

SENG202 Assignment

Team 1: Cameron Auld, Joshua
Bernasconi, Josh Burt, Ollie Chick, and
Ridge Nairn

CYC: Your Cycling

A Cycling Route Planner
and Data Analysis Tool

10 October 2017

Table of Contents

Table of Contents	1
Executive Summary	3
1. Business and System Context	4
1.1 Motivation	4
1.2 Market Research	4
1.3 Advantages	5
1.4 Disadvantages	5
1.5 System Context	5
2. Stakeholders and Requirements	7
2.1 Stakeholders	7
2.2 Use Cases	8
2.3 Functional Requirements	13
2.4 Quality Requirements	17
2.5 Key Drivers	18
3. Acceptance Tests	19
4. GUI Prototypes	20
5. Deployment Model	22
6. Detailed UML Class Diagram	22
7. Risk Assessment	24
8. Project Plan	26
8.1 Major Milestones	26
8.2 Minor Milestones	26
8.3 Progress Reviews	28
8.4 Deliverable 1	28
8.5 Deliverable 2	28
8.6 Deliverable 3	29
9. Testing Protocol and Procedures	29
9.1 Functional Testing	29
9.2 Quality Testing	32
9.3 Discussion	32
10. Current Product Version	34

10.1 Requirements and Features Implemented	34
10.2 Features We Are Most Proud Of	37
10.3 Requirements and Features Not Implemented	37
References	40

Executive Summary

This document describes a system for route planning and data analysis of cycling routes.

Business & System Context: The core experience is surrounding planning a route, and being given or selecting potential waypoints to visit on the trip, including retailers and WiFi hotspots, as well as any custom waypoints the user adds. Additional features include exporting, importing and the analysis of bulk data. Being a desktop application, there are some key differentiators from primarily mobile applications. Being able to handle large quantities of data is a result of this. However, this is at the cost of portability. Another differentiator is the inclusion of WiFi hotspots, and other local places to visit.

Stakeholders & Requirements: The target market for this system is primarily cyclists, such as tourists and commuters. There are also a few secondary stakeholders, who do not directly interact with the application but are affected by it, including local retailers and city government. Use cases for CYC include signing up, planning a route, and adding custom points, among others. The most important quality requirements are for the system to be responsive, and intuitive to use. These are important as they help ensure users keeping using the application and are not frustrated or intimidated by it.

GUI Prototypes: Skeletons of the main views are included for rapid production of functional GUI. These are used to visualise the application and how it will accomplish use cases. These prototypes also enable the developers to check how the application flows from the perspective of a user, and to look at GUI without thinking of implementation specifics and constraints.

Deployment Model: The app will be deployed on the users' device and will use the services of Google Maps. The JAR contains default CSVs, and the user can also load data from their own CSVs.

UML Class Diagram: Included is a package structure overview, along with detailed class diagrams. These diagrams reflect the Model View Controller (MVC) design pattern decided on for CYC.

Risk Assessment: There are a few risks which need to be considered. Major risks include team members being sick, and overestimation of the amount of work that can be completed. With proper communication, these can be mitigated.

Project Plan: A plan of the amount of work expected to be achieved weekly as well as buffer time is discussed. This has taken into account the time commitments of the development team.

Testing Protocol & Procedures: Various functional and quality testing has been carried out on CYC. Three of the four high priority acceptance tests pass, with the fourth not far away. GUI and controller testing has been carried out visually. All model classes have corresponding unit tests. These procedures allow us to be confident in the quality of the CYC system.

Current Product Version: Functional requirements that have and have not been meet are listed here giving a brief overview of the general state of the product. Of the 40 functional requirements specified, 22 have been implemented. Included here are the three features we are the most proud of: The map views placement of points of interest and ability to accurately cluster them, the filtering accuracy and speed in the table view and the general ease of use that the application has as a whole.

1. Business and System Context

Our cyclist data tracking application, named CYC (a recursive acronym for CYC: Your Cycling), will allow a wide range of users to view, edit, create and delete various bike trip data as well as display services near a planned route. The system will load data about bike trips, WiFi locations in New York City and retailers in lower Manhattan from external files, and display these to the user in a tabular form. The system will also display route data on a map, with the nearest WiFi locations and retailers listed for said route. We are focusing on two main aspects in our system. One focus is simplifying access to the trip data and making it quick and easy to find useful nearby services. The other focus is providing analysis and key facts about the data in both tabular and graphical form.

The target users of this system are cyclists (mainly commuters and tourists) and local service providers (including WiFi services and retailers). Cyclists are the only stakeholders we envision using our service, with the majority of the stakeholders being interested parties.

The features all types of cyclists will have access to include:

- User accounts for tracking a specific user's data, system sessions and data changes.
- Point-to-point route planning.
- Filtering through the data to find a specific service.

Tourists and commuters will be interested in:

- Listing of nearby services (to a route or a single location).
- Sharing of trips or analysed data.
- Add work and home points, among other custom points.

Local service providers will be able to use our service to:

- View the competitors for their service in a particular area.
- Add their own service to the data.

Fitness cyclers and professionals may find other existing solutions accommodate their needs better, whoever they may want access to:

- Basic analysis of data (times, distance, average speed etc.).

1.1 Motivation

Development of this application is worthwhile because it combines data viewing and analysis with bike routes and nearby services, which has not been done before. Most applications in the market today provide a lot of complex analysis of cycling and fitness data, whereas our application will provide simple, key facts about trips. The majority of those other apps (such as Strava, Map My Ride and Cyclemeter) are mobile applications that focus on GPS and training tracking, but do not provide the combination of data and mapping information our desktop application provides. Another application in our target market is Google Maps; our application will build on top of the Google Maps API. Another driver for the development of this application is the shared developer interest in creating the best possible product, resulting in the best possible mark for the SENG202 course.

1.2 Market Research

CYC competes quite well with the current solutions in the market today, as can be seen by the comparison below in figure 1.

GoldenCheetah is a desktop competitor in the data analysis side of our system. It provides in-depth analysis of cycling and GPS data, and supports editing of this data (1, GoldenCheetah, 2017) . This desktop application supports importing data from various popular GPS and cycle tracking hardware devices, which have not been considered in our system. However, our system is focused on simplifying key facts and working with the data sets provided, as well as the map system that is not supported by GoldenCheetah.

