route, car details such as the registration number and the seats that are still available. It also has a list with the members whose requests have been accepted, as shown in figure 17. The driver can select a person from the list and it will show them their profile.

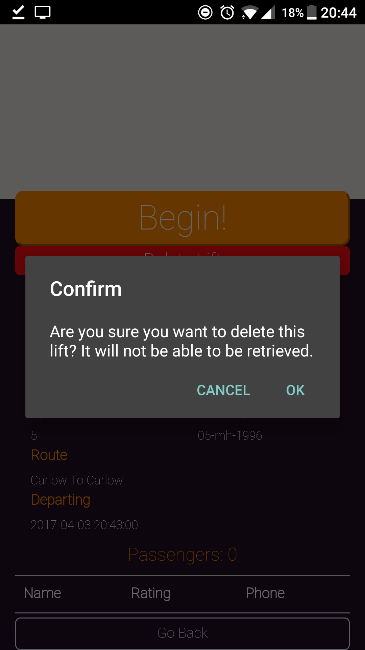
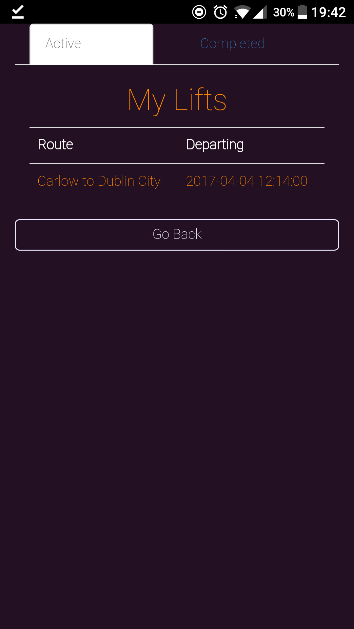
If the lift has no passengers in it yet, the driver is presented with the option to delete the lift. This option is only provided to lifts which don’t have passengers. An example is shown in figure 16.

The functionality here for figure 15 is the application sends an AJAX request to the Looprac API with the user’s ID and it sends back a list of all the user’s lifts.

In figures 16 and 17, an AJAX request is sent with the lift ID to the Looprac API and it sends back information about the lift. The lift details screen implements two Google API’s: JavaScript Maps and Directions API to display the lift route to the driver.

If the lift has no passengers assigned to it the attributes to the delete button are changed to display it as shown in figure 16, which gives the user the option to delete. If there are passengers assigned to the lift, the delete button remains hidden and the user is presented with a list of the passengers which is shown in figure 17.

#### Deleting a Lift

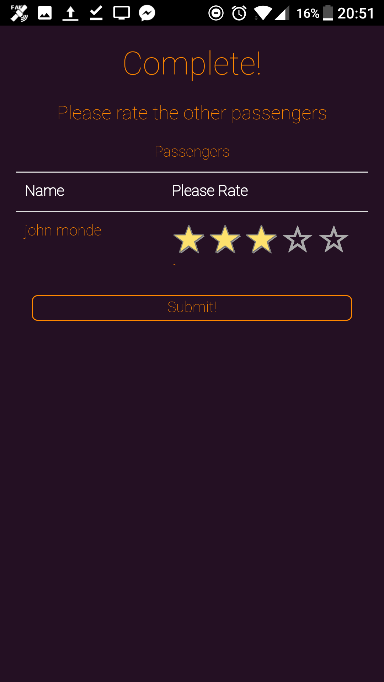
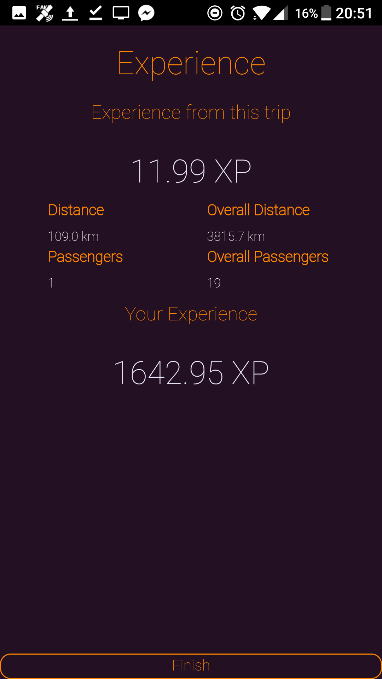


Figures 18 & 19

The feature provided here is to allow a user to delete a lift from Looprac, on the condition that it has no passengers assigned to it. When the user selects the delete button they are presented with a pop up to confirm their decision as the command is destructive and the lift won’t be retrieved, as shown in figure 18. They are then returned to the updated list as shown in figure 19.

The functionality here is an AJAX request is sent to the Looprac API and the lift is deleted from the database.

#### Starting a Lift - Driver



Figures 21, 22 & 23

This feature is the main part of the application. It allows a driver to start a lift instance, under the condition that it is within five minutes of the departing time and their location is less than five hundred meters from the start location to consider accuracy of the GPS or if the driver wishes to stop somewhere around the radius of the destination. When the driver selects the begin option which is in figure 17, the application checks the current time against the depart time and the user’s current location against the start location. They are then brought to a screen which shows their position on the map, shown in figure 21. When the driver gets within a five-hundred-meter radius of the destination which is highlighted by a red circle on the map, they can press the ‘Complete’ button. If they are not within five hundred meters of the destination they are presented with an alert that tells them they are too far and then updates their position on the map. If they are within the radius they are brought to the ratings page which will present them the list of passengers along with a five-star rating system, as shown in figure 22. When they are finished rating the passengers they are brought to their experience page. Here the user is presented with the distance they travelled, passengers they brought and their experience from the trip along with overall distance travelled, overall passengers that they shared lifts with and their overall experience.

The functionality in this part includes, for the screen in figure 21 there are two Google API’s being implemented which include JavaScript Maps and Geolocation to display the groups location on the map. The geolocation is also used to check if the lift can start by checking if the user’s current location is within five hundred meters of the start, before they click ‘Begin’ and five hundred meters from the destination, before they click ‘Complete’. If their coordinates are not within the finish radius the application presents them with an alert informing them that they are too far from the end to complete the journey and then calls the Geolocation to update their position on the map. The functionality of tracking the users position throughout the journey was tested by changing the geolocations function from getCurrentPosition() which updates the users position every time it’s called to watchPosition() which updates the user’s position on the map every time they move. The reasons for the choice of getCurrentPosition() being chosen is discussed in the section Review of Project. When they driver completes the lift, an AJAX request is sent to Looprac API that puts the driver and the passengers along with the lift ID, into the completed lifts table.

The functionality for the screen on figure 22, sends an AJAX request to the Looprac API which retrieves the list of passengers from the group for the driver to rate. The stars that are on this screen are from a library the author used called Bootstrap-Star-Rating (GitHub, 2017). This used the power of Bootstrap, JQuery and another library called FontAwesome (Gandy, 2017) to create the star rating system. It applies this look to an input type and uses Bootstrap classes and JQuery along with FontAwesome for the finished look. The stars were set to go in steps of 1.0, with a minimum of one star to a max of five stars and defaults to three when the page loads. When the driver selects a rating for each of the passengers that are in the list and continues, JavaScript is used to get all the passengers and ratings and an AJAX request is made to Looprac API implements an algorithm to calculate the user’s new ratings, this will be discussed in the Looprac API section.

