Meraki Bike

Deliverable 3

Team 7

10/10/2017

Lachlan Brewster (37248471)
Morgan English (78631141)
Mitchell Fenwick (72182882)
Joshua Meneghini (67842807)
Connor McEwan-McDowall (91234505)
Aidan Smith (24294411)

Contents

Executive summary

This report provides the current business model of an analysis and monitoring system for cyclist related data. This includes the general business and system context, providing the overall purpose of the software, stakeholders and requirements, main risks involved, and an overall project plan including a detailed UML diagram, GUI prototypes, and a deployment model.

The main service provided by the application is to provide analytics of bike trip data collected by either data sharing companies, or individual cyclist users. Data also provided is the location information about Wi-Fi locations, bike stations and retailers. The analytic capabilities would be aimed at retailers and Wi-Fi companies. Knowing the most frequently travelled routes and busy times, combined with the user's information could help a specific retailer tailor their business to specific demographics, hence increasing business. The secondary use of the application is for users (cyclists) to be able to search for the nearest Wi-Fi, retailer or bike station based on search criteria, including calculating distances between specified locations among other features discussed in the report.

The stakeholders of the application include retailers, Wi-Fi companies, data sharing companies, councils, cyclists, and non-cyclists. The high priority stakeholders are retailers and Wi-Fi providers we aim to market our analytic capabilities towards. The application is mainly focused on business prospect analytics rather than being a cyclist centric application.

A severe, but unlikely, potential risk for this application's longevity is that if the application were no longer supplied with up to date bike data, the analytics would be useless. This risk among others are discussed in detail, along with how our product differentiates from other similar bike applications already released.

1 Business and System Context

1.1 System Context

The application we will implement is designed to receive data from the user (who most likely located the data from specific data sharing companies). Specifically, information about the bike trips cyclists take, Wi-Fi and retailer locations. Trip information includes the start station (where the bikes are taken from), end station (where the bikes are returned to), trip duration, specific bike used, and other relevant information about the cyclist. Retailer information includes the name of the retailer, the type of retailer and address. Wi-Fi information includes the provider, the type, the location and, in some cases, remarks about the Wi-Fi. The system will be able to view the data and complete an analysis of the data given, producing statistics relevant to our stakeholders, for example, popular routes, age and gender demographics.

The system will include analytic features designed to meet the demands of retailer and Wi-Fi providers. Companies would be more interested in the flow of cyclists, that is their start/end points and popular routes, as well as seeing the other retailers and Wi-Fi outlets around them. A large company looking to expand with franchises would likely get the most from this application as they can use it to make informed decisions about better locations for new stores. Other, smaller companies and retailers could also potentially benefit from seeing more detailed information about trips. Seeing peak times and user information such as gender and age would allow them to tailor specific services to specific demographics.

The system implemented for retailers and Wi-Fi providers may also make the application of use to councils. However, much of the analytic features they will be after will be very specific. The addition of these features is therefore not a high priority as city councils are the minority of our target users and a low priority stakeholder. City councils may find some of the analytic features regarding trip data of use, such as knowing most frequently travelled routes and times so they can improve and customise civil infrastructure. This is to be taken as potential future opportunity for expansion rather than a priority in this application’s design. This is not only because of the specific nature of the features that would be implemented, but also because the system will only be implemented over New York City and therefore the only council who could potentially find the application useful would be New York City Council.

The system will provide users with information regarding the nearest stations to their selected start and end destination. It will also allow them to look for retailers and Wi-Fi hotspots on their route between two stations of their choice. Since the end user will also be able to store data they will be able to keep track of their own cycling history and gather some analytics about their trips over time such as duration and distance.

1.2 Worth of developing

The bike sharing business is expanding at an increasing rate, and with that growth there will be an ever-increasing need for applications that can support that growth, as seen by multiple news outlets discussing the “billion-dollar boom” of bike sharing. “Bike share is growing at an astounding clip across the U.S, with over 88 million trips made on a bike share bike in the U.S. since 2010. In 2016 alone, riders took over 28 million trips” (Nacto. 2016). There are multiple examples of new and exciting bike share companies that plan to expand into other areas cropping up. For example, the company Spin, “A California company will dump 300 dockless share bikes across the Big Apple on Monday” (Furfaro, D. 2017).

There is no clear sign of the expansion slowing down anytime soon. There will be a void of bike sharing support apps to be filled, so our venture is a very valid one.

The application is primarily useful for retailers and Wi-Fi providers as they can view analytics about cyclists in New York City. This information is sought after by the retailers as they can use it to target customers and get information about places to create new retail stores or Wi-Fi hotspots. Due to the usefulness of the application to Wi-Fi providers and retailers, they are likely to purchase the application. Due to potential financial gain from our application the Wi-Fi providers and retailers are likely to be willing to spend a reasonable amount to purchase it. In addition, they are numerous enough to make the application fiscally viable.

The specific future possible monetisation of the application will vary depending on future improvements made and features added. For instance, this project’s application will have only one model including all features, however a future possibility may be to split off the analytic features into a “premium” application at a higher price, while simultaneously lowering the price of the “standard” application potentially $0.

1.3 Description of services

The services aimed at Business users (Retailers, Wi-Fi providers, City Councils) will be:

- Upload large data sets of bike trips stored in a csv
- Run analytics on large sets of bike trip data such analytics will:
- Find information about the most and least common times for cycling
- Find information about the most and least travelled routes cyclists take
- Find information about the cyclists that pass an area
- Be displayable in a Google Maps interface
- Be displayable in graph format

The services aimed at Casual users (Cyclists, Non-Cyclists) will be:

- Upload bike trips individually into the GUI
- View bike trips on a Google Maps interface
- Plan a route between two destinations and see the closest bike stations to the start and end points of their journey
- Find the nearest Retailer of a specific type to a selected destination
- Find nearest Wi-fi Provider to a selected destination
- Find Wi-Fi and Retailers of specific type on their route

1.4 Business opportunities

1.4.1 Unique selling points

Analysis Heavy Application
As our application is largely focussed around the analysis of cyclist data as our primary market is much more concerned with this, due to this we will not be able to implement some features that may be beneficial to casual users. We will however, be able to target retailers and Wi-Fi providers, who would have a considerable interest in the analytics our application would provide. Many other bike-sharing applications fail to capitalise on this (as they focus more on the casual user experience), resulting in what we see as a gap in the market.

Pre-existing Products
There are already many applications, such as MoBike, oBike and Spotcycle (Google play. 2017), that provide more features to casual users than our application will provide in terms of route planning. We stand out by showing information about the bike sharing stations in their city. However, this has the major disadvantage that if there is no bike sharing companies in their city our application has no advantageous selling feature to casual users (this is of course a non-sequitur for this application based solely in New York City). Another notable caveat at this stage is competing with mobile applications. Such applications are targeted at casual users (Google play. 2017) and so are unlikely to impact on our Business users userbase.

1.4.2 Target markets, potential customers

Our target market is retailers and Wi-fi providers, much of the interests of these targets overlap so we can create an application appealing to both without having to create many specialised functions.

Retailers and Wi-Fi providers will get the most from the analytics and they are much more likely to be willing to spend money on this software as the analysis that it provides will give them information they can use to improve their business or show the best options for expansion into a new region.

We will build in some functionality focused on casual users. The rational for which is to build up a casual userbase to be potentially extended in later versions to retrieve data from. This functionality will be clearly separated from the analytic features. This means that if the application were to be split into a low cost or free “standard” application and a higher cost “premium” application (which would include all the analytic features), this would be incredibly easy as it would require the deletion of features with hardly any refactoring of the code.

1.4.3 Upgradeability and Extension

The application we are producing is heavily focussed on analytics of data and due to its implementation on a desktop it is not of much use to casual users. A mobile application could be developed to allow for cyclists to use the application on the go and get real time tracking information. Doing this would also require storing information about retailers and Wi-fi providers to an online database so that they would not be stored within the mobile application. As a result, we plan to introduce an online database into our final project so that the potential future upgrade of a mobile application will already have some grounding within the code.

Since the application will primarily be aimed at retailers and Wi-fi providers, there will be some analytics for other users that may not be implemented as they would not improve the sale potential of the application to the target market. However, overtime new analytics could be added to expand target market to city council, bike-sharing companies etc. The relevant analytical tools would require a partnership with these stakeholders as they would hold the necessary data. For instance, Citi-Bike could potentially provide us with more information about the condition of their bikes, allowing to plan scheduled maintenance based on usage. This would be relatively easy to include given the wide-ranging use of other analytic tools that we plan to include in our final project.

Currently we only aim for the application to be effectively operated within New York City. However, extending the application to other cities and potentially any other location is an obvious next step to make. This would be relatively easy to implement as the Google Maps API that we plan to use for viewing of routes and locations will have that functionality inherited. We would only have to change the other aspects of our project such as data entry and data filtering.

Due to the scope of this assignment and time restraints these features are strictly extensions that could be added to improve the program. We do not plan to include these features in our first product.

2 Stakeholders and requirements

2.1 Stakeholders

The stakeholder table below shows who our stakeholders are, how they are going to be using the application and a weight showing their level of interest in the project. The description also shows why they are a stakeholder in our project. This table helps visualize and bring together many of our possible use cases, as our use case diagrams are likely to get very complex.

ID	Stakeholder	Weight	Description	Use(s)
S1	Data-Sharing Companies	Low	Current source of data. (Main company in question is Citi Bike who supplies bike trip data). Can severely hinder the application if they decide to stop sharing their data. However, there is no indication that this is likely to happen in the near future.	Will be providing the app and users of it with the data required to load into the application for its full functionality.
S2	Retailers	High	Main source of income (eventually). The companies we eventually give access to our analytic data for a profit. Includes shops, food stores, cafes etc.	Will be using the app to find useful business information, including: - Most travelled routes (subscribers and customers) - Proximity of their business to others (filterable) - Detailed traffic data in the area (age, gender, time) - Start and End points of cyclist's trips
S3	Wi-Fi Providers	High	Secondary source of income (eventually). The companies we eventually give access to our analytic data for a profit.	Will be using the app to find useful business information, including: - Most travelled routes (subscribers and customers) - Proximity of their business to others (filterable) - Detailed traffic data in the area (age, gender, time) - Start and End points of cyclist's trips
S4	Cyclists	Medium	Secondary source of bike trip data. Ideally would have large regular user base of cyclists to add variance to data (not just Citi Bike data). Are a main target audience of the app and an important stakeholder to consider, but are not a very large source of income.	Will use the app to find useful locations on their bike trip, including: - Nearest bike station - Nearest business of a filterable type - Nearest free Wi-Fi area, also filterable to a type Overall will use the app to make cycle travel and use of certain bike sharing companies easier
S5	City council	Low	The organisation in charge of the maintenance and running of the city. Our app could help optimise cycleways, and help them keep up to date with road maintenance etc. Could help the city councils image and relations with the cyclist demographic by taking part in the app.	Will use the app to find potential ways to improve their city infrastructure, including data about: - Most travelled cyclist routes - Traffic data (busy areas, average travel times, rush hours)

S6	Non-Cyclists	Low	Very low priority user. A source of potential publicity, to get user base up, word of mouth etc. App is not targeted towards them, but could potentially be in later versions.	Will use the app to find the nearest location of a free Wi-Fi area or certain shop on their current route, also more practically, where to find the nearest bike station so they can begin a cycle trip
S7	Programmers /Developers/ The team	Low	The people/group creating, developing and maintaining the app. The better the product, the better the grade they get, as well as making maintainability/expansion of the app easier.	Will be creating every piece of the project; reports, source code, documentation etc.
S8	Staff	High, special case	The university staff involved with the project, tutors, lecturers etc. Have influence on the application as they will know what will and won't work well. Are a very valuable source of advice and education on the project and the tools we will be using.	Will be giving feedback and advice on the application during development, also will be using the finished application to eventually grade it