As mentioned above, most cycling applications are mobile based, with the most popular being Strava. Strava provides GPS route visualisation, trip statistics and data analysis. Strava combines this with a social aspect: users can share activity and compete (2, Strava, 2017), which is something that we have not considered in our system. Strava also connects to other GPS tracking systems. Our system will serve a different purpose to these fitness tracking apps as we are providing route planning, which includes searching for nearby WiFi and retailers on a route, as opposed to GPS tracking. Our system also differs from these applications by being desktop-based, allowing for more information to be displayed in more than one format.

MapMyRide is a web-based application for planning cycle routes. It provides detailed distance breakdowns, elevation details, and quick options for routes such as out-and-back, reversing routes and return trips (3, MapMyRide, 2017) . It also allows users to save routes and send them to a mobile device. One feature our system provides that MapMyRide does not is the stops for nearby WiFi and retailers.

System	Open-source	Lightweight	Dedicated Application	Java	Big Data
Golden Cheetah	✓	✗	✓	✗	✓
Strava	✗	✗	✗	✗	✗
MapMyRide	✗	✓	✗	✗	✗
CYC	✓	✓	✓	✓	✓

Figure 1: Comparison of CYC and other existing systems.

1.3 Advantages

CYC is a desktop application, so it has the potential to handle bigger data sets and has more functionality than a mobile application. Its combination of graphical representation and raw data viewing makes the application multi-purpose. It is easy to filter and view data that would otherwise be difficult to navigate. It combines various data sources into one location, allowing new perspectives on data. It also allows planning of cycle routes to include specific services. It is written in Java, so it can be easily ported to a web application or Android app later. It is a lightweight application that take minimal resources to run. Open source allows for users to further extend the functionality of the application.

1.4 Disadvantages

CYC is limited to a desktop application, so route information cannot be taken with the user while they are cycling. There are also no route tracking capabilities, nor integration with existing hardware such as cycle computers or GPS trackers. Access to initial data will be limited to WiFi locations in New York City and retailers in lower Manhattan.

1.5 System Context

The CYC system, its actors and the external systems are displayed in Figure 2, along with the actions the actors perform on the system.

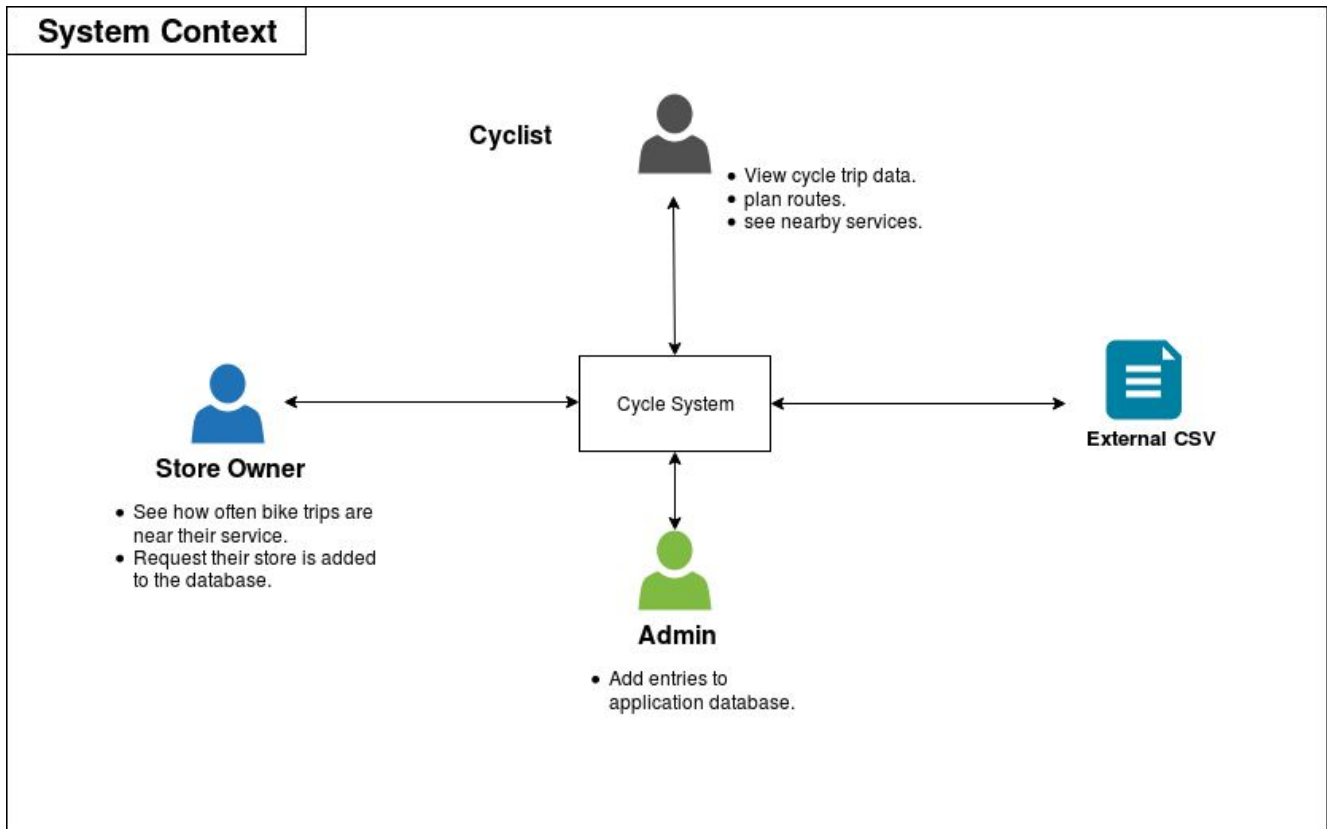


Figure 2: The system context of CYC.

1.6 Future Business Context

For future releases the follow feature can be implemented:

- More data analysis features, including average trip duration and distance covered.
- Different account type to allow separate out heavy data analysis from cyclist users.

2. Stakeholders and Requirements

2.1 Stakeholders

The stakeholders for CYC are listed in Table 1, where they are sorted by priority. Stakeholders are defined as either end users, interested party or client.

Table 1: The stakeholders for CYC.