The functionality for the screen in figure 23, involves an AJAX request that gets the distance of the journey from the database, which was entered when the lift was created and uses that and the number of passengers that the driver shared with on the trip. With these the drivers experience is calculated as follows:

Where:

D: distance of lift

n: number of passengers

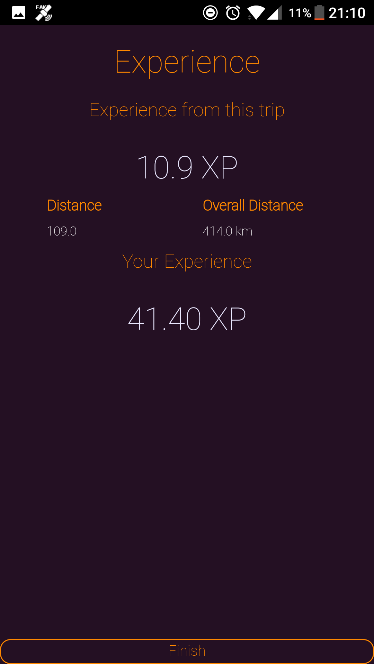
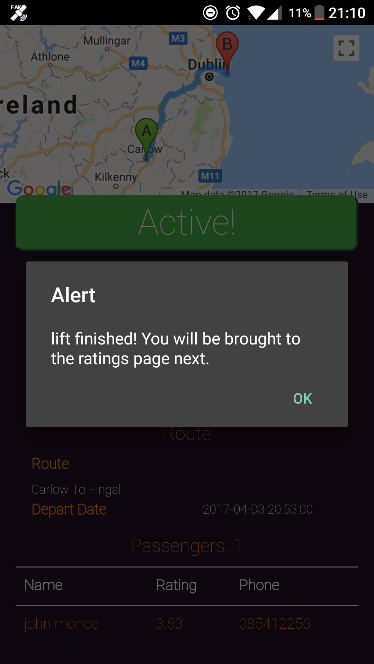
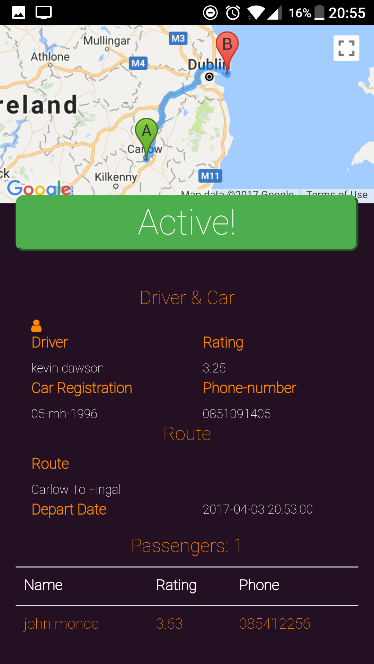
Figure 24

The driver gets an added percent for every passenger that they have in their car up to a max of six which is the available spaces limit they can advertise for a car. They get this small experience boost for being the driver whereas passengers experience is calculated as follows:

Figure 25

They author came up with these two types of experience calculations to reward the driver a little more for offering their car for the lift.

#### Starting a Lift – Passenger



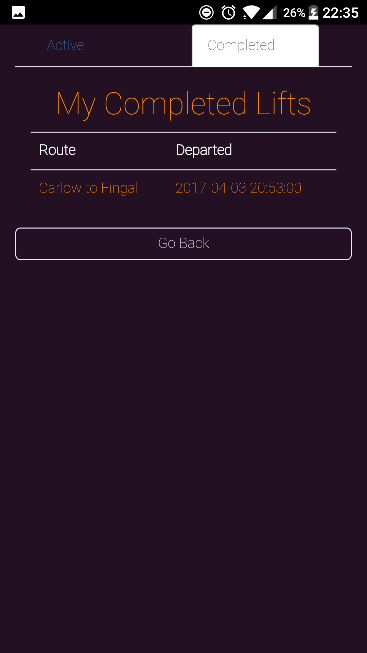
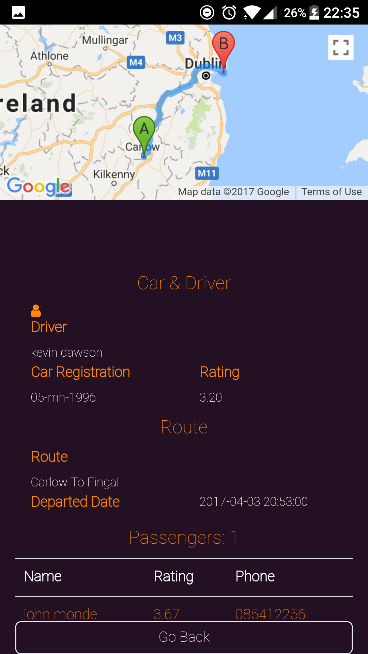
Figures 26, 27 & 28

This is the route that the passenger takes when a lift is in progress. The passenger will select ‘Begin’ from the group page which is shown in figure 14, on the same conditions as the driver, that they must be within a five-hundred-meter radius of the start position, and there must be less than five minutes from the depart time. When the lift begins, they are not presented a map like the driver is. Instead they are presented with figure 26, which activates the lift instance for themselves. When the lift is finished, they are presented with an alert informing them that the lift has been completed, shown in figure 27. They are then brought to the ratings page which is like figure 22, only it displays the driver as well as the other passengers to be rated. Once they have rated the other group members they are brought to their experience page. This displays a page similar to the driver’s experience page except that it does not display the number of passengers and overall passengers, it just displays experience and distance from the lift and their overall experience and distance, as shown in figure 28.

The functionality behind the screen in figure 26, is that once the user selects ‘Begin’ a function is called which sends an AJAX query to check if the lift ID has been inserted into the completed lifts table. This function is set on an interval for every fifth teen seconds. When the lift ID has been entered into the completed lifts table, along with the driver and passenger ID’s, when the check function finds that they are in the table the Looprac API sends back a response to the application letting it know that it has been completed. When this happens, an alert is displayed for the passenger informing them that the driver completed and finished the lift on their device. This is shown in figure 27

In the screen in figure 28, the passengers experience is calculated by the calculation in figure 25. The difference for the same lift between passenger and the driver with one passenger can be seen by looking at figures 28 and 23.

#### Viewing Completed Lifts



Figures 29 & 30

The feature of this part is to show a user all the lifts that they have completed as a driver. Navigation to this page is done by selecting ‘My Lifts’ from the menu in figure 10, and selecting the ‘Completed’ tab at the top of the page of figure 29. It displays a list of all completed lifts by the driver as shown in figure 29. When the driver selects any of the lifts from the list they are brought to the details page of that lift. This shows information regarding the lift such as the journey route being displayed on a map, details on the car and driver including the drivers name, the car registration that was used and the drivers rating along with route information such as the start and end counties or county areas and the time and date that this lift departed on. It also displays a list of the users which were passengers for this lift. The driver can also select any of these users from the list and it will bring them to their profile page, this page is shown in figure 30.

The functionality behind the screen in figure 29 is that an AJAX request is sent with the user’s ID to the Looprac API and it returns a list of lifts that are related to the driver from the completed lifts table in the database. When the driver is brought to the screen in figure 30, an AJAX request is sent to the Looprac API and relevant information is pulled from database tables to populate the page with the information that is seen. Two Google API’s are also implemented on this page: JavaScript Maps and Directions to display the journey on the map for the user to view.