Table 1 - Stakeholder Table

2.2 Requirements

2.2.1 Use cases

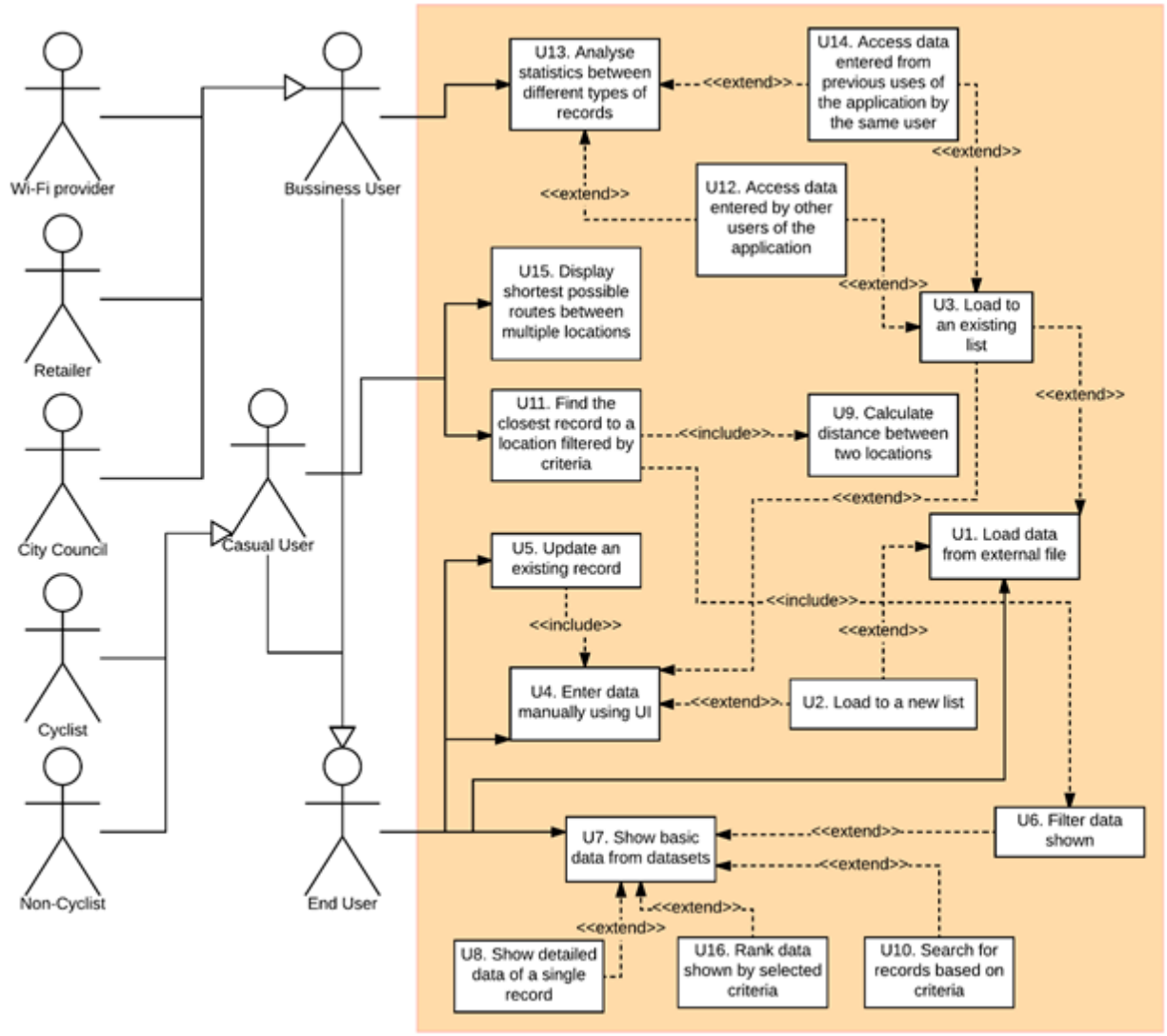

Figure 1 - Use case diagram

U1 - Load data from external file

Actor: Business users

Preconditions:

- The file will be stored in the correct format somewhere on the pc.

Basic flow:

1. The user will navigate to the data entry page.
2. The user will select to load data from a CSV.
3. The user will browse through the file structure on their pc to find the file to upload.
4. The user will select the file to upload

Alternative flow:

1. The user will navigate to the data entry page.
2. The user will select to load data from a CSV.
3. The user will browse through the file structure on their pc to find the file to upload.

4. The user cannot find the file to upload
5. The user will cancel the upload.

Exceptional flow:

1. The user will navigate to the data entry page.
2. The user will select to load data from a CSV.
3. The user will browse through the file structure on their pc to find the file to upload.
4. The user will select the file to upload, however the file will be in the incorrect format.

Postconditions:

Basic flow:

- The program will upload the data to the application.
- The uploaded data will be available for viewing, analysis and manipulation on the application.

Alternative flow:

- The program will return to the data entry page
- No data will be uploaded to the application

Exceptional flow:

- The program will return to the data entry page
- No data will be uploaded to the application
- An appropriate error message will be displayed to the user

U2 - When loading data, load to a new list

Actor: Cyclists

Preconditions:

- The user has selected a file to upload.

Basic flow:

1. The user will select to load the file to a new list.
2. The user will enter the list's name.

Postconditions:

Basic flow:

- The list will be created with the relevant data.
- The user will be able to filter by this list wherever filtering data occurs.

U3 - When loading data, load to an existing list

Actor: Business users

Preconditions:

Basic flow:

- The user has selected a file to upload.
- The user already has the list of records on their application that they wish to upload to.

Alternative flow:

- The user has selected a file to upload.

- The online database has a list of records they wish to upload to

Basic flow:

1. The user will select load to an existing list.
2. The user will select the list to which the data is to be added.

Alternative flow:

1. The user will select load to an existing list.
2. The user will select the list to which the data is to be added from the online database.

Postconditions:

Basic flow:

- The data will be added to the specified list.
- When filtering data by this list, the data uploaded will now be visible as well.

Alternative flow:

- The data will be added to the specified list.
- When filtering data by this list, the data uploaded will now be visible as well.
- The data will be available to other users through the online database

U4 - Manually enter data using UI

Actor: Cyclists

Preconditions:

- The user knows the details of the data to be entered.
- The user has decided whether they want their data to be used externally or not.

Basic flow:

1. The user will select to manually load some data.
2. The user will select the type of data they wish to upload.
3. The user will complete a form with the relevant details required.
4. The user will submit the form for uploading.

Alternative flow:

1. The user will select to manually load some data.
2. The user will select the type of data they wish to upload.
3. The user will not complete all the required details on the form.
4. The user will submit the form for uploading.

Exceptional flow:

1. The user will select to manually load some data.
2. The user will select the type of data they wish to upload.
3. The user will complete details of the form with invalid input.
4. The user will submit the form for uploading.

There will be both compulsory sections and invalid inputs for various sections.

Postconditions:

Basic flow:

- The record will be uploaded to the application and potentially to an external database.

- The uploaded data will be available for viewing, analysis and manipulation on the application.

Alternative flow:

- The program will display an error message specifying what required details were left blank.
- No data will be uploaded.

Exceptional flow:

- The program will display an error message specifying what details were in the incorrect format.
- No data will be uploaded.

U5 - Update an existing record

Actor: Cyclists

Preconditions:

Basic flow:

- There is some inaccurate data on the application
- The user knows the accurate details of the record to be updated.

Alternative flow:

- There is some inaccurate data on the application
- The inaccurate data is stored in an online database
- The user knows the accurate details of the record to be updated.

Basic flow:

1. The user will select to update a specific record.
2. The user will make edits to a form with the record's details.
3. The user will submit the updated record for uploading.

Alternative flow:

1. The user will select to update a specific record (stored in an online database).
2. The user will make edits to a form with the record's details.
3. The user will submit the updated record for uploading.

Exceptional flow:

1. The user will select to update a specific record.
2. The user will make invalid edits to a form with the record's details.
3. The user will submit the form for uploading.

Postconditions:

Basic and alternative flow:

- The details of the record will be updated.
- The old details are replaced by the new details and will be unrecoverable.

Exceptional flow:

- The program will display an error message specifying what edited details were invalid.
- The record will not be updated.

U6 - Filter data shown

Actor: All

Preconditions:

Basic flow:

- There is data on the application which can be filtered by some detail.

Alternative flow:

- The data on the application returns no results when filtered by specific criteria.

Basic and alternative flow:

1. The user will choose to filter the data.
2. The user will enter the criteria by which the want to filter the data.
3. The user will repeat step 2 for all needed criteria.

Postconditions:

Basic and alternative flow:

- The displayed data will only be data that meets the criteria of the filter.
- No other data will be visible.

Alternative flow:

- No data will be displayed
- A message will be visible saying that no records matching that criteria could be found

U7 - Show basic data from datasets

Actor: Casual users

Preconditions:

Basic flow:

- There are data sets to display.
- The records in these datasets contain basic data.

Alternative flow:

- There is no data in the datasets to display

Basic flow:

1. The user will select to view multiple records at the same time.
2. The user will input the types of records to view.

Alternative flow:

1. The user will select to view multiple records at the same time.
2. The user will input the types of records to view.

Postconditions:

Basic flow:

- Basic details of many records will be on display to the user.

Alternative flow:

- A message will be displayed to inform the user that no data is available.

U8 - Show detailed data of a single record

Actor: Casual users

Preconditions:

Basic flow:

- The record must have detailed data stored by the application.
- Multiple records will be on display to the user.

Alternative flow:

- Multiple record details will be stored in the application
- A single record will be displayed to the user

Basic flow:

1. The user will select which record they wish to specifically view.

Alternative flow:

1. The user will select to load the next or previous record

Postconditions:

Basic and alternative flow:

- All the data pertaining to a single record will be on display to the user.

U9 - Calculate distance between two locations

Actor: Casual users

Preconditions:

- The latitude and longitude of these locations must be known by the application.

Basic flow:

1. The user will select two locations
2. The user will select to calculate the distance between the locations.

Exceptional flow:

1. The user will attempt to select 0, 1, 3 etc. locations.

Postconditions:

Basic flow:

- The user will be able to see the distance between the two locations they selected.

Exceptional flow:

- An appropriate error message will be displayed.

U10 - Search for records based on criteria

Actor: Casual users

Preconditions:

Basic and alternative flow:

- The application must have data stored.
- The data must have details for which the user wants to search for.

Exceptional flow:

- The application must have data stored
- None of the data stored matches the required search criteria

Basic flow and exceptional flow:

1. The user will enter the criteria by which they which to search the current dataset.
2. The user will submit their search

Alternative flow:

1. The user will enter the criteria by which they which to search the current dataset.
2. The user will submit their search.
3. The user will choose to filter or rank their search.
4. If no results may be found that match the search criteria.

Postconditions:

Basic flow and alternative flow:

- All records that match the selected criteria will be on display to the user.

Exceptional flow:

- An appropriate error message will be displayed

U11 - Find the closest record to a location filtered by criteria

Actor: Casual users

Preconditions:

Basic flow:

- The application must have stored locations which can be filtered by criteria.
- The application must know the latitude and longitude of the stored locations.

Exceptional flow:

- The application must have stored locations which can be filtered by criteria.
- The application must know the latitude and longitude of the stored locations.
- None of the stored locations fit the required criteria

Basic flow and alternative flow:

1. The user will select a starting location.
2. The user will select the criteria of the location they wish to find.

Postconditions:

Basic flow:

- The record of the closest location that matches the criteria will be displayed to the user.

Alternative flow:

- An appropriate error message will be displayed.

U12 - Access data entered from previous uses of the application by the same user

Actor: All

Preconditions:

Basic flow:

- The user must have previously entered data which is now stored in the application.

Exceptional flow:

- The data stored inside the application was edited between instances

Basic flow and exceptional flow:

1. The user will open the application after using a previous instance of the application.
2. The user will access some data retrieval function.

Postconditions:

Basic flow:

- The data will be available for many uses such as viewing, analysis or manipulation.

Alternative flow:

- If editing is detected an error message will be displayed, otherwise inconsistencies may be created in the data.