ID	Stakeholder	Description	User Type	Priority
S1	Cyclists	General cyclists, such as those who commute by bike, or for fun/exercise. They may wish to track and store their data.	End User	High
S2	Local retailers	Local retailers interested in routes commonly taken through their area. They may want to get information about impressions from the application and promote their location to cyclists.	Interested Party	High
S3	Application developers	Developers (SENG202 Group 1). They will develop the application.	Interested Party	Medium
S4	Course tutors and lecturers	Staff at the University of Canterbury who will be grading or reviewing our work. They are interested in the outcome of our project and will be evaluating it.	Client	Medium
S5	Developers' friends and family	Friends and family of the members of Group 1. They may wish to use and try out the application to see what has been built.	Interested Party	Low
S6	Local government and authorities	City council (more specifically departments involved in cycle routes and planning) and police. They could use information gathered to see which routes are popular, and consider improvements that could be made to infrastructure.	Interested Party	Low

2.2 Use Cases

Use case diagrams are shown below in figures 3,4,5,6. Below that is a table further detailing the flow and pre and post conditions for the use cases.

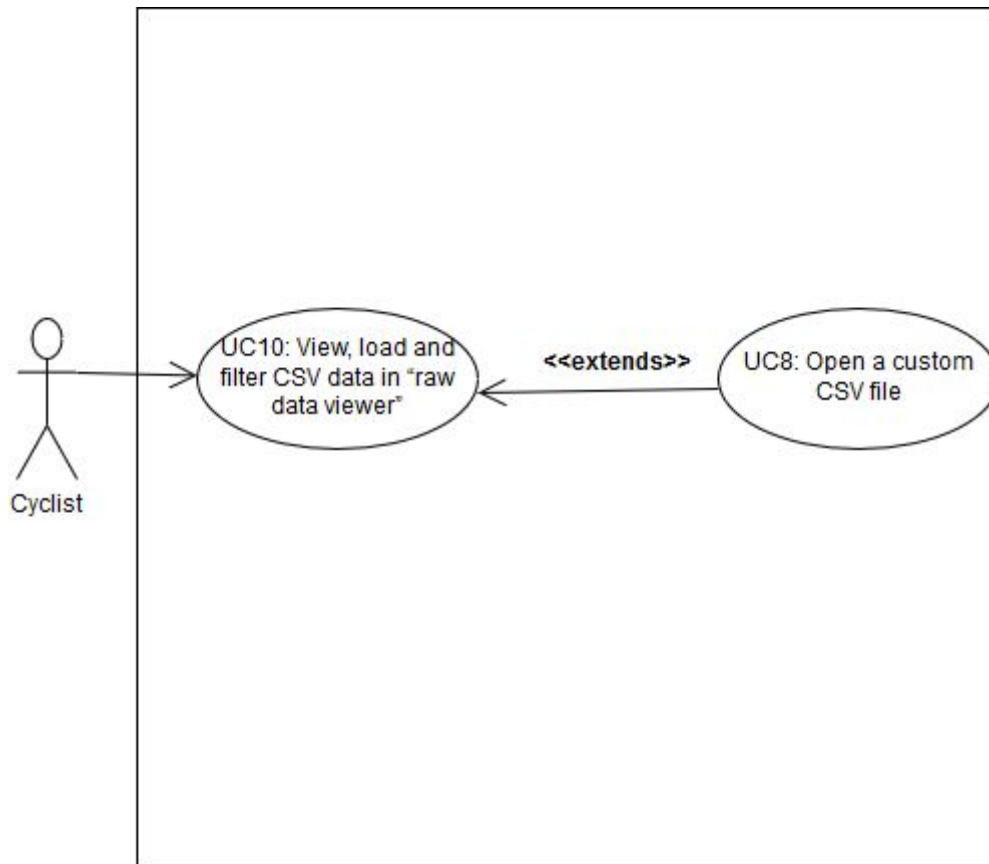


Figure 3: Use cases for CSV Files.