#### Viewing Completed Groups

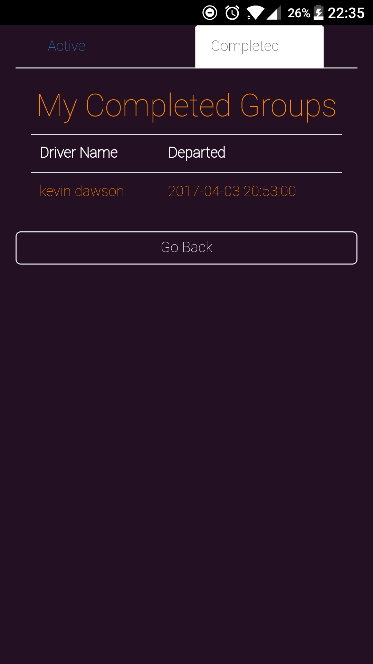
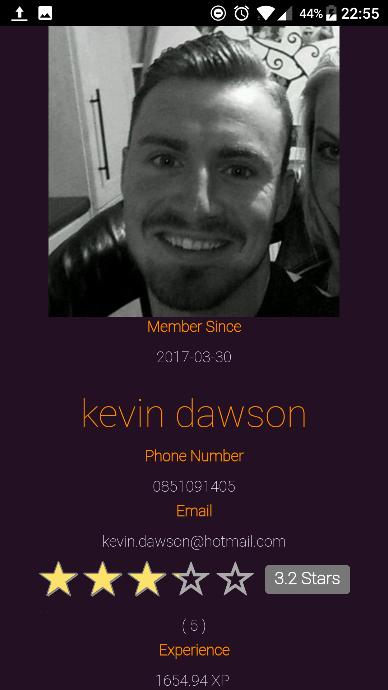
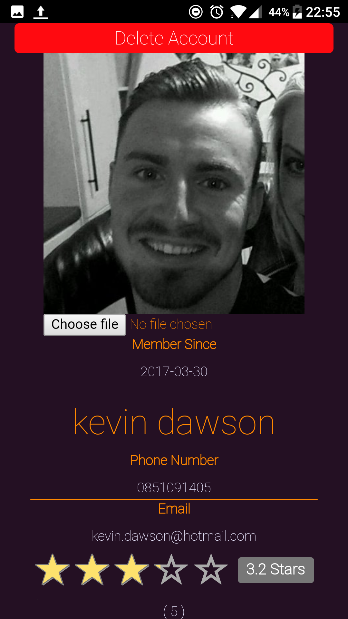


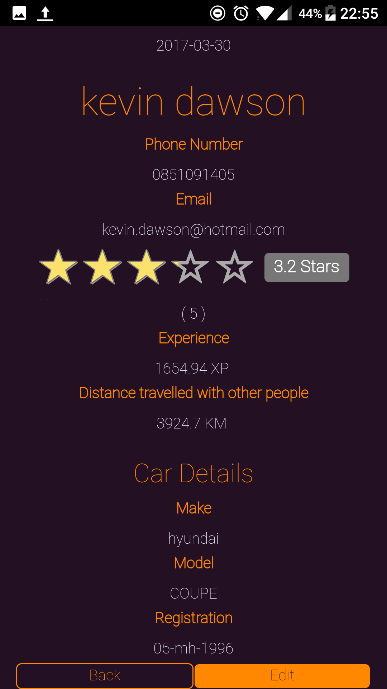
Figure 31

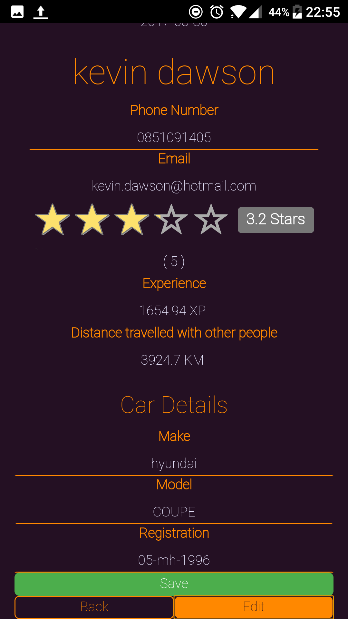
The feature of this page is that it displays a list of all the lifts that the user completed as a passenger. Navigation to this page is done by selecting ‘My Groups’ from figure 10 and selecting the ‘Completed’ tab at the top of the page in figure 31. When the user selects a group, which is identified by the driver of the group and the day it departed, they are brought to the lift details page, which is the same as the driver route, as shown in figure 30.

Functionality for the screen in figure 31 is the same as it is for the driver in the sub section above.

#### Viewing User Profile







Figures 32 & 33

The feature of this page is that it is a profile of the user. A user’s own profile can be navigated to by selecting their name at the top of the menu in figure 10. Other user’s profiles can be viewed by either selecting them from tables in requests or lift information, or by clicking on the profile icon’s, for example in figure 14. The profile is where it displays all information of a user including their name, contact details, overall rating, overall experience, overall passengers they have given lifts to if it’s not zero, otherwise it is hidden and car details if they have registered a car, otherwise this part is hidden. An example of a profile is in figure 32. The user is also able to edit certain information on their page such as car details, phone number and their profile picture. A user can only edit their car if they have first registered one, which they are prompted to do the first time they offer a lift. The user can also choose to delete their account by selecting the ‘Edit’ button followed by the ‘Delete’ button above their picture. Editable fields become underlined. An example of this is in figure 33.

The functionality behind this page, is an AJAX call is made to the Looprac API with the user’s ID and information is pulled from different tables in the database such as user, user rating, experience and car details where it is needed. The JQuery and Bootstrap star rating library is implemented in this page also to display the user’ s overall rating and number of times they have been rated.

If the user chooses to edit their page, JavaScript enables the fields that are editable, and when the save button is selected an AJAX request is sent to the Looprac API with the new information and the data base details are updated

#### Viewing Leader Board

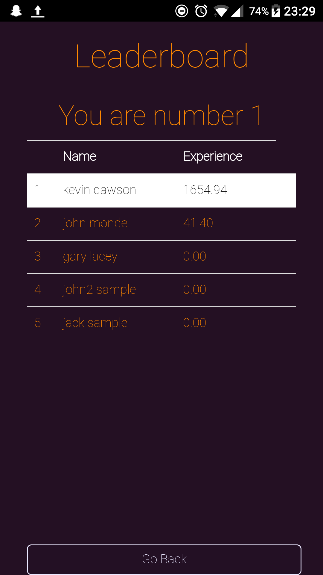


Figure 34

This feature is a way for a user to see how they compete against all other users of the Looprac application. It displays all users of the application in descending order based on their overall experience. It also highlights the logged in user so they can easily find their rank amongst the table and their rank is also displayed at the top of the table, as shown in figure 34. The user can click on any other user’s name in the table and it will bring them to their profile. This can also be used by users as a form of search, amongst all users of the application.

The functionality behind this page is an AJAX call is made to the Looprac API and it retrieves all user’s names and experience from the relevant tables in the database. When the data is sent back in the response, a function checks each user ID that is sent back with each record and if it matches the logged in user’s ID it is highlighted in the table to be found easily. The row number that they rank on is then taken and displayed above the leader board.

### Back End: Looprac API

#### Overview

The Looprac API is a Flask application that is hosted on pythonanywhere.com and is programmed in Python 3.5. The main purpose of the API is to handle requests sent from the front-end devices with the Looprac application running on them and bridges the gap between the application and the MySQL database. The other purpose of the API is to do some processing of data before it is entered into the database. The following sections will look at the functionality that the API offers.

#### Password Hashing

When a user registers with the application and provides their details, including the password they wish to set for their account, when it is sent to the API the password is taken out of the request and using the Python library Hashlib (Docs.python.org, 2017). This takes in the user’s password and creates a 64-bit hash of the string using the SHA256 cryptographic hash function. This 64-bit hash is then inserted into the table along with the other user information.