U13 - Analyse relevant statistics between different types of records

Actor: Business users

Preconditions:

- Multiple datasets must be stored on the application.
- The data sets must contain details which the application can analyse and graphically display.

Basic flow:

1. The user will select the datasets that they wish to compare.
2. The user will select the specific statistical data they want to analyse.

Alternative flow:

1. The user will select the datasets that they wish to compare.
2. The user will select the specific statistical data they want to analyse.
3. The user will select to print or otherwise export the analysis.

Postconditions:

Basic flow:

- The program will then compute the statistics and provide them for analysis to the user in various forms, including, but not limited to, graphs and tables.

Alternative flow:

- The analysis will be printed or exported out of the application.

U14 - Access data entered by other users of the application

Actor: All

Preconditions:

Basic flow:

- Data entered by other users must be stored externally from their application.
- The application must have an internet connection.

Alternative flow:

- Data entered by other users must be stored externally from their application.
- The application does not have an internet connection.

Basic flow and alternative flow:

1. The user will open the application.
2. The user will use a feature that accesses stored data.

Postconditions:

Basic flow:

- The data will be available for many uses such as viewing, analysis or manipulation.

Alternative flow:

- The user will be unable to access the data and will be notified accordingly.

U15 - Display shortest possible routes between multiple locations

Actor: Cyclists

Preconditions:

Basic flow:

- The application must know the latitude and longitude of the locations required.
- There is a shortest possible route that can be found between these locations.

Exceptional flow:

- The application must know the latitude and longitude of the locations required.
- There is a shortest possible route that can be found between these locations.

Basic flow and exceptional flow:

1. The user will select the locations to visit.
2. The user will enter the order in which to visit these locations.
3. The user will select to display the shortest possible route between the locations.

Postconditions:

Basic flow:

- The route will be displayed to the user, potentially with directions and time approximations.

Exceptional flow:

- An error message will be displayed stating that no route could be found.

U16 - Rank data shown by selected criteria

Actor: Casual users

Preconditions:

- The application must have data stored.
- The data must have some detail which can be ordered by the application.

Basic flow:

1. The user will select the criteria they wish to rank the records they are viewing on.

Alternative flow:

1. The user will attempt to select criteria which the records cannot be ranked by.

Postconditions:

Basic flow:

- The data shown will be ordered by the criteria selected.

Alternative flow:

- No change will be visible.

2.2.2 Functional requirements

The following functional requirements table is sorted by priority. Any requirement with very high priority is essential for the project as they are the requirements of the feature packages outlined in the project description.

ID	Description	Stakeholder(s)	Priority	Use Case(s)
F1	The product must be able to add bike trip data from manually entered input	S4, S8	Very High	U2, U3, U4
F2	The product must be able to add retailer location data from manually entered input	S2, S4, S6, S8	Very High	U2, U3, U4
F3	The product must be able to add Wi-Fi location data from manually entered input	S3, S4, S6, S8	Very High	U2, U3, U4
F4	The product must be able to calculate the distance between the start and end locations of a bike trip	S4, S8	Very High	U9
F5	The product must be able to display all the data pertaining to a single retailer record	S4, S6, S8	Very High	U8
F6	The product must be able to display all the data pertaining to a single trip record	S4, S6, S8	Very High	U8
F7	The product must be able to display all the data pertaining to a single Wi-Fi location record	S4, S6, S8	Very High	U8
F8	The product must be able to display important details of many retailer records at once	S4, S6, S8	Very High	U7
F9	The product must be able to display important details of many trip records at once	S4, S6, S8	Very High	U7
F10	The product must be able to display important details of many Wi-Fi records at once	S4, S6, S8	Very High	U7
F11	The product must be able to filter the bike trip records displayed by gender	S2, S3, S8	Very High	U6, U7
F12	The product must be able to filter the bike trip records displayed by stations	S4, S8	Very High	U6, U7
F13	The product must be able to filter the bike trip records displayed by userpass	S4, S8	Very High	U6, U7
F14	The product must be able to filter the retailer records displayed by secondary type	S4, S6, S8	Very High	U6, U7
F15	The product must be able to filter the retailer records displayed by street	S4, S6, S8	Very High	U6, U7
F16	The product must be able to filter the retailer records displayed by zip code	S4, S5, S6, S8	Very High	U6, U7
F17	The product must be able to filter the Wi-Fi records displayed by borough	S4, S5, S6, S8	Very High	U6, U7
F18	The product must be able to filter the Wi-Fi records displayed by provider	S4, S6, S8	Very High	U6, U7
F19	The product must be able to filter the Wi-Fi records displayed by type	S4, S6, S8	Very High	U6, U7
F20	The product must be able to find the closest retailer to a specified bike route	S4, S8	Very High	U9, U11
F21	The product must be able to find the closest Wi-Fi location to a specified bike route	S4, S8	Very High	U9, U11
F21	The product must be able to find the closest Wi-Fi location to a specified retailer	S4, S6, S8	Very High	U9, U11
F23	The product must be able to load data from a file stored inside the application	S2, S3, S4, S5, S6, S8	Very High	U12
F24	The product must be able to rank the bike trip records displayed by distance	S4, S8	Very High	U16
F25	The product must be able to retailer data to an existing list	S2, S3, S8	Very High	U1, U3, U4
F26	The product must be able to search for bike trips based on station	S4, S8	Very High	U7, U10
F27	The product must be able to search for retailers based on name	S4, S6, S8	Very High	U7, U10
F28	The product must be able to search for Wi-Fi locations based on provider	S4, S6, S8	Very High	U7, U10
F29	The product must be able to store retailer data in a file inside the application	S2, S3, S4, S5, S6, S8	Very High	U2, U3, U12
F30	The product must be able to store trip data in a file inside the application	S2, S3, S4, S5, S6, S8	Very High	U2, U3, U12
F31	The product must be able to store Wi-Fi location data in a file inside the application	S2, S3, S4, S5, S6, S8	Very High	U2, U3, U12
F32	The product must be able to update the details of an existing bike trip record	S2, S3, S4, S6, S8	Very High	U4, U5
F33	The product must be able to update the details of an existing retailer record	S2, S3, S4, S6, S8	Very High	U4, U5
F34	The product must be able to update the details of an existing Wi-Fi record	S2, S3, S4, S6, S8	Very High	U4, U5

F35	The product must be able to upload bike trip data from a file outside the application in a specific format	S2, S3, S8	Very High	U1, U2, U3
F36	The product must be able to upload retailer data to a new list	S4, S6, S8	Very High	U1, U2, U4
F37	The product must be able to upload retailer location data from a file outside the application in a specific format	S2, S8	Very High	U1, U2, U3
F38	The product must be able to upload trip data to a new list	S4, S6, S8	Very High	U1, U2, U4
F39	The product must be able to upload trip data to an existing list	S2, S3, S8	Very High	U1, U3, U4
F40	The product must be able to upload Wi-Fi location data from a file outside the application in a specific format	S3, S8	Very High	U1, U2, U3
F41	The product must be able to upload Wi-Fi location data to a new list	S4, S6, S8	Very High	U1, U2, U4
F42	The product must be able to upload Wi-Fi location data to an existing list	S2, S3, S8	Very High	U1, U3, U4
F43	The product must allow the user to analyse cyclists' age with in regard to bike traffic	S2	Medium	U13
F44	The product must allow the user to analyse cyclists' gender with in regard to bike traffic	S2	Medium	U13
F45	The product must allow the user to analyse cyclists' age in regard to stations visited	S8	High	U13
F46	The product must be able to find the most popular stations for cyclists to visit	S2, S3, S5	High	U13
F47	The product must be able to rank the bike trip records displayed by time taken	S4	High	U16
F48	The product must allow the user to analyse cyclists' userpass with in regard to bike traffic	S2	Medium	U13
F49	The product must allow the user to analyse the most popular locations for cyclists to visit	S2, S3, S5	Medium	U13
F50	The product must be able to display a route between any two locations	S4, S6	Medium	U15
F51	The product must be able to filter the retailer records displayed by borough	S4, S6	Medium	U6, U7
F52	The product must be able to load retailer data from outside the local PC	S4, S6	Medium	U14
F53	The product must be able to load Wi-Fi data from outside the local PC	S4, S6	Medium	U14
F54	The product must be able to search for Wi-Fi locations based on type	S4, S6	Medium	U7, U10
F55	The product must be able to search the retailer records displayed by primary type	S4, S6	Medium	U7, U10
F56	The product must allow the user to analyse the frequency of retailers added to routes with reference to Wi-Fi locations added to routes	S2, S3	Low	U13
F57	The product must be able to display a route between three or more locations	S4, S6	Low	U15
F58	The product must be able to filter the trips displayed based on date	S2, S3, S5	Low	U6, U7
F59	The product must be able to load trip data from outside the local PC	S2, S3, S5	Low	U2, U3, U14
F60	The product must be able to search for Wi-Fi locations based on name	S4, S6	Low	U7, U10
F61	The product must be able to store retailer data to outside the local PC	S4, S6	Low	U2, U3, U14
F62	The product must be able to store trip data to outside the local PC	S2, S3, S5	Low	U14
F63	The product must be able to store Wi-Fi data to outside the local PC	S4, S6	Low	U2, U3, U14
F64	The product must be able to update the details of multiple records of any type at once	S2, S3	Low	U5

Table 2 - Functional Requirements Table

2.2.3 Quality requirements

The quality of the application was aimed towards achieving the quality requirements in Table 3. It is important however to note that these quality requirements do not fully cover all quality aspects of the application. This is due to the difficult nature of creating quality requirements that can be measured in a meaningful and accurate way. Therefore, a portion of quality aspects related to the application were testing during the acceptance test phase. The main quality aspects covered in the acceptance tests are bug related. This is because there is no way realistic way to prove that bugs are non-existent in an application of this size and so an acceptance test relating to the consistency of the application and prevalence of bugs was added shown in
Table 5. This allows the end user to be the one who is testing the application, making the subjectivity of testing for such no longer a major issue. This ultimately leads to a product that is guaranteed to be suitable in this respect of quality for the end user as they have been the one to pass or fail it.

ID	Requirement	Stakeholder ID	Priority
Q1	The app must be completed to a shippable standard by the 10th October 2017	S7, S8	High
Q2	The app should be able to handle large data I/O running in linear time relative to the amount of the data	S2, S3, S5	High
Q3	All end user functions of the app should be accessible (where appropriate) through a GUI	S2, S3, S4, S5, S6	High
Q4	The app should have a response time on any GUI interaction of less than 1 second unless prompted/indicated otherwise (e.g. Loading screen)	S2, S3, S4, S5, S6	High
Q5	The app should be a stand-alone application with minimal external dependencies (E.g. external libraries, non-local software, etc.)	S2, S3, S7	Medium
Q6	There should be no licensing issues with the product	S2, S3, S5, S7	Medium
Q7	The app should keep user information anonymous when stored/transferred online unless specified otherwise	S4, S6	Low
Q8	The source code should follow all predefined coding conventions	S7, S8	Low

Table 3 - Quality Requirements Table

2.2.4 Key drivers

The following keys for Table 4 have been used to simplify the layout. **W** represents “Weighted” and **UW** represents “Unweighted”.

Stakeholders	Weight	Analytic Accuracy		Speed		Usability		Graphics	
		UW	W	UW	W	UW	W	UW	W
S2	0.35	45	15.75	25	8.75	15	5.25	15	5.25
S3	0.30	45	13.5	25	7.5	15	4.5	15	4.5
S4	0.20	20	4	30	6	30	6	20	4
S5	0.10	40	4	15	1.5	25	2.5	20	2
S6	0.05	5	0.25	30	1.5	35	1.75	30	1.5
Total Weighting	1	37.5		25.25		20		17.25	

Table 4 - Key Drivers

Analytic accuracy:
The correctness of analytics available to the user.

Speed:
The speed of the application. This encompasses both the speed of loading data and analytics, as well as planning a route and navigating the application screens.

Usability:
The ability for users to navigate the application easily and achieve their desired outcome receiving help where needed.