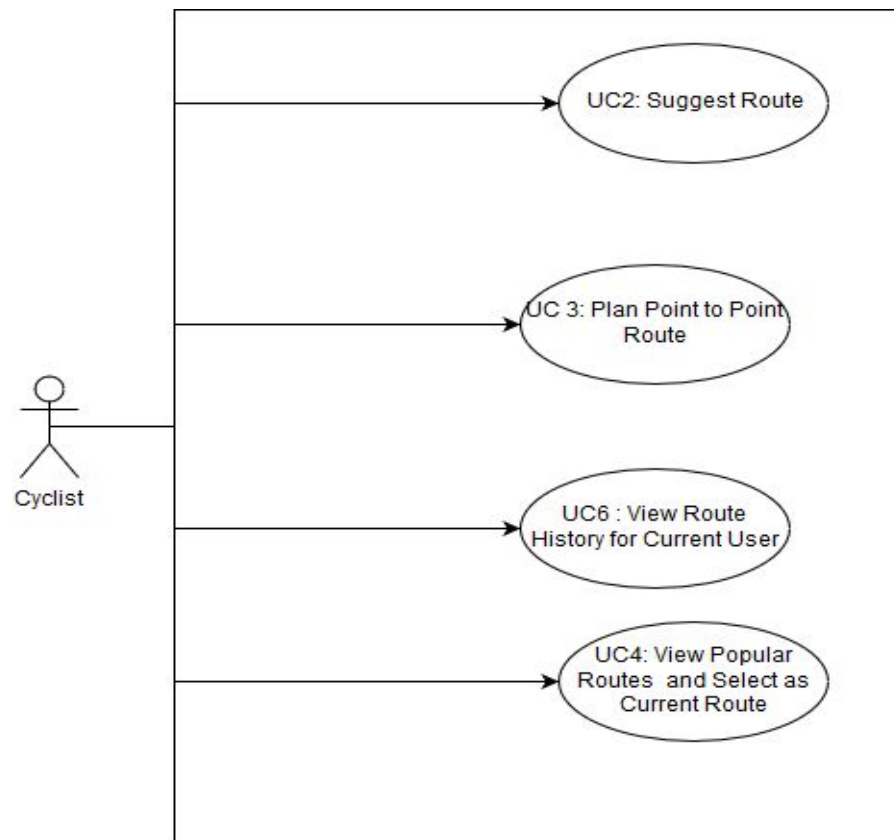


Figure 4: Use case Diagram for Route Planning

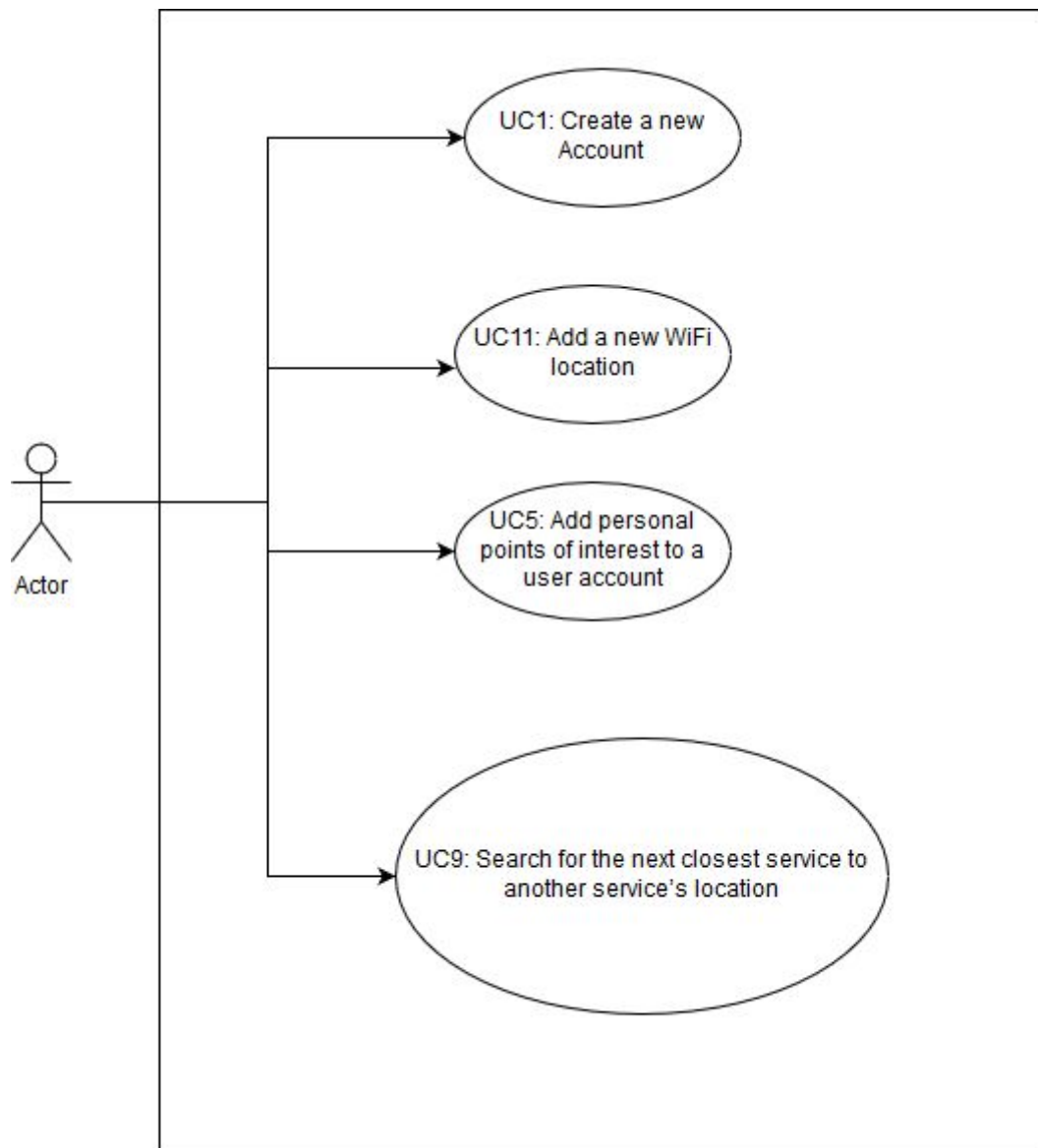


Figure 5: Use Case Diagram for Account Lifecycle

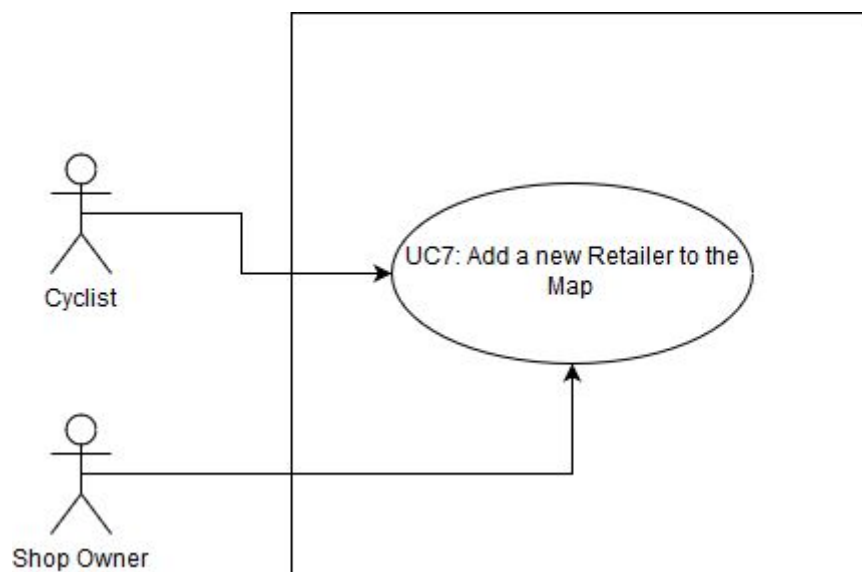


Figure 6: Adding a Retailer to the Map View Use Case Diagram

Table 2: Use cases for CYC.