When a user tries to log into the Looprac application, using their email and password, which are sent in an AJAX request from the application, if the email address is found in the table the relevant 64-bit hash key is retrieved. The password that was sent with the email address in the AJAX request is then turned into a 64-bit hash key and compared against the retrieved key from the database table. If they match the API responds to the application granting the user access to the Looprac.

#### User Overall Rating Algorithm

Each time a user completes a lift in the Looprac system, either as a driver or a passenger, and they rank the other users while they themselves are ranked. When each user has finished rating the other users the application sends the ratings with an AJAX request. The ratings are then gotten from the request. The number rating is added to the database in the relevant field, and then the overall number of ratings for that user is retrieved along with the number of each star rating they got and the weighted average is gotten by putting the retrieved data into a function that contains the following calculation:

Figure 35

This returns the weighted average, so for example if a user has

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1\_Star | 2\_Star | 3\_Star | 4\_Star | 5\_Star | Total ratings |
| 56 | 34 | 102 | 45 | 28 | 265 |

Figure 36

Then the following is worked out:

= 2.830188

= 2.83 overall rating

Figure 37

The result gotten from figure 37 is then stored in the user ratings table and that figure is what is used for the input of the Bootstrap star rating system mentioned above.

#### DBcm – Data Base Context Manager

For the API to be able to talk to the database and execute SQL queries, the author used the DBcm library (Barry, 2017) which provides an easy way to open a connection to the database using predefined credentials, allowing SQL queries to be executed, along with other operations such as fetching data and then automatically closing the connection with the database once the scope ends.

# Conformance to Specification and Design

## 3.1 Functional Specification Conformance

Overall the system conforms to the functionality that the author outlined in the functional specification.

The use cases that were outlined in the functional specification were implemented into the Looprac system. There were a few additional use cases added in to the final product which are not documented in the functional specification, the added functionality is specified below.

## 3.1.1 Functional Updates

### Delete a Lift

This functionality was added during iteration three of the project and involves a user that has advertised a lift to delete it, under the conditions that there are no passengers associated with that lift. The benefit of this use case is if a user makes an error in selecting an address, or departing time, they just don’t accept any requests for it and delete it from the system.

#### Detailed Use Case

**Actors:** driver, Looprac API

**Brief description:** this use case begins when a driver wishes to delete a lift they created. The user navigates to ‘My Lifts’, and finds the lift from the list. The user selects the lift. The application validates that the lift can be deleted and displays the delete option to the driver. The driver selects the delete option and the application asks the driver to confirm their choice. They select ‘ok’ and the application sends the lift ID to the Looprac API and the Looprac API deletes the lift from the system. The use case ends when the application informs the driver that the lift was deleted and returns them to ‘My Lifts’

**Main success scenarios:**

1. The user navigates to ‘My Lifts’
2. The user selects the lift from the list of active lifts
3. The application validates that the lift has no passengers associated with it
4. The application displays the ‘Delete’ button on the screen
5. The driver selects delete
6. The application asks the driver to confirm their decision
7. The driver confirms the operation
8. The application sends the lift ID to the Looprac API
9. The Looprac API deletes the lift from the system
10. The application informs driver that the lift has been deleted
11. The application returns the driver to ‘My Lifts’

**Alternatives**

**3A.** The lift has passengers associated with it

1. The application doesn’t not make the delete option visible to the driver

#### System Sequence Diagram

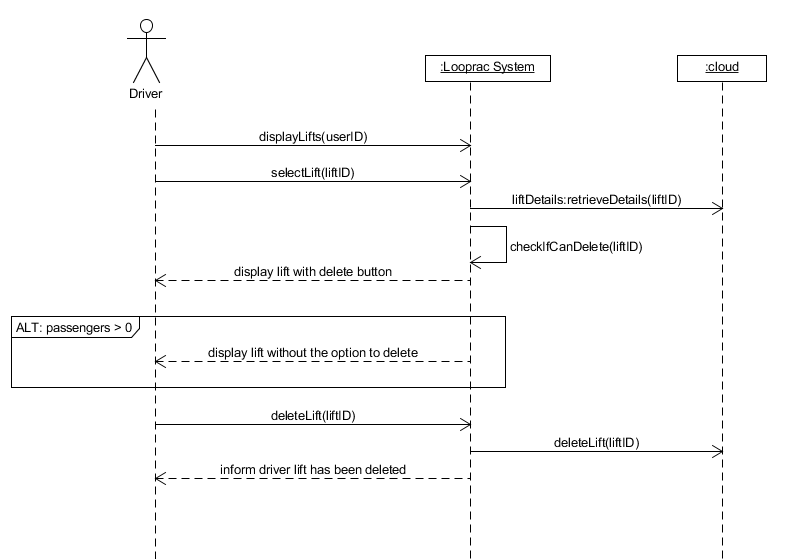


Figure 38

### Begin Lift Instance -Driver

This is when a driver wants to begin a lift instance. They navigate to the lift they want to start and select ‘Begin’. The application validates that the lift can start by checking – there is at least one passenger, the current time is within five minutes of the agreed departure time, and the driver’s location is within five hundred meters of the start location. The application will then display the driver’s location – which represents the car group, on a map. When the driver wants to complete the lift instance, they start the Complete Lift Use Case specified in the functional specification.

#### Detailed Use Case

**Actors:** driver, cloud

**Brief description:** This use case begins when a driver wishes to begin a lift. The driver navigates to the lift and selects begin. The application validates that the lift can begin. The application displays the groups location on the map. This use case ends when the driver wants to start the Complete Lift use case.

**Main success scenario:**

1. The driver wishes to start a lift
2. The cloud updates drivers list of lifts
3. The driver navigates to the lift and selects the lift
4. The application retrieves the lift details from the cloud
5. The driver selects the option to begin
6. The application validates that the lift can start – checks that number of passengers is one or more, the current time is within five minutes of depart time, and driver’s location is within five hundred meters of start location
7. The application displays the driver’s location (represents the car group) on a map along with the start and end destinations

**Alternatives**

**3A.** The lift has no passengers assigned to it

1. An alert informs the driver that the lift cannot be started with no passengers, and to check their requests for any passengers wishing to join this trip

**3B.** The current time is greater than five minutes of depart time

1. An alert informs the driver that it is too early to start the lift from the agreed start time

**3B.** The driver’s current location is not within five hundred meters of the start location

1. An alert informs the driver that they are too far from the start location to begin the lift

#### System Sequence Diagram

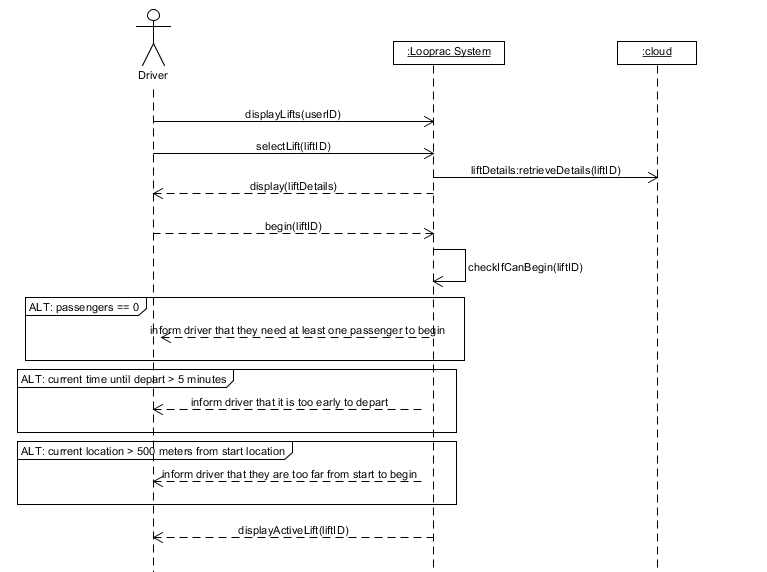


Figure 39

### Begin Lift – Passenger

This is when a passenger has met up with the driver and is in their vehicle. For the passenger to join the drivers lift instance, they must navigate to ‘My Groups’ and select the corresponding group for the lift. The passenger selects begin and the application validates that they can begin – checks the current time is within five minutes of the depart time and the passenger’s location is within five hundred meters of the start location. When the lift has begun, the begin button changes to ‘Active’ to make them aware that the lift has begun. The lift is complete when the driver finishes the Complete Lift use case.