Graphics:
The ability to present information in an aesthetically pleasing graphical format, such as the Google Map API or graphical representations of analytics (e.g. Pie charts, bar graphs).

The Key Drivers table above focuses on the four qualities that are most relevant and critical to our stakeholders. By assigning overall weightings to each stakeholder and then estimating the value each holds towards the four key drivers, a clearer objective overview of what qualities are most critical to the application was achieved. Table 4 shows that analytic accuracy has the highest overall weighting between stakeholders. This is because our high priority stakeholders **S2** and **S3** will both be interested mainly in the analytics and need this to be accurate as they could be basing costly business decisions on this data. This also reveals the need for a range of different analytic functions which drove the analytics part of the application to be more in depth containing graphs and map functionality. It also pushed for more testing of these analytic functions as accuracy is, as stated before, very crucial.

Speed was found to be the second most important key driver. This is partially related to the analytic side as mentioned above since typically large quantities of data need to be processed to find useful analytic data. This means that good speed would be preferable to aid in making this process efficient and make the application worth using. It also is driven by **S4**, and **S6** as generally slow applications are not popular with users who are typically less technologically inclined and using the application for simpler purposes.

Lastly, usability and graphics were most central to **S4**, **S5**, and **S6** as these stakeholders are more interested in an application that performs tasks fast, with a user-friendly environment that is aesthetically pleasing to the eye. It is not as key to **S2**, and **S3** however, due to being more concerned with development time being spent on the analytic functionality of the application as they will be most interested in this. **S2** and **S3** are less bothered by having a slightly harder to use application with less fancy graphics then more casual users like **S4**, **S6**.

Note: Table 4 excluded **S8** as they will not be an end user of our application so including them would skew our data and may result in us making poor design decision for the actual users of the application. Also, **S1** was excluded as they are a supplier of data more than an end user of the application and mostly irrelevant in the key drivers.

3 Acceptance tests

3.1 Acceptance test table

ID	Description	Responsibility	Acceptance Criteria	Criticality	Use Case
A1	Given a user loads data from an external file, that data will then be able to be viewed, filtered, searched etc. Within the GUI	Casual user, Business user	When the data is loaded and fully navigable/intractable	High	Load data from external file
A2	Given a user loads an external file to a new list, a new list will be created containing all the data from the selected external file.	Casual user, Business user	When a new list containing the external file, data is created it can be queried as a list with no previous data	High	When loading a file, load to a new list
A3	Given a user loads an external file to an existing list, the existing list will have the new data sorted into it without the existing data being disrupted.	Casual user, Business user	When the existing list has had the external file, data included and sorted with previous data being left untouched.	High	When loading a file, load to an existing list
A4	Given a user is in the raw data viewer and manually enters data into the list, the data will be added to the specified list being without the existing data being disrupted.	Business user	When the manually entered data is added to the specified list and sorted with previous data being left untouched.	High	Manually enter data using UI
A5	Given that a user has updated a record, the selected record will contain the updated data and the be sorted into the list.	Business user	When the updated data is included in the record and sorted into the list with non-updated data being left untouched.	High	Update an existing record
A6	Given that a user is in the raw data viewer or in the map view, and filters data to show only a small category of the data, only that category will be displayed.	Casual user, Business user	When the data displayed shows only those that match the filtering criteria.	High	Filter data shown
A7	Given that a user has selected to view basic data from datasets, the appropriate data will be shown the GUI and the user will be able to interact with it.	Business user	When the data is displayed on screen for the user to see and can be updated/changed.	High	Show basic data from datasets.
A8	Given that a user has selected to view a single record, that record will be shown with all detail fields being displayed.	Casual user, Business user	When a single record is displayed on screen with every data point shown.	Medium	Show detailed data of a single record

A9	Given the user has selected two points on the route selection screen, a route between the two points will be shown with distance between them also being shown.	Casual user	When a route between the two selected routes is on display within a google maps interface, with the distance between them.	Low	Calculate distance between two locations
A10	Given a user has searched for data using certain criteria in the data viewing section, only the data relevant to that criteria will be displayed.	Casual user, Business user	When the data shown after a search all matches the user inputted search criteria	Medium	Search for records based on criteria
A11	Given a user has searched for data using certain criteria and filtered the search space, only data relevant to the search criteria and filter will be shown.	Casual user, Business user	When the data shown after searching and filtering all matches both the search criteria and filtering criteria	Medium	Search for records based on criteria
A12	Given that a location has been selected and the user has chosen to find the closest point, the correct closest point to the chosen location based on user specified criteria will be shown on the raw data viewer/map.	Casual user	When the correct closest record to the chosen location is displayed and matches the search criteria, and no other records are shown.	Low	Find the closest record to a location filtered by criteria
A13	Given that a user has selected to view previous data, the previous data entered by the user will be shown and can be searched through and filtered.	Casual user	When the data entered by the user previously is shown and on display and can be interacted with.	Low	Access data entered from previous uses of the application by the same user
A14	Given that a user has selected to view statistics of records, an option to filter the data by relevant statistics will appear and the statistics will be on display for the user to analyse.	Business user	When many analytics for different types of data can be viewed graphically and/or textually.	High	Analyse relevant statistics between different types of records
A15	Given that a user has chosen to view a set of data from another user, that data will be imported to the user’s screen and can be filtered through, changed, just like it was the user's own data.	Casual user, Business user	When the other user's data has successfully been imported, and is displayed onscreen and can be interacted with.	Medium	Access data entered by other users of the application.

A16	Given that a user has chosen two locations in the map view, the application will display the shortest route from the starting location to the end location.	Casual user	When the shortest route is on display and correctly aligned with the streets/paths, and any other path taken to get from location A to location B is either as equally long or longer.	Low	Display shortest possible routes between multiple locations.
A17	Given that a user has selected to view the data based on some criteria and selected to view them ranked, the data will be displayed from "high to low", "long to short", "latest to earliest" etc...	Casual user, Business user	When the data is correctly ranked by the selected criteria and on display.	High	Rank data shown by selected criteria.
A18	Given that a user is using the program, the likelihood of encountering a bug is low	Casual user, Business user	In the testing of the program under expected use no bugs are found.	High	General Use case

Table 5 - Acceptance Test Table

Acceptance tests are where developer or client will perform a set of actions or commands in the program to see if the actual output is the same as the expected output. They are a form of black box testing so are written with no access to the source code. These are extremely helpful as they help the developer to know if the program is working as intended as well as assuring the client that the application is working how they expected.

Our acceptance tests are based on the various use cases we have for our project. These tests are mainly used by the end user to test the end product; however, they can also be used by us, the development team, during the coding phase to make sure our program is on track and working correctly. They are there to visualise what our program should be outputting so we can easily see where our program needs fixing.

As shown above in
Table 5 outlining all the acceptance tests relevant to our program. Each of the tests has been given a priority based on the related use case and which stakeholders they affect the most. During the coding phase, we will be more heavily focused on the high priority acceptance tests as these are the most important. They are categorized as high priority as they are going to be in the end product. The tests that are of low priority however are for extensions and will be implemented into the product if time permits.

The acceptance test table is a living table that can and most likely will, change throughout product development. If or when we identify new stakeholders, or more use cases that we would like to have, the acceptance test table must be updated to accommodate for that.

5 Deployment model

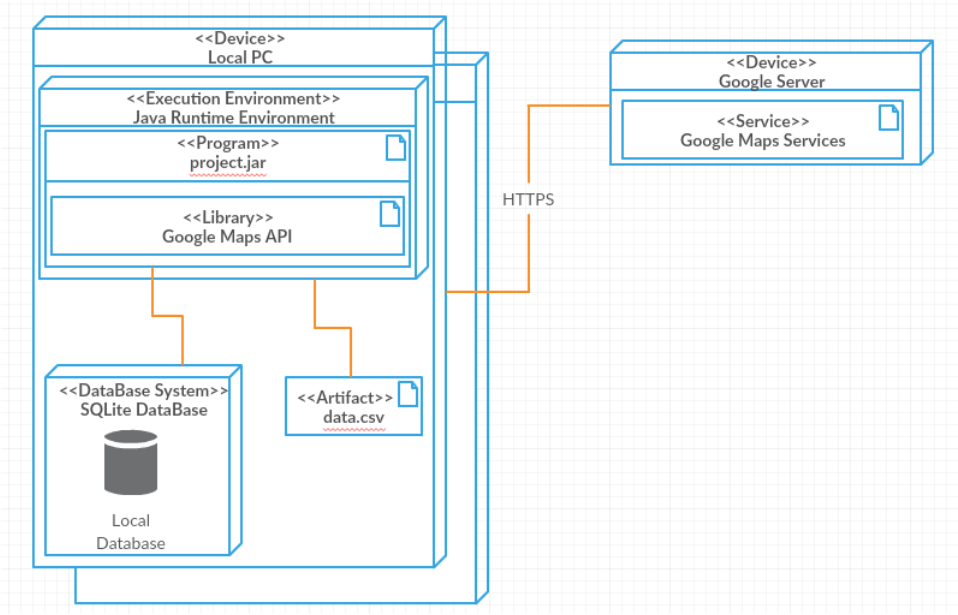

Figure 2 - Deployment model diagram

Our application is designed to run locally on any number of PCs in the Java Runtime Environment, this allows it to be run on many different operating systems with one file. The jar file shipped will include the google maps API as this is vital for displaying data on a map. To display the maps the application will need access to the internet such that it can access Google Maps Services. This connection will be Through an HTTPS wrapper as it needs all the information to be sent and received so the data shown is correct. As the application, will be running on a PC there are obvious limitations in terms of processing power and speed however since the application will do analysis on a given set of data it allows for users with more powerful machines to complete analysis on larger datasets. This is a helpful feature for business users such as large companies are more likely to have better hardware to run the application.

Our application will ship with a local SQLite database which can be added to by the user. This database will come prepopulated with data. This is useful for users of the application as they do not have to gather this data themselves. The user can add data individually or batches through a csv on the machine.

6 UML class diagrams

The class diagrams shown in the figures below are a high-level representation of the interactions and relationships between major classes of the application and includes most of the core attributes and methods. It is important to note that these diagrams are not fully detailed with all attributes and methods included. For example, all figures getters and setters have been removed. Figure 3 shows the overall package structure of the application. The application is fully packaged so that each sub-package has a one main functionality.

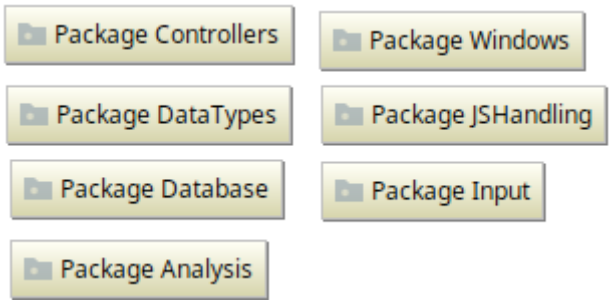

Figure 3 - Overall package structure for the application

The Controllers package and Windows package contain further sub-packages representing the help window related classes in one sub-package and the main window related classes in the other sub-package as shown in the figure below.

Figure 4 - Controllers package structure

These two packages have a similar structure and the MainWindowControllers package is shown in full in Figure 6 to show the basic structure of these packages. Each of these controller classes has a model class to match it in the Windows package. The Windows package is not shown in a figure as most of the key ideas are in the controller counterparts. We then have the Analysis package shown in Figure 5 which contains classes related to the trip analysis side of the application.