ID	Use case	Actor	Result	Preconditions
UC1	Create a new account	Cyclist	Adds a new User to the application.	The user has the application open.
UC2	Suggest route	Cyclist	Suggests a route to and from a place with any wanted stops.	The application is open, logged in, data is loaded and available and is in the map view.
UC3	Plan point to point route	Cyclist	Allows a route from one point to another to be created with the option to find nearby WiFi and retailers.	The application is open, logged in and is in the map view.
UC4	View popular routes and select it as current route	Cyclist	Shows popular routes based near current start point.	The application open and logged in, data is loaded and available and is in the map view.
UC5	Add personal points of interest to a user account	Cyclist	Adds a particular point to the database. Viewable in anonymous route data.	The application is open, logged in and is in the map view. User enters point to be added to personal points
UC6	View route history for a current user	Cyclist	Shows a specific user their previous routes.	The application is open, logged in, data is loaded and available and is in the map view.
UC7	Add a new retailer to the map	Retailers, Cyclists	Adds a new retailer waypoint.	The application is open, logged in and in the map view.
UC8	Open a custom CSV data file	Cyclists	The data is displayed in the raw data viewer.	CSV file exists. The Application is open , logged in and in the table viewer
UC9	Search for the next closest service to another service's location	Cyclist	The next closest WiFi or retailer is displayed.	The application is open, logged in,data is loaded and available and is in the map view.

UC10	View, load and filter CSV data in "raw data viewer"	Cyclists	The bike trip, WiFi, or retailer information is displayed and can be filtered by different criteria.	CSV file exists. The Application is open , logged in and in the table viewer
UC11	Add a new WiFi location	Cyclist	Adds a new location of public accessible WiFi to the database.	The application is open, logged in and is in the map view.

Table 3: Use case scenarios.

ID	Blue sky scenario	Postconditions	Alternate flow
UC1	<ol style="list-style-type: none"> 1. Request is sent to the account handling method 2. Validity and uniqueness is checked 3. Account is approved and user is logged in 	New account is created for the user and the new user is logged in.	<ol style="list-style-type: none"> 1. Request is sent to the account handling method 2. Validity or uniqueness check fails 3. User is shown appropriate error message
UC2	<ol style="list-style-type: none"> 1. User's inputs a location 2. Location is searched in the database and most popular route for this location is suggested 	User is shown a suggested route, and has the option to select it as their current route.	<ol style="list-style-type: none"> 1. Location has no routes available to show. 2. User is informed.
UC3	<ol style="list-style-type: none"> 1. User enters start and endpoint to their route 2. Request is sent to the routing module 3. Most appropriate route is shown to the user as the current route 	User can accept the current route or choose an alternative.	<ol style="list-style-type: none"> 1. User enters a start and endpoint to their route 2. Request is sent to the routing module 3. No route can be found user is alerted to this
UC4	<ol style="list-style-type: none"> 1. User requests popular routes for current location 2. Location is retrieved 3. Database is queried for the ten most popular routes from this location 	User is shown a list of routes and can select one as current.	<ol style="list-style-type: none"> 1. User requests popular routes for current location 2. Location is retrieved 3. No routes can be found for location and user is alerted
UC5	<ol style="list-style-type: none"> 1. Point is checked against the user's personal points for uniqueness and validity 2. Point is added to the user's account and conformation is shown 	User has a new point added to their account, this is available only to them.	<ol style="list-style-type: none"> 1. Point is found to be non unique or invalid 2. Point is discarded 3. Users is alerted the point was unable to be added.
UC6	<ol style="list-style-type: none"> 1. User requests history 2. Database is queried 3. Last 10 trips are returned 	User history is available to the application to be displayed.	<ol style="list-style-type: none"> 1. User requests history 2. Database is queried 3. No history is found and user is alerted

UC7	<ol style="list-style-type: none"> 1. Request is received and is checked for uniqueness and validity 2. Request is approved by the application 3. Retailer is the added to the database 	New retailer is searchable in the database.	<ol style="list-style-type: none"> 1. Request is received and is checked for uniqueness and validity 2. Request is rejected by the application 3. Applicant is alerted and shown appropriate error message.
UC8	<ol style="list-style-type: none"> 1. CSV file is opened 2. Data is made available to the display module 	Display module is able to manipulate this data.	<ol style="list-style-type: none"> 1. CSV file is unable to be opened 2. Buffer is purged 3. User is shown an error message
UC9	<ol style="list-style-type: none"> 1. User location is retrieved 2. Database is queried and nearest WiFi location is returned 	WiFi location is able to be routed to.	<ol style="list-style-type: none"> 1. User location is retrieved 2. There are no nearby WiFi or retailers and the user is notified and asked if they want to expand the search area
UC10	<ol style="list-style-type: none"> 1. CSV file is opened 2. Data is made available to the filter module 	Data is added to the dataset.	<ol style="list-style-type: none"> 1. CSV file is unable to be opened 2. Buffer is purged 3. User is shown an error message
UC11	<ol style="list-style-type: none"> 1. Request is received and is checked for uniqueness and validity 2. Request is approved by the application 3. Request is then passed to the admin account for final approval 4. Retailer is the added to the database 	New WiFi location is added to the database and can be viewed by users.	<ol style="list-style-type: none"> 1. Request is received and is checked for uniqueness and validity 2. Request is rejected by the application 3. Applicant is alerted and shown appropriate error message

2.3 Functional Requirements

Functional requirements that the system must meet in order to fulfill the use cases in Table 2 and 3 are listed in Table 4, sorted based on priority.

Table 4: Functional requirements for CYC.

ID	Description	Stakeholders	Priority	Use cases(s)
FR1	The application can load data about bike trips by parsing CSV files.	S1, S2, S6	High	UC10
FR2	The application can load 'WiFi locations in NYC' data by parsing CSV files.	S1, S2, S6	High	UC10
FR3	The application can load 'Lower Manhattan Retailers' data by parsing CSV files.	S1, S2, S6	High	UC10
FR4	User can calculate the distance between two waypoints of a bike trip, based on the difference between their longitude and latitude.	S1	High	UC3
FR5	User can calculate the distance between the start and end points of a route based on the total distance traveled.	S1	High	UC3
FR6	Given start and end points a possible route is generated and displayed to the user	S1	High	UC2
FR7	User can find the closest WiFi hotspot to a given retailer.	S1, S2	Medium	UC9
FR8	User can enter a start point and request popular routes. The most common routes taken from this starting location should then be displayed for the user. The user can then select one of these routes.	S1	High	UC4
FR11	User can find the closest retailers to a bike route.	S1, S2	Medium	UC2, UC9
FR12	Duplicate records are not imported to a list. The user is notified of the duplicates.	S1, S2, S6	High	UC8, UC10
FR13	User can add new trips to the list of trips.	S1	High	UC5, UC10
FR14	User can add new WiFi locations to the list of WiFi locations.	S1	Medium	UC5, UC11