#### Detailed Use Case

**Actors:** passenger, cloud

**Brief Description:** This use case begins when a passenger is in the driver’s vehicle and wants to join the drivers lift instance. The passenger must navigate to the car group in their ‘My Groups’ section and select the begin option. The application validates that they can begin. The application changes the text of the ‘Begin’ button to ‘Active’ to show the passenger that the lift has begun. This use case ends when the passenger is informed that the driver has completed the Complete Lift use case.

**Main success scenario:**

1. The passenger wishes to join the drivers lift instance for a lift
2. The application retrieves the passenger’s groups from the cloud and displays them
3. The passenger navigates to the group and selects it
4. The application retrieves the groups details from the cloud and displays them
5. The passenger selects the ‘Begin’ option
6. The application validates that the passenger can begin – checks that current time is within five minutes of depart time and current location is within five hundred meters of start location
7. The application changes the ‘Begin’ button to ‘Active’
8. The application checks every fifth teen seconds if the driver has finished the Complete Lift use case
9. The application informs the passenger that the lift has been complete

**Alternatives**

**4A.** The current time is greater than five minutes of depart time

1. An alert informs the passenger that it is too early to start the lift from the agreed start time

**4B.** The driver’s current location is not within five hundred meters of the start location

1. An alert informs the passenger that they are too far from the start location to begin the lift

#### System Sequence Diagram

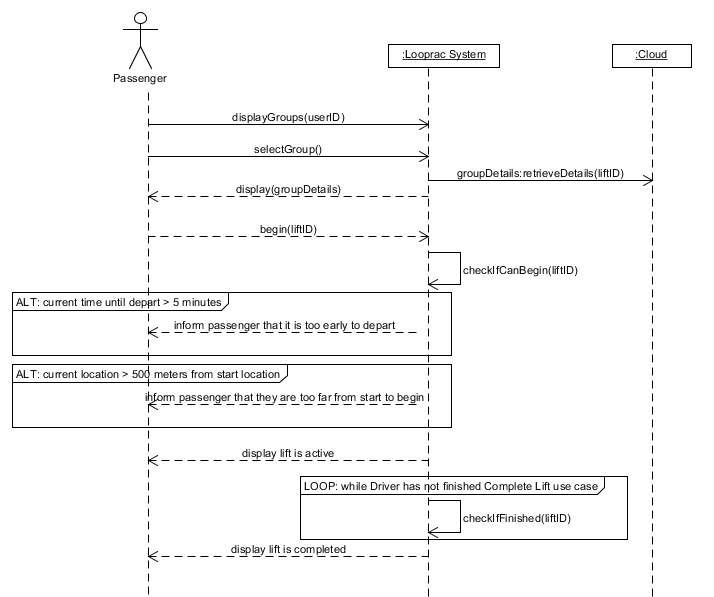


Figure 40

## Updated Main Use Case

The main use case diagram that was presented in the functional specification has been updated below in figure 41, to include the three new use cases that are specified in the section above

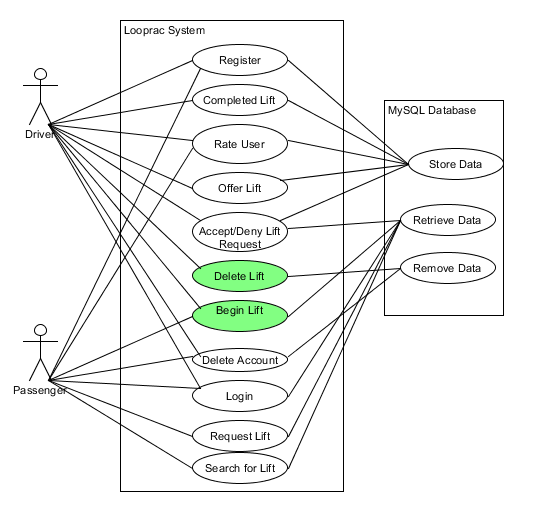


Figure 41

## 3.2 Design Document Conformance

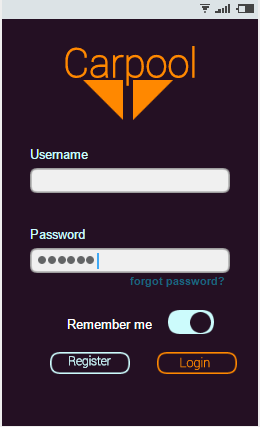
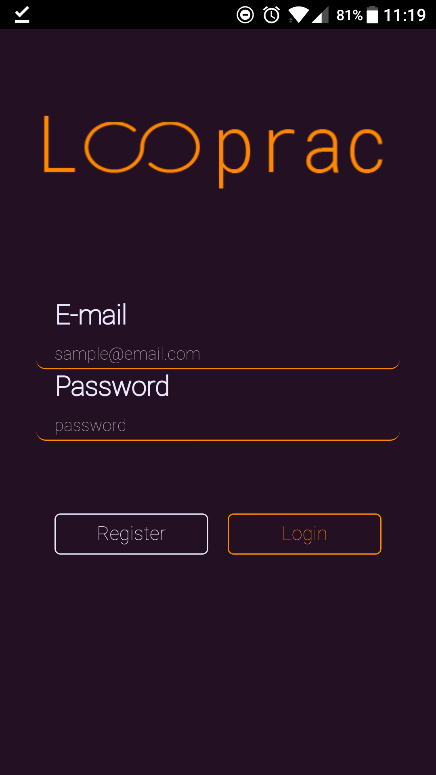
The project conformed to the Design Document as it was stated. The database design was implemented and used with no additional changes to its structure or design. The use cases that were added through duration of the project are specified in the above section. Where the final project did not conform to the design document was the user interface designs and the flow through the system. These will be discussed below.

## 3.2.1 Design Updates

### 3.2.1.1 Overall Design

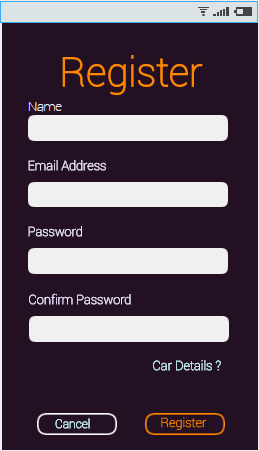
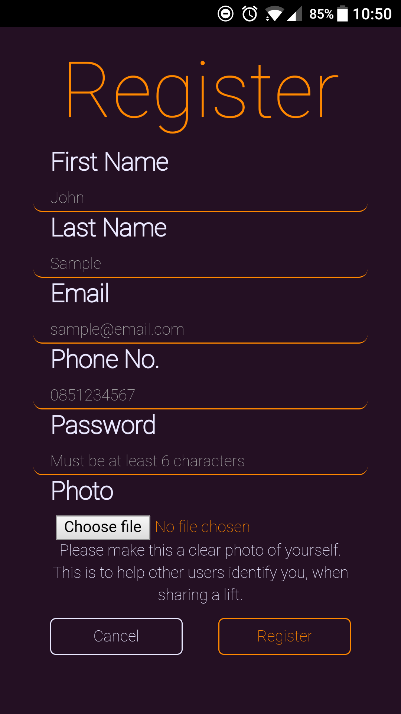
The overall user interface design that was implemented, used what was declared in the design document as a guide. The colour palette which was selected and specified in the design document remained the primary colour palette for the project. Some of the interfaces however were different than what was drawn up in the design document. Below are screenshots of the finished project and their drawn up counter parts from the design document to show where the UI design conformed with the document and where they differ. Comparison is only provided for what was documented in the design document.