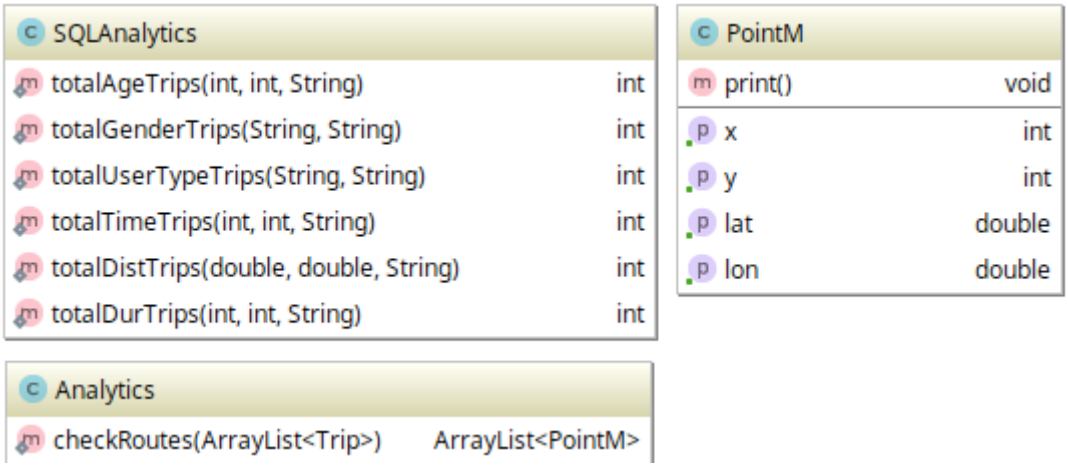

Figure 5 - Analysis package

Next, we have the Database package shown in Figure 7 which contains all database related classes which retrieve, update, and handle all database related functionality.

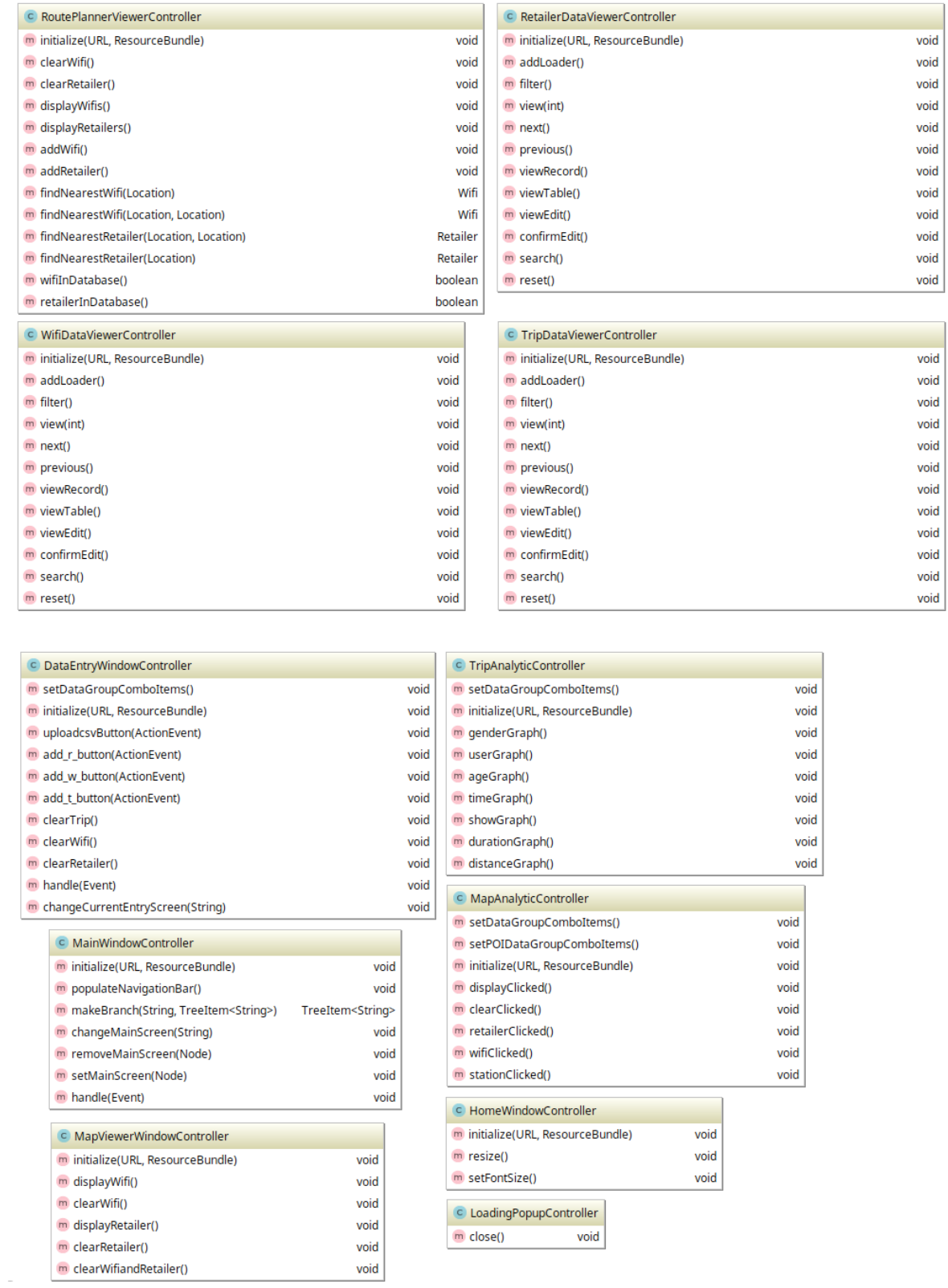

Figure 6 - MainWindowControllers package

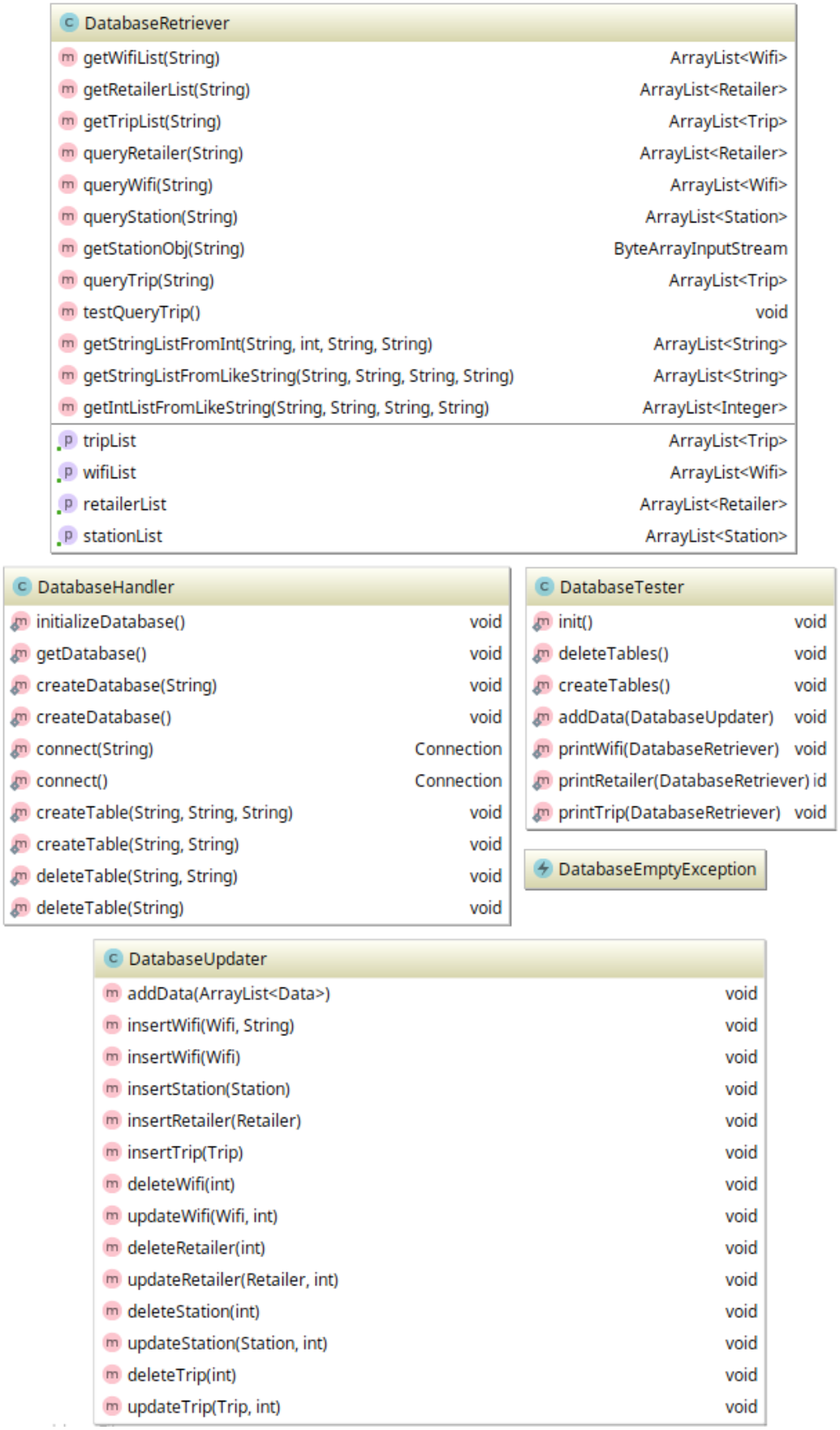

Figure 7 - Database package

Then, we have the InputHandler package which oversees handling all the data input of the application shown in Figure 8.

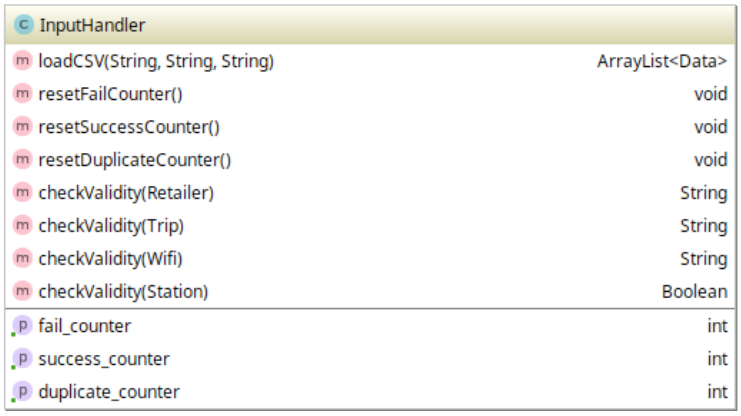

Figure 8 - Input package

We also have a JSHandling package show in Figure 9 which contains classes that interact with JavaScript. This separation was to make it clear where JavaScript is used for the application and make it easier to find errors relating to JavaScript errors as only one package contains these classes.

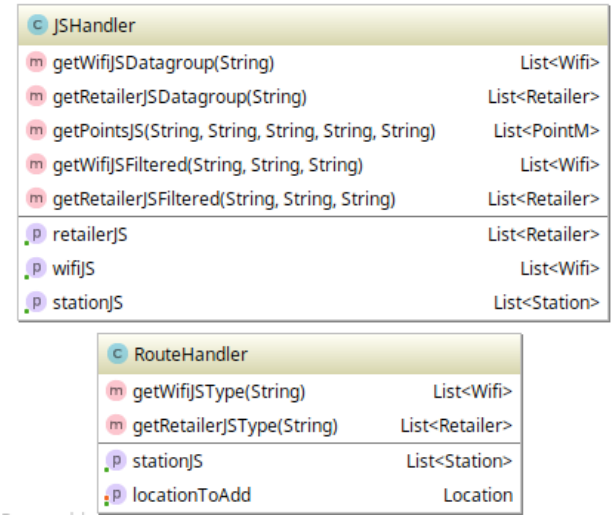

Figure 9 - JSHandling package

Then lastly, there is the DataTypes package shown in Figure 10 which contains all the classes related to the different data types used in the application. Figure 11 shows the classes in full for this package.