FR15	User can add new unique retailers to the list of retailers. Duplicate retailers will not be accepted.	S1, S2	Medium	UC5, UC7
FR16	User can update existing records.	S1,S2, S6	High	UC10
FR17	User can either log in to an existing user account or create a new account.	S1, S2,S6	High	UC1
FR18	User can create an account by accepting the terms of service.	S1, S2, S6	High	UC1
FR19	Accounts have default 'bike trip', 'WiFi' and 'retailer' lists when created.	S1, S2, S6	High	UC1
FR20	The data visualization screen has a window that displays a map of New York City.	S1	High	UC3, UC4, UC6
FR21	Bike trips can be displayed visually on the map.	S1	High	UC3, UC4, UC6
FR22	WiFi locations can be displayed on the map.	S1	High	UC3, UC9
FR23	Retailer locations can be shown on the map.	S1, S2	High	UC3,UC9
FR24	The retailers along a bike trip can be displayed on the map, with their distances ranked.	S1, S2	High	UC3, UC9
FR25	The WiFi access points near a retailer can be displayed on the map, with their distances ranked.	S1, S2	Medium	UC3, UC9, UC11
FR26	The WiFi access points along a bike trip can be displayed on the map with their distances ranked	S1, S2	Medium	UC3, UC9
FR27	User can load CSV data with an alternate number of attributes per entity/line as defined by the user at runtime.	S1	Medium	UC8
FR28	The raw data viewer lists data for bike trips, WiFi hotspots, and retailers, each in their own separate table.	S1, S2	Medium	UC10
FR29	For each record on each list, basic details are shown by default. Upon opening a record, further details are displayed.	S1, S2	Medium	UC10
FR30	User can filter bike trip data by start point, end point, bike ID, or gender.	S1	Medium	UC10

FR31	User can filter WiFi location data by borough, type, and provider.	S1, S2	Medium	UC10
FR32	User can filter retailer data by street and ZIP code.	S1, S2	Medium	UC10
FR33	User can search bike trips based on search criteria: either absolute search criteria (e.g. 'start location X'), or ranges (e.g. 'start location between X and Y').	S1	Medium	UC10
FR34	User can search WiFi locations based on search criteria: either absolute search criteria (e.g. at location X'), or ranges (e.g. located between X and Y').	S1	Medium	UC10
FR35	User can search retailers based on search criteria: either absolute search criteria (e.g. at location X'), or ranges (e.g. located between X and Y').	S1, S2	Medium	UC10
FR36	User can sort bike trips based on distance.	S1	Medium	UC10
FR37	When a new data set is imported, the data set can be added to an existing list of its type, or create a new list of said type.	S1	Medium	UC8, UC10
FR38	When multiple lists are associated with an account, the user can select between lists within the information panel.	S1	Medium	UC10
FR40	When a user adds data to their lists, it is persistent.	S1	Medium	UC6, UC10
FR41	Bike trip, WiFi, and retailer data can be exported over a network to a central database.	S1, S2	Medium	UC5, UC7, UC11
FR44	When a user attempts to log in to an account, the password they have entered is checked against the hash of the password for that account; if successful, they are logged in.	S1	Medium	UC1
FR45	User can export their lists as CSV files.	S1, S2	Low	UC10

2.4 Quality Requirements

Quality requirements that the system must meet in order to fulfill non-functional stakeholder requirements (see Table 1 for stakeholders) are listed in Table 5, where they are sorted by their priority.

Table 5: Quality requirements for CYC.

ID	Description	Stakeholder	Priority	Use Case
QR1	The response time in interactive use on UC lab computers should be: <ol style="list-style-type: none">1. Less than 200 ms for basic UI interaction and single local database fetches2. Less than 1000ms for single remote database fetches and logins3. Less than 5000ms for importing and exporting 500 rows of data to local database	S1	High	No Use cases implement the database explicitly
QR2	In an unstructured usability test on a UC lab computer, a first time user, with average computing skills, should be able to learn to: <ol style="list-style-type: none">1. Create an account and login in less than 1 minute2. Create a route in less than 3 minutes3. Search for the nearest service to another service in less than 2 minutes	S1	High	UC1, UC2, UC3, UC9
QR3	All application code must comply with internal code style standards.	S3, S4	Medium	N/A
QR4	The system should run on Windows, Mac and Linux platforms without requiring changes to the source code.	S3, S4	Low	N/A
QR5	The centralized database should store and serve user submitted data in such a way that does not violate any privacy laws of the country and state of the user. Specifically the following countries/states: <ol style="list-style-type: none">1. New Zealand2. United States of America3. New York State	S6, S3, S4	Low	N/A

2.5 Key Drivers

The most important quality requirements (see Table 6) were ranked for each stakeholder, and then multiplied by a factor based on the stakeholder's priority, to give key drivers. These values were estimated by the developers. This provides focus to the design process. A stakeholder with high priority is given a weighting of 3, medium 2 and low 1. Stakeholder 5, developer's friend and family, has been removed from the table as any use of theirs would be identical S1.

Table 6: Key drivers of CYC.

Stakeholder	Weight	Usability (QR2)	Performance (QR1)	Privacy (QR5)	Portability (QR4)	Total
S1	3	40	30	20	10	100
S2	3	60	20	0	20	100
S3	2	30	30	10	30	100
S4	2	30	30	10	30	100
S6	1	25	25	25	25	100
Total	-	445	295	125	235	1100
Normalised total	-	40	27	12	21	100

Based on the normalised totals, usability (QR2) is defined to be the most important aspect of the project, performance (QR1) and portability (QR1) are also of importance, with privacy (QR5) being a concern but not that important to our stakeholders.

3. Acceptance Tests

The acceptance test are shown in Table 7, sorted by priority. The higher the priority, the more important it is that the test doesn't fail. The acceptance tests are all the responsibility of the developers for this project.

Table 7: Acceptance tests for CYC.