#### 3.2.1.1.1 Login



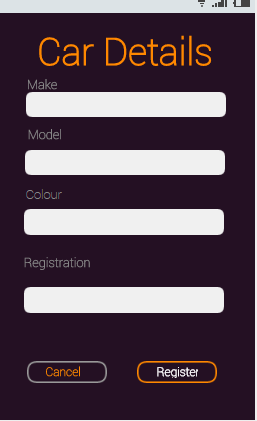
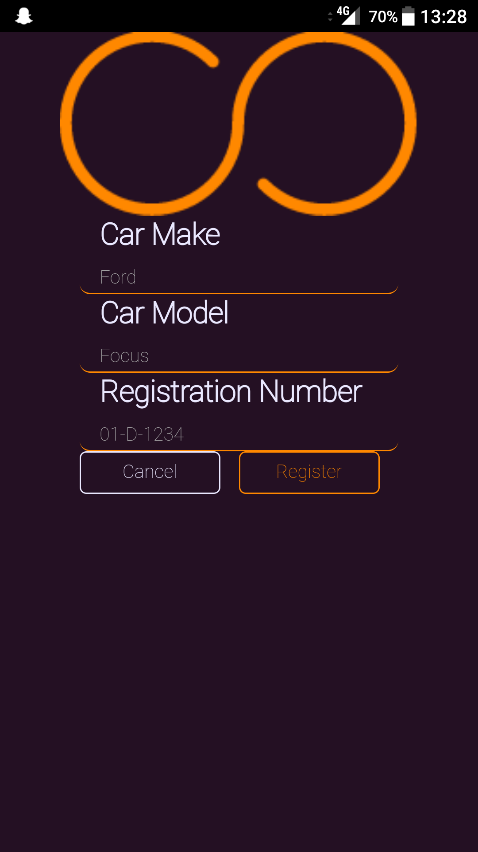
Figures 42 & 43

#### 3.2.1.1.2 Register



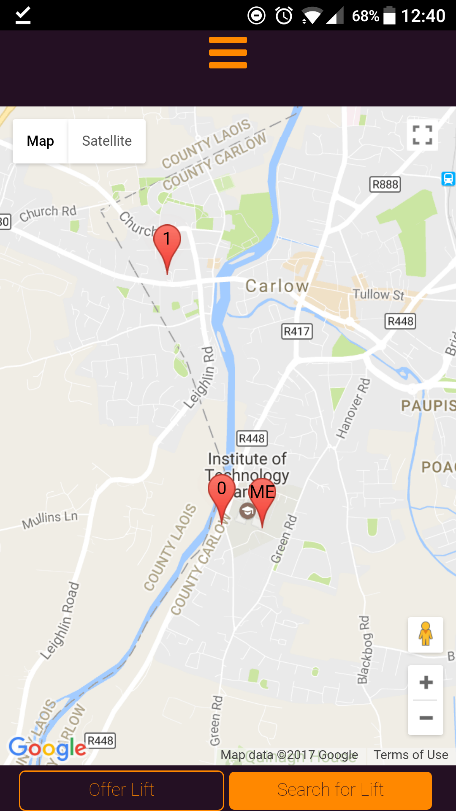
Figures 44 & 45

#### 3.2.1.1.3 Car Register



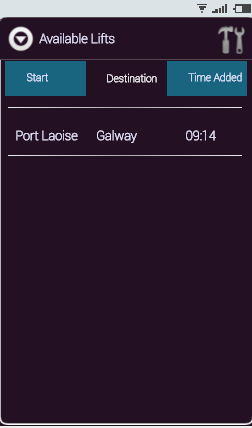
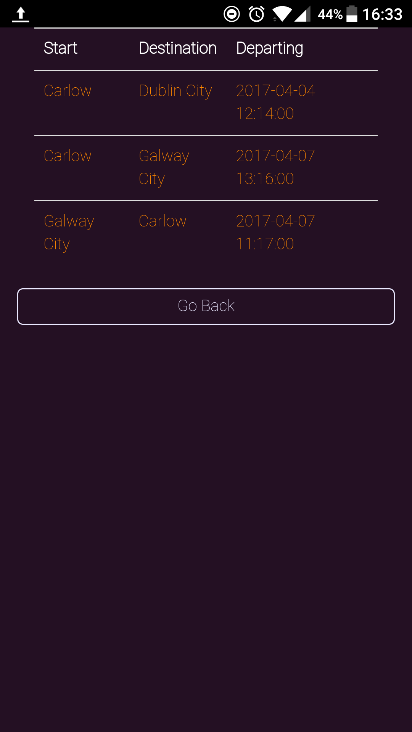
Figures 46 & 47

#### 3.2.1.1.4 Main Page



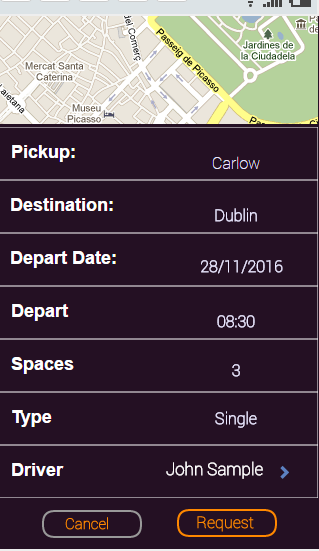
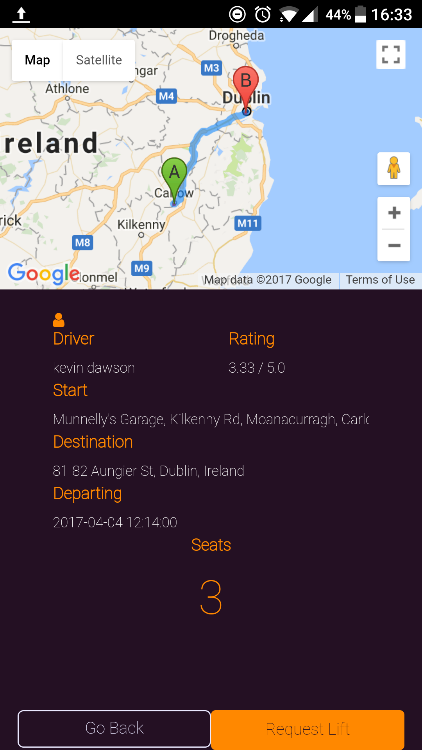
Figures 48 & 49