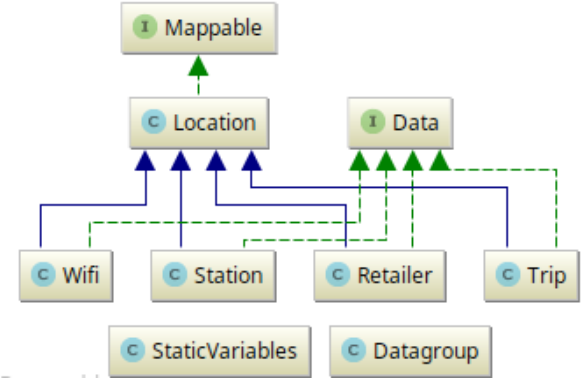

Figure 10 - DataTypes package structure

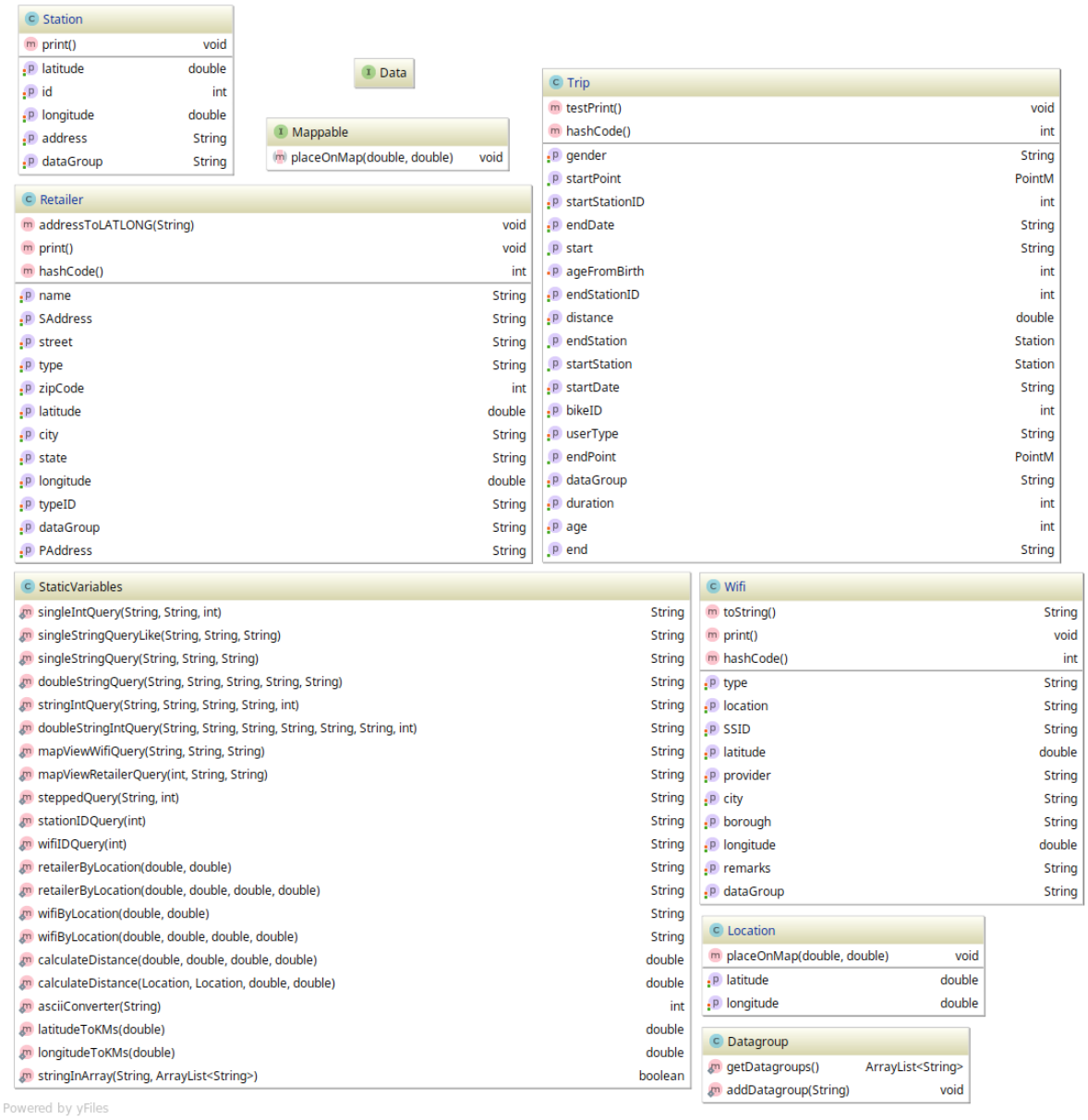

Figure 11 - DataTypes package

It is important to note a few key design principles used in the DataTypes package. Firstly, the use of the Data interface. This was done to generalise Data objects (Retailer, Wi-Fi, Station, and Trip objects) so they can be handled the same. This allows flexibility in our methods that use and manipulate Data objects such as those used in the GUI related classes. Secondly, there is a Mappable interface which similarly is used to generalise Mappable objects (Not necessarily Data objects) so that it was easier to implement methods for plotting these. Thirdly, the Location class is used to separate the Mappable interface to allow flexibility in possible future extensions of the application where there may be classes not directly related to Data.

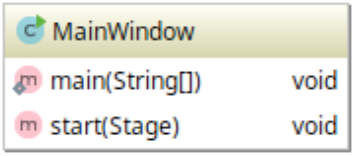

Figure 12 - The MainWindow class

The Windows package contains a variety of custom JavaFX classes (excluding the MainWindow class shown in Figure 12) which have a constructor contained inside of them to create the custom object. Having custom JavaFX objects representing each window of our program allows a separation between the MainWindow functionality and everything else. The MainWindow class creates and keeps track of each window in the application and which one is currently being shown. It therefore does not need to worry about any details of how each individual window works. Designing the application in this way makes it very easy to add/remove different windows painlessly and means that windows can be worked on separate from the main application until suitably functional, therefore removing error susceptibility, and allowing easier scaling of the application. MainWindow is the starting point of the application and is the main application container. It is important to note that one window has one controller to follow a MVC (Model-View-Controller) approach. The View is also separated by having FXML files to represent the GUI layout. Following this MVC approach allows easier to follow code and an easy to manage system. This in turn means that future additions such as more analytic functionality possibly requested by, for example Retailers, can be painlessly added with minimal effort, saving development time and money.

7 Risk assessment

ID	Description	Impact	Likelihood	Responsibility	Consequences	Prevention
R1	Miscommunica-tion within the team	2	2	Everyone	One person or group may focus on the wrong task for a period of time.	Make it clear who is working on what by assigning tasks on Trello. Schedule meetings multiple times a week where we check what everyone is doing allowing all members to be on the same page.
R2	Loss of Google Doc Data	3	1	Connor, Everyone	Loss of work and ideas.	Connor to back-up the google drive folder every Sunday during meetings. Rest of team to remind him and do it if he cannot.
R3	Git and CI environment inaccessible	3	1	Everyone	Coding work cannot be completed whilst it is down.	Pull latest versions and in the case of total loss push the latest version between all team members.
R4	Code not understandable	1	2	Everyone	Members may not be able to understand others' code and will make understanding it very hard.	All team members have agreed upon a coding convention that will be upheld.
R5	Change in team Structure	2	1	Everyone	If someone leaves the course others will have to pick-up a larger workload.	Make sure we do not over extend our plans and have a development schedule that gives us some extra time, should something happen.
R6	Miscalculation of difficulty of tasks	2	2	Everyone	If a task is given a much smaller time frame due to a miscalculation of difficulty it could lead to falling behind schedule.	Allow more time than expected for tasks, especially those no-one is familiar with.
R7	Application strays from Stakeholder Expectations	2	1	Everyone, Stakeholders (where applicable)	The final application would not be what was intended and would not be of use to the stakeholders.	Talk with the staff every week to ensure that we are keeping on track. Make sure all members are on the same page when it comes the expected end application.

R8	Broken Analysis	3	1	Everyone	The analysis features that are broken will be of no use to the user and the value of our product will be significantly diminished.	Create JUnit tests to run for all analyses and run them whenever a change is made to the analysis algorithms and ensure that they pass.
R9	Team Members being sick	1	3	Sick Team Member, Everyone	A team member may not be able to complete the work they had planned or may miss contact sessions.	Sick member will advise others as early as possible. The rest of the team will make sure any vital jobs they had to do are redistributed. If they miss a session give them a rundown of what they missed. If they miss a lab the sick person will catch up when they are well, with help from other members where applicable.
R10	Loss of Work	1	2	Everyone	Forgetting to save work often may lead to some work having to be redone.	All members will save often when working on code so that in the event of a crash or similar, they amount of work lost will be minimal.
R11	No Internet Access	2	1	Everyone	Less time to complete required tasks and no access to working directories.	Should a member lose internet connectivity they should inform the rest of the team, and work around the problem. Such as working on campus where internet is available
R12	Running out of time for a Deliverable	3	1	Everyone	Large negative effect on our grade for the deliverable.	Have a code/writing freeze several days before the hand-in date where nothing new is added, rather the time is spent reviewing and making sure the code/report is written properly and achieves what is expected.
R13	Merge Failures in Git	2	2	Everyone	If merge conflicts are not corrected properly it may break parts of our application.	Make sure to check merge conflicts and consult other team members if a change is not clear. Should the program break after deciding on merge conflict resolutions, go back to a working state and select other resolutions.
R14	Handing in Deliverable Late	3	1	Available Members	If the deliverable is not handed in on time we will receive a grade penalty	Have a meeting the day the deliverable is due, several hours before the due time, with whomever on the team is available and make sure the correct copy of the deliverable is delivered.

					and shows unprofessional practice.	
R15	Data sharing companies stop sharing their data	3	1	Everyone	The developers and the users will no longer have access to data critical for the functionality of the product.	Download all types of data early and store them locally so that the application can still at least be developed and tested with the appropriate data.

Table 6 - Risk Assessment Table

Analysis of Risks:
The identified risks in Table 6 above mainly fall into several main categories. These are human error, technical error and unavoidable circumstances. Technical error risks are often hard to prevent as many services that we are rely on we have no control over, such as google docs or the git server. As such there is little we can do to prevent events, like inaccessibility or loss of data. However, we can take active steps to ensure that the effects are lessened should something bad occur. This is as simple as backing up the work we do on our local machines so that losses are not catastrophic.

Human errors are another major source of risk as humans are not perfect and we are bound to make mistakes, however since there are many of us in the team we can help ask other members for help should we become stuck on something. Also, if someone sees a mistake they can inform the person who made it and suggest how it can be fixed.

Unavoidable circumstances are often quite rare but also impossible to avoid completely. By the second week of the project a situation already occurred where a member come down with sickness and missed out on several meetings. Risks like this cannot be completely removed, instead we have incorporated these occurring into our plans, allowing ourselves extra time to complete task. And we have focussed on actions that can be taken during and after the incident to reduce its impact on the project.

Whilst analysing risks for this project we have agreed that keeping good communication is key to not only reducing the chance of risk but mainly keeping the impact of any problem that occurs minimal, from this communication has been an important part of our team mantra.

8 Project Plan

The following three figures and tables were part of our project plan for Deliverable 2. As Deliverable 3 is our last deliverable there is naturally no project plan for beyond Deliverable 3. We have kept this section to display the planning procedures that were used in the implementation of our product. It should be noted, however, that the functional requirements have been considerably refactored since the plan was created. This means that the functional requirements in the task table do not reference the correct requirements in the table included in this report.