ID	Description	Acceptance criteria	Priority	Use case(s)
AT1	User is able to add an account and access the application	A new user, given the application open on screen, with basic computing skills should be able to signup and login in under 5 minutes with no prompting from the developers.	High	UC1
AT2	Suggests route to a place from a place	A user with the map view open can select two points on the map and a route is generated. Route suggested is a suitable route, correctly includes the start and end points. Additional waypoints are able to be added and correctly routed through.	High	UC2 and UC3
AT3	Data can be viewed and filtered in the table view.	A user with the application open can select the table view. The user is then able to see loaded data in the table. Filters can then be selected, applied and the data is filtered and shown to the user with no delay.	High	UC10
AT4	CSV files can be loaded and displayed in a reasonable time.	User is able to open a file selector, select a csv file of 1000 entries and have it open and displayed in the table view in under 5 seconds.	High	UC8
AT10	The nearest service of a certain type can be found for a given point	A user with the map view open and the data set loaded into the application can then select a point of interest. From this point of interest the nearest relevant point should be displayed. A route to this point should then be offered.	Medium	UC9
AT5	View popular routes and then accept that as the current route.	A user with the map view open can pick a point either by clicking on the map or entering the address. A list of 10 popular routes for that location is then shown to them. A route can then be selected as the current route.	Medium	UC4
AT6	Add personal points	A user with the map view open is able to specify GPS co-ordinates as a custom waypoint and can then name the point. This point is stored persistently and is available to the user on their further usage of the account.	Medium	UC5
AT7	User can view personal route history.	A user with the application open, can select in either the table view or the map view, is able to view previously travelled routes on a map or table. A previous route can then be selected as the current route.	Medium	UC6

AT8	User can add custom Wifi locations	A user with the application open in the table or map view, is able to enter a point that is then stored as a custom wifi point for them.	Medium	UC11
AT9	Additional retailers able to be added	A User is able to add new retailers as waypoints to the application.	Low	UC7
AT1 1	Data Points can be modified	A User can select a point from a list and edit the point, persistently and only on the selected list.	High	UC5
AT1 2	Data Points can be deleted.	A User can delete a point from a list and delete the point, persistently and only on the selected list. Optionally this can be extended to all points in a list	High	UC5
AT1 3	Lists can be created	A User can create a list of data of a single type. This can be populated.	High	UC8, UC10
AT1 4	Lists can be selected	A User can change between Lists after their creation.	High	UC8, UC10

4. GUI Prototypes

The login screen will prompt the user to enter their username and password, as well as give the option to create a user. This is shown in Figure 7.

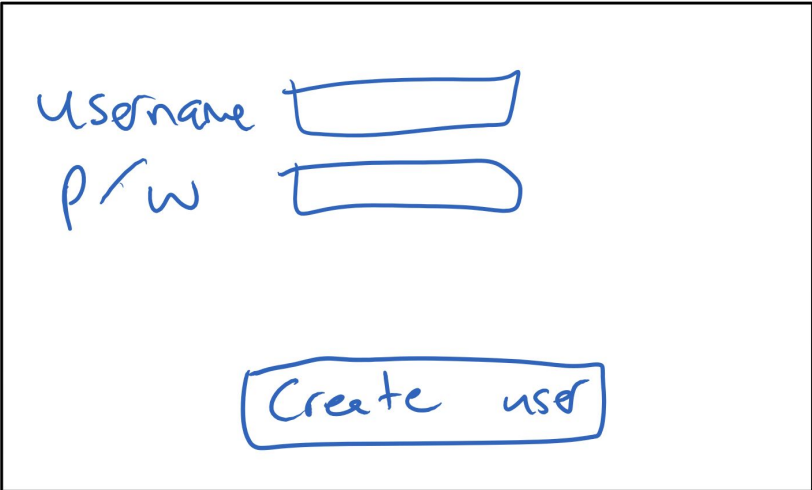


Figure 7: The login screen of CYC.

The create account screen will offer options for the user to setup their account. This is shown in Figure 8.

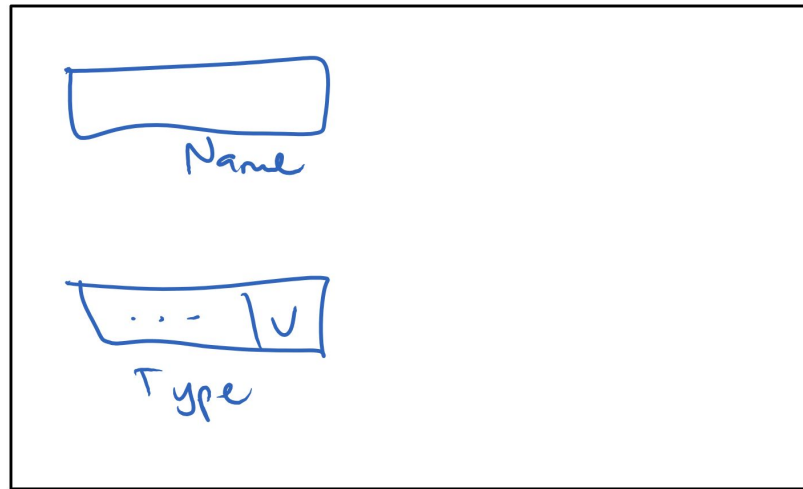


Figure 8: The create account screen of CYC.

The main screen of the app will be used for route viewing and selection. Previous routes, and WiFi and retailer locations can be viewed on the left side of the window, and filters can be selected down the bottom. This is shown in Figure 9.

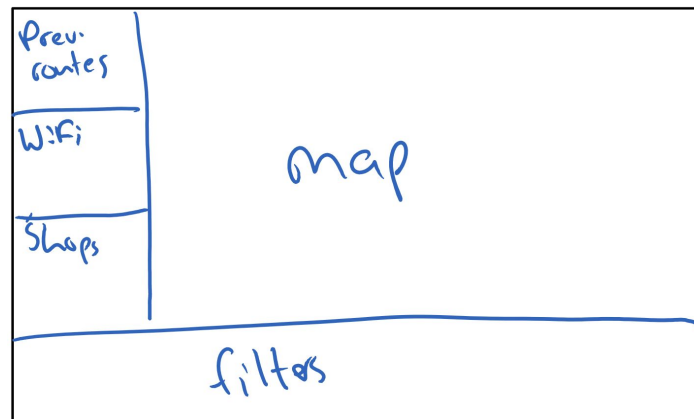


Figure 9: The main screen of CYC.

The raw trip data viewer allows users with the appropriate permissions to view trip data, and use the section at the bottom to search for and filter trips. As well as trip data, this can also be used to view WiFi location data and retailer data. This is shown in Figure 10.