#### 3.2.1.1.5 Search for a Lift



Figures 50 &51

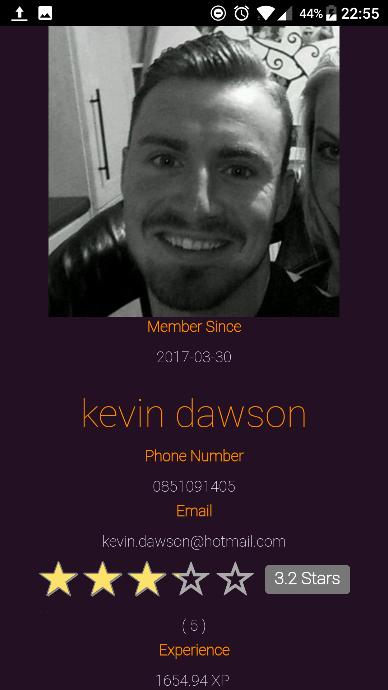
#### 3.2.1.1.6 Request a Lift

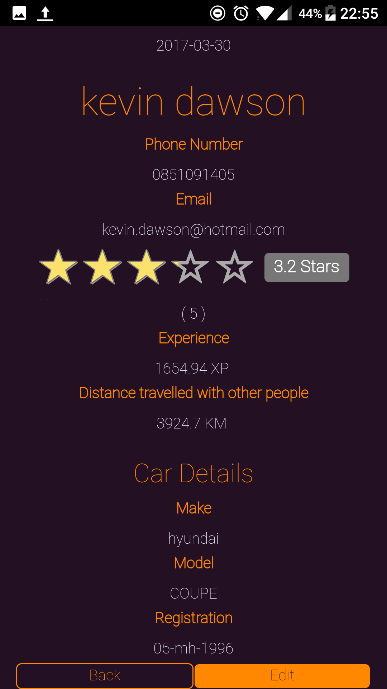


Figures 52 & 53

#### 3.2.1.1.7 Profile

This page originally was just a page to display details about a driver. It was further implemented in the final product as a profile page that all users have.

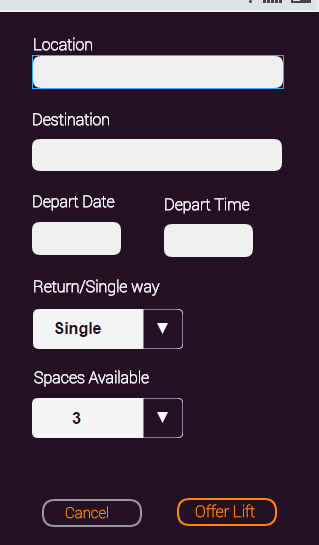
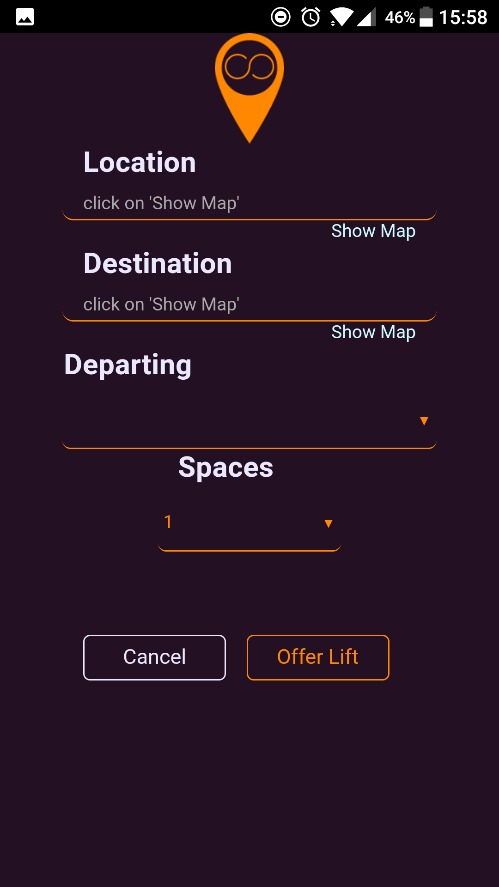




Figures 54 & 55

#### 3.2.1.1.8 Offer a Lift

In figure 56, the initial design had taken in a lift type, either return trip or single. This was taken out to lower complexity and problems in case a passenger only wanted a lift one way on a return trip. All lifts were changed to single way trips for this reason



Figures 56 & 57

### 3.2.1.2 UI Flow

The UI flow again conforms with the originally specified flow in the design document in some areas, and in others it differs. Below in figure 58 shows the finished projects UI flow for the main activities of the application.

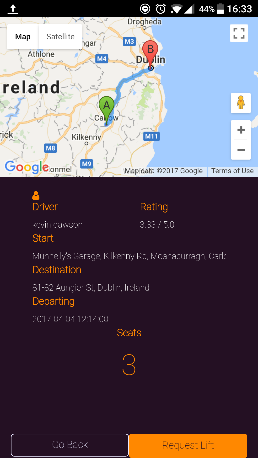
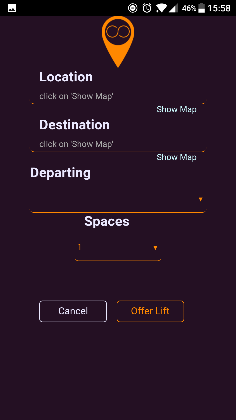
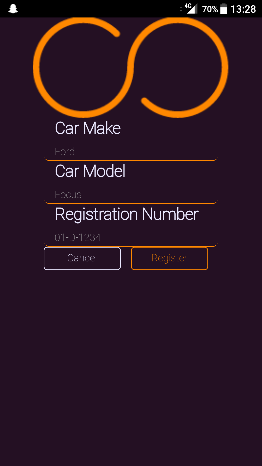
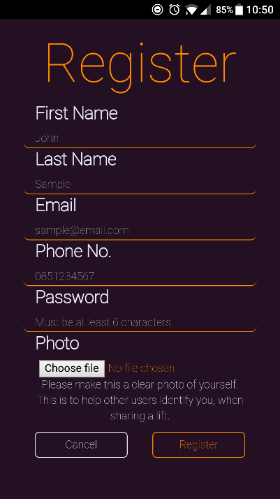
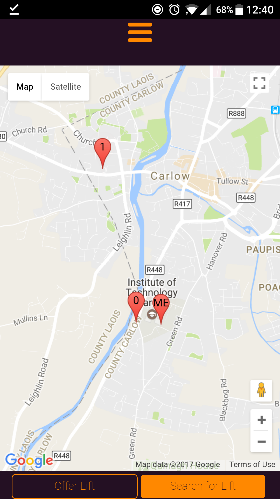
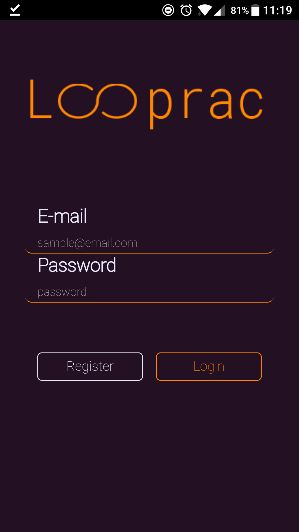
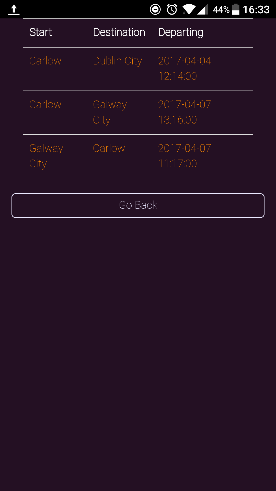


Figure 58

# Description of Learning

## 4.1 Technical

Throughout the project the author used many technologies which included some which the author was familiar with and others that required research and practice to get to a level that it could be implemented.

The use of Adobe PhoneGap was new to the author, however, the fact that it was built using HTML 5, CSS and JavaScript allowed the author to strengthen their knowledge and experience with these technologies. Creating a hybrid phone application was a new concept to the author, although the author had no experience in mobile phone development at the commencement of the project, the experience gained in creating this hybrid application is unmeasurable.