8.1 Task table

ID	Task	Functional requirements
T1	Make the list structure more intuitive for the user	F2, F18
T2	Remove the possibility of duplicate records with appropriate error messages	F1, F9 - F15, F30
T3	Format the GUI to correctly encompass all information for records	F26
T4	Implement extra search functionality	F49, F52, F59
T5	Implement loading bars whenever there is significant wait time for the application	F1, F13, F15, F32
T6	Increase the efficiency of uploading all data	F1, F13, F15
T7	Fully implement the route planner to include functionality which requires finding distances between routes and locations etc.	F29, F31, F33, F34, F60
T8	Fully implement the required analytic capabilities to include analysis of bike traffic.	F36, F38, F41 - F47, F57, F63
T9	Implement the online database with stored Wi-Fi and retailer records.	F39, F40, F55, F56, F61, F62

Table 7 - Task Table

8.2 Project calendar

Week	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday
9	23/09	24/09 Code freeze/last minute bug fixes	25/09	26/09 Lecture 9 **Deliverable 2**	27/09	28/09 Lab 17 Presentation	29/09 Lab 17
10	30/09	01/10 T1-T6 implemented	02/10	03/10 Lecture 10	04/10	05/10 Lab 19 COSC265 Test	06/10 Lab 18 T7, T8, T9 implemented
11	07/10	08/10 Code freeze/last minute bug fixes	09/10	10/10 Lecture 11 **Deliverable 3** **Demos**	11/10	12/10 Lab 21 **Demos**	13/10 Lab 22 **Demos**
12	14/10	15/10	16/10 COSC264 Test	17/10 Lecture 12	18/10	19/10 Lab 23	20/10 Lab 24

Table 8 - Project Calendar

8.4 Gantt Chart

Task Name	Assigned To
Minor changes, additions and features	
Make the list structure more intuitive for the user	Lachlan
Remove the possibility of duplicate records with appropriate error messages	Lachlan
Format the GUI to correctly encompass all information for records	Connor
Implement extra search functionality	Aidan
Implement loading bars whenever there is significant wait time for the application	Aidan
Increase the efficiency of uploading all data	Lachlan
Major features to be implemented	
Fully implement the route planner to include functionality which requires finding distances between routes and locations etc.	Josh
Fully implement the required analytic capabilities to include analysis of bike traffic.	Connor
Implement the online database with stored Wi-Fi and retailer records.	Morgan
User manual	Mitchell

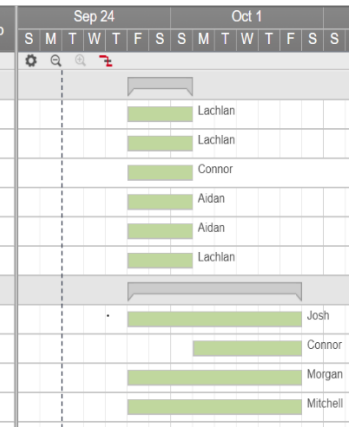

Figure 15 - Gantt Chart

9 Test protocol and testing procedures

9.1 Functional testing

Multiple generic object classes have no unit tests written for them. This is because these classes contain no complex or complicated functions that rely on anything else and there is really no way they can “break”, and there is no point at all testing their getters and setters. For example, Trip has tests because of the way it derives values from its constructor, where there is a possibility of the functions not working correctly. Also, there are multiple layers of “checks” that objects go through when they are created to ensure they are valid, these methods check individual parts of each object to ensure they are not over 100 characters long, just for example. Another layer being that it is often not possible for objects to be created with invalid data as it will often be caught by the multiple try catch blocks in place.

Another key area with little Junit tests written are the database classes. These types of classes are hard to test as what is in the database and how it is structured is constantly changing and it is difficult to know what the “right” output of their functions is. The class “Database Tester” has been the main way of testing the database in progress while we work on it. This is also expanded on later in the discussion.

There are no unit tests written for the GUI controller classes or any part of the GUI element of the app. This is because there is no viable way for us to efficiently write tests that check each aspect of the GUI as these tests would have to be constantly changing as we refactor our GUI and come up with new features. We did however consider “TestFX”, a JavaFX module that can be used to test controller classes. We decided against using it because learning to use it would likely take more time than it is worth, as well as because of advice from a tutor.

As a way around this issue, our controller classes have only code that is necessary for that controller inside it, with “helper code” being put elsewhere where it can be more easily tested. Also, we all follow a test procedure that we use when changing/writing code for the GUI part of our app. This means that when something is changed, we make sure that certain features still work and don’t throw errors. For example, if we make a change to something in data entry, we check each part of data entry still works as it should for each scenario.

A main issue was coming up with tests that could properly validate if something had the correct output. For example, it is quite difficult to check if a 1000 object long list was parsed “Correctly”.

Most issues encountered with unit testing all revolve around having to deal with parsing large datasets, especially once the queries checking for duplicates were in place. This meant that to accurately test our methods that use our data we would have to delete and recreate all data in our database to get consistent results from the parsing method outputs and the methods that outputs depend on what data is in our database.

Testing issues in the future would all likely be very similar, as our complexity of analytics and total data in the database increases, so would the difficulty to write reliable tests.

9.2 Quality testing

Table 9 - Test 1

SUT:	MapViewer
Test:	**T1** Wifi Displaying for **A6**
Description:	Testing the display and filtering of Wifi points onto the google maps interface. Checking that the filtering works and that the delay is within the accepted bounds <1sec
Result:	Wifi points filtered as expected, with different colours to distinguish each type. Wifi locations updated as the filters are changed. These points can also be cleared with the click of a button, upon this action all points are removed
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U6**

Table 10 - Test 2

SUT:	MapViewer
Test:	**T2** Retailer Displaying for **A6**
Description:	Testing the display and filtering of Retailer points on the google maps interface. Checking that the filtering works and that the delay is within accepted bounds <1sec
Result:	Retailer points filtered as expected, with different colours to distinguish types. Locations for Retailers are where they should be. Filtering works as expected on all types. These points can also be cleared at the click of a button, upon this action all points are removed.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U6**

Table 11 - Test 3

SUT:	Graph Analytics
Test:	**T3** Graph Analytic for **A14**
Description:	Testing the display and correctness of the Graph Analytics window. Testing each graph to make sure it displays correctly, including making sure all axis are labelled, the number of records is not larger than the number of records in the database.
Result:	All graphs acted as expected with a default graph being displayed. Titles are not obscuring graphs. Filtering by datagroup works. Different selection of graphs from the combo box correspond to the expected graphs
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U13**

Table 12 - Test 4

SUT:	Route Planner
Test:	**T4** Finding Route for **A16**, **A9**
Description:	Testing RouteFinder window to make sure it can find the route between two entered locations and display the route on the map
Result:	Works as expected producing the route between two locations entered by the user. These locations can be entered by entering addresses into the corresponding text fields. Or by clicking on the map to add markers.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U15**, **U9**

Table 13 - Test 5

SUT:	DataViewer
Test:	**T5** Displaying specific data **A13**, **A15**
Description:	Test that data viewer shows data relevant to the user. Making sure that different datagroups can be viewed individually. Check that this data is persistent between uses.
Result:	data viewer allows for filtering by datagroup to allow the viewing of separate series of data as though they were individual files. The data can be from another user as the data and grouping remains consistent between operations of the application.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U12**, **U14**, **U7**

Table 14 - Test 6

SUT:	DataViewer (all types)
Test:	**T6** Ordering displayed data for **A17**
Description:	Test that the table works such that by clicking on a column header it will order the data in the viewer both top to bottom and reversed
Result:	Table view will order by alphabetical order not A-Z and Z-A for string fields and Low-to-High and High-to-Low for integers.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U6**

Table 15 - Test 7

SUT:	DataViewer (all types)
Test:	**T7** Filtering Data **A11**, **A7**
Description:	Testing that the data within the table will change when filtered. Making sure that the data displayed matches the filtering criteria selected for the test.
Result:	Filtering fields work as expected and only show the matching data. Lazy loading also continues to work when the filter is for large subsets of the data such as Food Retailers
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U16**

Table 16 - Test 8

SUT:	DataViewer (all types)
Test:	**T8** Searching **A10**
Description:	Testing the functionality of the search bar within the data viewer to show the data only matching the search criterion
Result:	Searching looks at the columns expected (primary column) for that type and the search can be easily reset by clicking the reset search button
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U10**

Table 17 - Test 9

SUT:	DataViewer (all types) Single selection window
Test:	**T9** Displaying singular data points **A8**
Description:	Testing the displaying of individual data points for all types of data; Trip, Wifi and Retailers. Making sure that fields matched those shown on the table from which it was selected
Result:	The data selected in all cases matched what was shown on the table and included more data previously not shown for the record in the table. The singular viewer buttons to move to records either side worked as expected along with the button to return to the table
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U8**

Table 18 - Test 10

SUT:	DataViewer (all types) Single selection window
Test:	**T10** Updating singular data points **A4**, **A5**
Description:	Testing the updating of individual data points from within the single view screen for all data types Retailer, Wifi and Trip. Making sure every field can be changed
Result:	The records change as expected, this change is reflected instantly in the database when the confirm button is clicked
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U5**

Table 19 - Test 11

SUT:	Input Handler
Test:	**T11** Adding Data from CSV(all types) **A1**, **A2**, **A3**
Description:	Testing that Retailers, Wifis, and Trips all load from CSV files that are selected by the user. And that should any errors occur the user is notified and that only the erroneous row is ignored. Then check that this data is visible from the data viewer without having removed previous data
Result:	The CSV loading works as expected however can take a long time with larger datasets. Specifically, retailers where it has to run google geocoding queries, this can lead to a problem if the user want to geocode more records than their key allows. The File loader also shows how many data points failed and gives a general message about the likely cause
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U1**, **U2**, **U3**

Table 20 - Test 12

SUT:	Input Handler
Test:	**T12** Adding data manually (all types) **A4**
Description:	Testing that adding data manually by filling out each field works, ensuring that fields only take valid inputs and that all necessary inputs are provided
Result:	Manual entries are sanity checked upon trying to add the object. An error message prints describing what is wrong to help guide the user. Sanity checks include making sure all required fields are filled. And that the data is useful, such as checking that time of trips is not too long, short or negative, checking that Wifi borough match the expected format. The combo boxes and date pickers also work as intended and help to enter data.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U2**, **U3**, **U4**

Table 21 - Test 13

SUT:	Map Analytics
Test:	**T13** Trip Analysis **A14**
Description:	Test that the Map Analytic window works. Checking that the heatmap is displayed and cleared and works with the selections the user has made
Result:	The heatmap works as expected with different filtering fields and datagroups. The clear button can be used to quickly remove the heatmap. Changing the density works and allows for faster loading of the heatmap or more detailed heatmaps.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U13**

Table 22 - Test 14

SUT:	Map Analytics
Test:	**T14** Displaying points on map **A14**, **A6**
Description:	Test that the Map Analytic window works. Checking that the Wifi and Retailer points are displayed and cleared matching to the POI datagroup selected. Station points should also be shown. These should work alongside the heatmap, with no adverse effects.
Result:	Wifis and Retailers showed up matching the datagroup and can be added and removed independently. The stations can be added to and removed from the map. Displaying and clearing of these points did not affect the heatmap.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U13**

Table 23 - Test 15

SUT:	Route Planner
Test:	**T15** Adding waypoints to route **A12**, **A9**
Description:	A Wifi and Retailer must be able to be added to the route such that a route from the start to the end with these waypoints is displayed.
Result:	Upon showing a route the nearest Wifi and the nearest Retailer can be added to the route. These additions can be chosen for each type. The route displayed then goes from start to end through the Wifi and retailer points provided
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U15**, **U9**

Table 24 - Test 16

SUT:	Route Planner
Test:	**T16** Displaying points on the map **A12**, **A6**
Description:	Test that the Wifi, Stations and Retailers how up on the map as expected
Result:	Stations show up upon loading of the map. Wifis displayed upon click matching the datagroup field. Similarly, for Retailers. Both can be cleared upon the click of the corresponding clear buttons. These points also show that the function to find nearest works as expected.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U11**

Table 25 - Test 17

SUT:	Database
Test:	**T17** Loading the stored database into the user's directory **A2**, **A15**
Description:	Test that the Database is loaded into the directory the jar is stored upon first load. Or whenever the database is not locatable.
Result:	The database is created as expected in a short time as expected and allows for data to viewed and then added to.
Pass/Fail:	Pass
Tester:	Morgan English
UC/Requirement:	**U1**, **U12**

9.3 Discussion

The overall test coverage generated from our IDE gave 25% of classes and 17% of methods. This low percentage comes mainly from the GUI controllers, as mentioned before, the large number of getters and setters in our classes, and the generic object classes, that contain a large amount of methods, but they are very “simple”, and so have no tests written for them.