Technologies that were new to the author included Python, which the author previously had no experience in. What helped in learning and understanding this technology was that it was being thought in one of the authors other subjects. The author could then apply what he learnt into the project. This enabled the ability to develop the entire Looprac API from scratch in Python. Another new technology that went with this was Flask, which the author had no previous experience in either. Like Python, this was also part of another subject that the author was taking during the year and allowed him to utilise Flask alongside Python which is another major component of the Looprac API. JQuery and AJAX were more new technologies to the author, again having no previous experience with them, through research and practice the author learnt how to use JQuery and AJAX to send requests with JSON data from the application to the Looprac API and to receive response to those requests, another fundamental element in the creation of Looprac.

Working with Google’s API’s was also new to the author, this presented a steep learning curve at the beginning of the project. However, as the project progressed the author became familiar with the API and so utilised more Google API’s to increase the functionality of the project.

MySQL experience was majorly increased over the course of the project, having not used the technology in some time. Learning about PythonAnywhere and MySQL’s integration with it allowing the author to learn how to use the likes of Python to communicate with the database by executing SQL within the Python code thanks to available Python libraries. Furthermore, in regards to PythonAnywhere, learning about this was highly valuable, which helped the author to learn in dealing with a server, although PythonAnywhere made this process easy, experience was still gained.

## 4.2 Personal

Over the duration of the project the author underwent personal development in the form of strengthening personal skills as well as acquiring new skills.

A skill that the author strengthened was of their approach to solving a problem. Over the course of the project, many problems arose. Earlier on in the beginning of the project, these problems were tackled head on with no research into the problem itself, basically working on a trial and error approach. Although in some situations trial and error is a perfectly fine approach, the author learnt to take a step back from the presented problem and research into the possible reasons as to why it was happening, then finding solutions and implementing them. This skill strengthened over the course of the project and by the end problem solving became a cleaner and more efficient process.

Another personal skill that was gained by the author was confidence in their ability to develop and implement such a project on their own. At the start of the project the task seemed daunting and the author felt overwhelmed by it, to point of questioning if it was even possible for him to finish the project. However, over the duration of the project the authors confidence in their own abilities increased to a point where the negative thinking was no longer a factor and allowed the author to progress with the development of their project.

Time keeping and organisational skills were also strengthened, although the author didn’t get to implement every element that they wanted due to time constraints, overall their time management skills increased in balancing the project with the other subject assignments and tests over the course of the year.

# Review of Project

## 5.1 What went right

Overall the author feels that the project was a success. The measure of success outlined in the Vision section of the Functional specification, which stated that ‘The application and cloud API are at a standard that the application is useable’, has been achieved. The application is at a working and useable standard for normal users.

With the idea of the car sharing application and the potential for all the different types of functionality that could be implemented into the application the author got to a level where the core functionality was met and working and from there added functionality onto it. Due to time constraints of the project the author had to pick a point where no more functionality would be implemented and what was implemented to be working to a useable standard. This the author feels was achieved.

With the design of making the core functionalities as fluid as possible for the end user, from registration to either offering a lift or searching for a lift, to beginning a lift instance the author had stated in the functional specification that he wanted to create a positive user experience, where there was no unnecessary jargon or clutter, and to have as few clicks as possible was also achieved.

## 5.2 What went wrong

One thing that the author feels could have been better, was the time dedicated to testing the application more. A small amount of manual testing was done on the application by the author and getting family and friends to try it, but the author feels that the project would have benefited more from more testing of the core functionalities. The author had wanted to test the application with the use of Selenium to test the application, but due to time constraints this was not achieved.

Another area where the author felt could have been better, was time management itself. Although, the author feels that their time management skills improved over the course of the project in general, parts of the project were omitted due to the time constraints, and the author believes that had time been better managed it would have been possible to achieve them.

An area where the author felt went differently than, rather than wrong, was with the Google geolocation API for the screen when the driver begins a lift, as seen in figure 21. Is that the author was using the free account for Google API’s which applied a quota of 2500 requests a day. The author wished to implement a function called watchPosition() over the currently implemented getCurrentPosition() as the first function updated the drivers position on the map every time they moved so they could see their position in real time over whenever they press the ‘complete’ button and their position is updated. This idea was scrapped for the sole purpose that every time watchPosition() updated the drivers position it was rapidly using up the daily quota, and when this happened the author and application had to wait twenty-four hours for the quota to reset. On the free account this option was just not feasible.

## 5.3 Outstanding/Missing Work

Overall, the applications product and work completed adheres to the workplan that was stated at the commencement of the project. However, as stated in the Research document under the Project Work Plan section, the author didn’t get to implement the achievements or badges functionality that he wanted due to time constraints. The idea here was to provide users with badges to display on their profile for milestone feats within the application, such as distance travelled with other users, or number of people a driver has shared his car with.

## 5.4 Technology Changes

At the start of research phase of this project the author believed that he was going to create an Android application using Java. However, after researching more the author then changed to creating a Hybrid application with the use of HTML5, CSS and JavaScript. This was the only major technology change throughout the project, and it was advantageous at the point that the change was made as it was very early on and the coding part had not commenced. There were no other technology changes during the duration of the project.

## 5.5 What the author would do differently

If the author was to do the project again, the areas that he would do differently are:

In regards to the password hashing being done within the API, and firstly being sent to the API in a request as plain text, the author knew this was bad practice and unsecure, and having a chance to do it again, he would look into JavaScript libraries to hash or encrypt the password on the client side, before it was sent.

He would have specified more time on testing, as more testing results in better quality code. The mistake he made was leaving it too close to the end that only a small amount of testing was conducted on the application.

Another area that the author would have like to have spent more time in is to optimize the applications code. Getting the functionality and application to work took priority over optimizing the code most the time.

Another area the author would do differently is implementing better time management, by setting monthly objectives and then breaking them down into weekly and daily objectives. This could have prevented the lack of time for testing and the omitted functionality that is stated in the section above.

## 5.6 Future Features

If the author was to continue working on this project, there are a few bits of functionality that he would like to have working as future features. Theses bits of functionality include user’s being able to store a favourite driver, making it easier for them in future to see lifts that they advertise.

The author would implement the badges and achievement system for all user’s. Another part of functionality would be to provide a search which would allow users to find other user’s or to help in finding a lift.

In regards to lifts, the author would implement multiple stops on the route, to perhaps collect or drop off users at different locations instead of a general start and end destination.

Apart from the badges and achievements, none of these were scheduled to be implemented over the course of the project. These are all just features that the author would have liked to implement in the future.

# Acknowledgements

The author would like to sincerely thank the help and support offered by his supervisor Joseph Kehoe. He would also like to thank his girlfriend and family for the support over the year. This project would not have been completed without the support of them all.

# References

GitHub. (2017). *kartik-v/bootstrap-star-rating*. [online] Available at: https://github.com/kartik-v/bootstrap-star-rating [Accessed 3 Apr. 2017].

Gandy, D. (2017). *Font Awesome, the iconic font and CSS toolkit*. [online] Fontawesome.io. Available at: http://fontawesome.io/ [Accessed 3 Apr. 2017].

Docs.python.org. (2017). *15.1. hashlib — Secure hashes and message digests — Python 3.4.5 documentation*. [online] Available at: https://docs.python.org/3.4/library/hashlib.html [Accessed 3 Apr. 2017].

Barry, P. (2017). *DBcm 1.6.3 : Python Package Index*. [online] Pypi.python.org. Available at: https://pypi.python.org/pypi/DBcm/1.6.3 [Accessed 4 Apr. 2017].