Classes with 0% coverage (excluding controllers, generic near empty classes and window launchers) include JShandler, RouteHandler and SQLAnalytics. This is due to these all having methods that are very difficult to accurately test. This is because the output of these functions all depend on the state of the database at a certain moment, and so there can be no definite way to know if the output of a certain function is correct.

If an average is taken of the coverage of the classes that were actually tested, there is a 49% method coverage, with a 48% line coverage over those classes. A much better coverage, still not ideal but a lot better than the 17% as before. This is again a result of many getters, setters and ‘simple’ functions.

Whilst no manual tests were done specifically on the database it can be inferred that it works to the expected degree as manual tests relied on data provided by querying the database. From this it can be shown that the database structure can deliver what is promised and that the queries used work.

All acceptance tests have been specifically tested with a positive result sometimes across several tests, except for A18, which states “… the likelihood of encountering a bug is low”. This test cannot be fully tested due to the nature of software. As it is impossible to prove a program has no bugs. However, since all our other acceptance tests passed it is assumed that A18 is passed as well. As common bugs would be picked up during this testing phase.

All our manual tests passed as expected through the jar executable on both Windows and Linux operating systems. Whilst these tests are designed such that they check all areas of the application we cannot prove that these tests will pass under circumstances they were not designed for, however by testing over multiple platforms we increase confidence that the application will work.

Overall, functions and classes that are important to the functionality of our app all have tests of some sort written for them that effectively test if they are working correctly. Some have Junit tests written and some have manual test procedures. Our app at current passes all necessary unit tests and manual tests. However, there is room for more tests. Ideally there would be many more tests that attempt to break the app, but it proved difficult to come up with these tests.

10 Product Description

Meraki bikes has 4 main sections in its application, route planning, map viewer, analytics, data viewer, and data entry. The features in this app are outlined in our functional requirements and use cases, except for those not implemented as outlined later.

In the route planner, a user can either enter two address or click two locations on the displayed map, a route will then be planned for them. In addition, the nearest Wi-Fi or retailer locations can be displayed, with the user having the option of adding them to their route, which will then re calculate and be shown. The distance of their route will also be shown.

The map viewer shows locations on the map of retailers and/or Wi-Fi, with the option of filtering these by their values. A legend is also displayed on the map, showing what type each location is, e.g. Food retailer, Nightlife.

In analytics, the trip data is analysed by a heat map, showing the density of trips by way of colour, i.e. if the colour is darker on the map, more trips have been taken through that area. There is also the option of displaying retailers and Wi-Fi locations at the same time. All data displayed can be filtered by its relevant values. The graph section of analytics shows a breakdown of trip data, for example showing the range of ages and how many trips in each age bracket.

The data viewer shows all data in the app, with the ability to search, and filter by several key fields.

In data entry, a user can choose to manually upload individual retailer or Wi-Fi locations or trips, by filling in the needed fields, or they can upload a csv file with multiple objects within, which will be parsed and uploaded. This section has error and status feedback, letting the user know exactly what the result is. No duplicate data can be added into the app.

In this new release of the Meraki bikes app, we have introduced some new features into the app; being able to click a location on the map and showing the route on the route planner, a clear button to remove all markers and routes displayed on the route planner, bike stations on the route planner map, a filter box to filter which nearest Wi-Fi and/or retailer location to your route, the map view analysis of the local traffic, map viewer icon keys and clear button, data entry and graph analytics drop down menu to be more usable and the data viewer filtering capabilities increased. The data entry and graph analytics drop down menu were simply implemented so that they would be more user friendly. Most of the new features added to the route planning feature included some of the requirements including; **U6, U7, U9, U10, U11, U15, F7, F12, F13, F14, F29, F30, F31, F33, F34, F49**. The map viewer's new features such as the analysis included some of the requirements including; **U1, U2, U6, U10**. The data viewer's filtering capabilities being increased included some of the requirements including; **U6, U7, U8, U10, U12, U14, U16, F3 – F29, F32, F35 – F37, F48, F49**.

Our proudest features of the application include our route planner, a heat map of trip activity, our help screen, lazy loading, our database, and the analytic graphs. The route planner can plot a route between two points selected or entered by the user, then can add the nearest Wi-Fi or retailer in the area to that route. As well as this, all Wi-Fi and retailers within a certain proximity can be displayed with the user having the option of picking any one of them to add to their route. The heat map function can display a heat map of trips, with the more trips the darker the colour on the map. This can all be filtered by the trip age, gender, user

type, and data group. This in combination with the ability to be able to display Wi-Fi, bike stations and retailers on the same map makes this feature very powerful and useful to the relevant person. The help screen of our application uses the same layout as our actual app, so that if a user is confused about a certain feature, they can easily find information about that particular page. The lazy loading list view method can deal with data that is big and loads when scrolling. This helps the app to load data from the csv files in small successions when scrolling. The database allows us to query the data quickly for analytics and searching along with persistent storage. The analytic graphs and the transitions they make. The analytic graphs look professional, they are aesthetically pleasing to the users and are not boring, mono-coloured graphs so the users can happily see the analytics we have to offer.

Features that are not implemented are functional requirements **F51 – F56**, and **F58 – F64**, relating to use cases **U2, U3, U5, U6, U7, U10** and **U14**. Five of these are of medium priority and eight are of low priority. Nine of these features revolve around using an online database to access and store data outside of the local PC, with most having a low priority relating to use cases **U2, U3, U14**. Four features are being able to filter/search retailer by primary type and borough and, Wi-Fi records by name and type, and trip records by date, with most being of medium priority, all relating to use cases **U6, U7, U10**. One being able to update the details of multiple records at once with a low priority and relating to use case **U5**. And the remaining missing feature relates to analysing frequency of retailer and Wi-Fi locations being added to routes, with a low priority relating to use case **U13**.

11 Changelog

11.1 Phase 2 Changes

Section Number	Section Title	Change
0	Executive Summary	Consistency changes with rest of document
1	Business and System Context	Major edits throughout to make business and system context more realistic
2.1	Stakeholders	Minor edits with the data-sharing company stakeholder
2.2.1	Use Cases	Major edits to all use cases' format and use case diagram
2.2.2	Functional requirements	Refactoring of many different functional requirements and some changes to priorities
2.2.3	Quality Requirements	Consistency changes and minor edits to descriptions
2.2.4	Key Drivers	Minor edits to weightings to keep consistency and major edits to discussion.
3	Acceptance Tests	Minor changes to descriptions to keep consistency and increase clarity.
5	Deployment Model	Minor changes to deployment model diagram and description
6	UML Class Diagrams	Major redo of UML class diagrams to reflect implementation
7	Risk Assessment	Minor edit to risks table to include risk 15
8	Project Plan	Redo of project plan to focus on Deliverable 3
9	Test protocol and testing procedures	Added new section
10	Product description	Added new section

Figure 13 - Changelog for Phase 2

11.2 Phase 3 Changes

Section Number	Section Title	Change
2.2.2	Functional Requirements	Refactored many functional requirements to be more specific and removed requirements that were not relevant. Also reordered and renumbered the requirements.
5	Deployment Model	Removed online database from the deployment model and added mention of the local database that will be created
6	UML Class Diagram	Updated the figures to reflect the current version of the application and added explanations for new packages that were added to the application.
9	Test protocol and testing procedures	Recompleted previous manual tests to ensure they still passed. Some minor changes to these tests. Added more manual tests for the new features we had implemented
10	Product Description	Redone the description of the whole product, new features, most proud features and the features that were intended to be implemented but were not.
12	Key Lesson Learnt	Added this entire new section.

Figure 14 - Changelog for Phase 3

12 Key Lessons Learnt

Connor McEwan-McDowall

This project has taught me a lot. Firstly, I have learnt how powerful a team can be when everyone is on the same page and working together with the same vision. But on the other hand, I have seen how much extra work it can be when people are not on the same page and things are happening two different ways creating conflicts. That is why I have a new-found respect for the importance of good and detailed planning as well as management of the plan created. I now better realise that a team can be extremely powerful provided planning and proper management is used.

Secondly, I have realised that there are many ways that people work and approach problems meaning that I must be flexible in how I work to both increase performance, as well as gain experience by working from a different standpoint.

Lastly, I have learnt that a team project can form a really cool relationship with your peers and a good team can definitely be told apart from a poor one just by getting along with each other.

Aidan Smith

Firstly, learnt that when working in a large team, meetings are key to progress and can be quite productive just as work sessions if individuals are struggling to meet their commitments. In this case the meetings effectively function as a motivating aspect that is potentially lacking for individuals.

Secondly, I learnt how to properly format use cases, functional requirements and tasks for planning. I also learnt the resolution of information required in each of these to allow for them to be useful for interpretation.

Lastly, I learnt that I should ask as many questions as possible as early as possible even if I think I have a solution to a problem. This is because, while I most likely do have a working solution it is not the best practice in terms of design quality and will have an impact later that I could not foresee (specifically in this project it was not asking how MVC would be implemented before starting to code).

Morgan English

The importance of delegating tasks adaptively and keeping a plan. This helps as bugs inevitably show up and starting early gives more time should a task be harder or easier than expected the plan can be changed and other tasks can be re-delegated. It is also important this plan is up to date and viewable by everyone on the team so the team can function well.

The need for Communication within a team. Communication was vital to ensure that people could work to their full potential and get help when needed, whether it be from other team members or staff/tutors. This communication is important to overcome challenges such as bugs and questions.

That it is good to see what technologies already exist that may help solve your problem. And if one is found to do some research and testing of it before using it in your project. This helps get an idea of what it is capable of and how you can best use the technology.

Mitchell Fenwick

The importance of fully communicating with each other what our perspective is, i.e. Each other’s perspective on the direction of our application or how one of the features of our

application will be implemented. This is very important as throughout the development of the app there were many times when not everyone in the group was clear on who our application was for, and other things like having similar features in different parts of the app.

Another key lesson learned would be to set deadlines early. There were many times when our group had deadlines set but they were kind of set a bit too late and we ended up not meeting set deadlines often. Having these deadlines set earlier would have been better as we would have maybe been able to keep to the deadlines and finish work beforehand. It is hard to tell how long something will take in a coding perspective though.

Another key lesson leaner is to get to know your group early on. Get to know each other, how they work, how much work they will put in etc. as this will help get the group to the 'productive' stage quicker and result in less arguments or disruptions in group progress.

Lachlan Brewster

How much good planning can improve the product as well as make a large task seem much more manageable, in workload and team management.

That there are many software engineering tools out there, and we should take full advantage of whatever we can find, there's no need to re do something that's already been done by someone on stack overflow.

Joshua Meneghini

The most important lesson learnt from this project for me was working as a team. I have not had so much work assigned as a team like this project before and it has taught me valuable lessons on how important communication and meetings can be when production levels are high. I think I have also noticed that everyone must do their part in completing our assigned tasks on time to work good as a team and keep everyone happy.

I also learnt how to code some JavaScript code while doing the route planner feature. How to communicate between java and JavaScript. This was completed through a 'bridge' which was definitely new to me.

Lastly, I learnt that leaving an assignment to the last minute may be a few student's tendency and with this assignment, in a team environment, this could just not happen as you will always end up letting someone down and getting the whole team behind if you leave something to the last minute. Therefore, starting early is always key and requires a good level of organisation.

13 References

Furfaro, D. (2016, August 10). Rogue bike sharing company sets sights on NYC. Retrieved from NYPost.com: http://nypost.com/2017/08/10/rogue-bike-sharing-company-sets-sights-on-nyc/

Google, P. S. (n.d.). Spotcylce. Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.eightd.android.spotcycle&hl=en

Google, P. S. (n.d.). oBike. Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.obike

Google, P. S. (n.d.). MoBike. Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.mobike.mobikeapp

Nacto. (2016). Bike share statistics. Retrieved from Nacto.org: https://nacto.org/bike-share-statistics-2016/

Nielsen, J. (1995, January 1). 10 Usability Heuristics for User Interface Design. Retrieved from Nielsen Norman Group: https://www.nngroup.com/articles/ten-usability-heuristics/