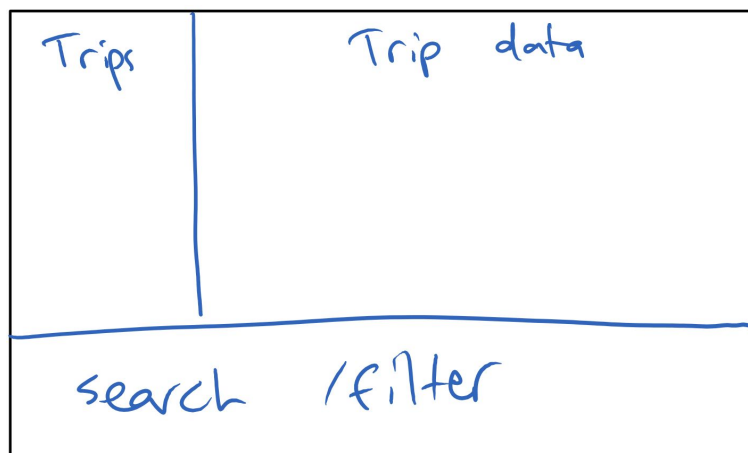


Figure 10: Raw trip data viewer.

The add POI screen can be used to add points of interest to the map, such as a cyclist adding their home and work locations, or a retailer adding a new retailer or WiFi location. This is done by selecting a point on the map and entering the point's details in the top section. This is shown in Figure 11.

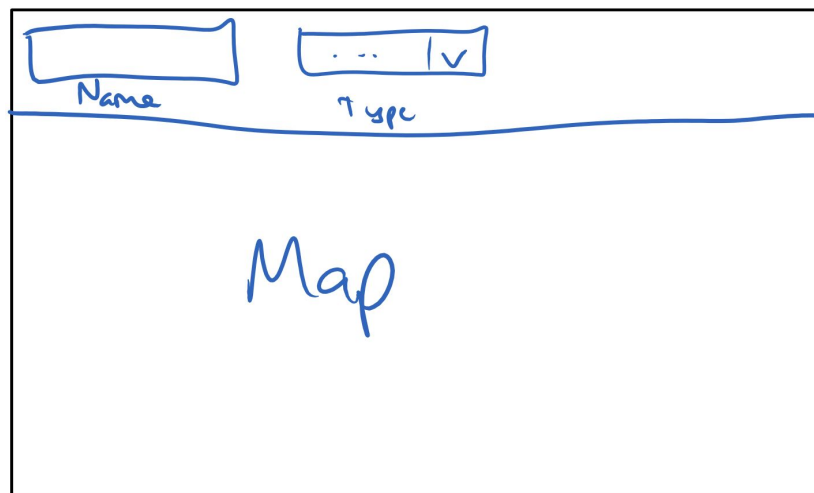


Figure 11: The add POI screen in CYC.

4.1 GUI Prototype Reflection

These GUI prototypes helped with the initial design of the scenes. Having pre-planned made it easier to plan the methods for the back end. The first GUI prototype could then be rapidly made. These also help with envisioning flow between scenes and how the user would interact with the view.

There were some missing scenes discovered as the application progressed. These were mostly landing screens, errors, alerts, and data input. These scenes are shown in figures 12, 13, 14, 15.

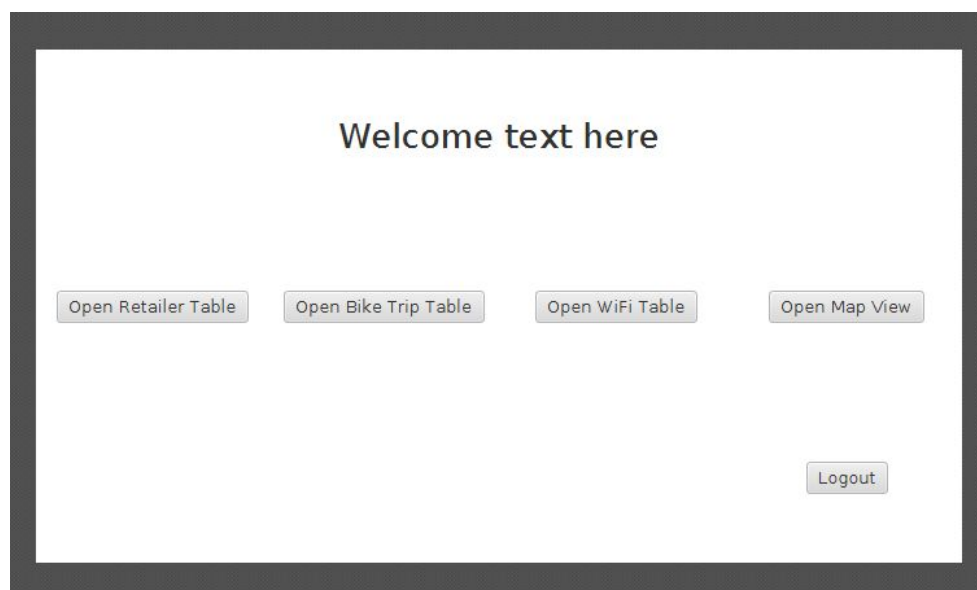


Figure 12: The Initial Application Landing screen

Add a Custom Wifi Point

WiFi Name (ssid):

Street:

Neighborhood:

ZIP:

Borough:

Latitude:

Provider:

Longitude:

Cost:

Details: (Optional)

Add

Cancel

Figure 13: Add Custom Wifi Point

Add a Custom Bike Trip

Bike ID:

Start Time: ☐ AM ☐ PM

Stop Time: ☐ AM ☐ PM

Start Date:

Stop Date:

Start Lat:

End Lat:

Start Long:

End Long:

Add

Cancel

Figure 14: Add Custom Bike Trip

Add a custom Retailer Location

Retailer Name:

Address:

Address 2:

ZIP:

Primary Function:

Secondary Function:

Add

Cancel

Figure 15: Add Custom RetailerLocation

5. Deployment Model

There are two main nodes in use for CYC: the user's device and the servers which the Google Maps API uses. The app will run on the user's device. It stores users' data in a directory structure it creates, and can pull data from custom CSVs (provided they conform to the format). The JAR contains default CSVs which it loads data from. It sends requests to and gets data from the Google Maps servers. The deployment model is shown in Figure 16.

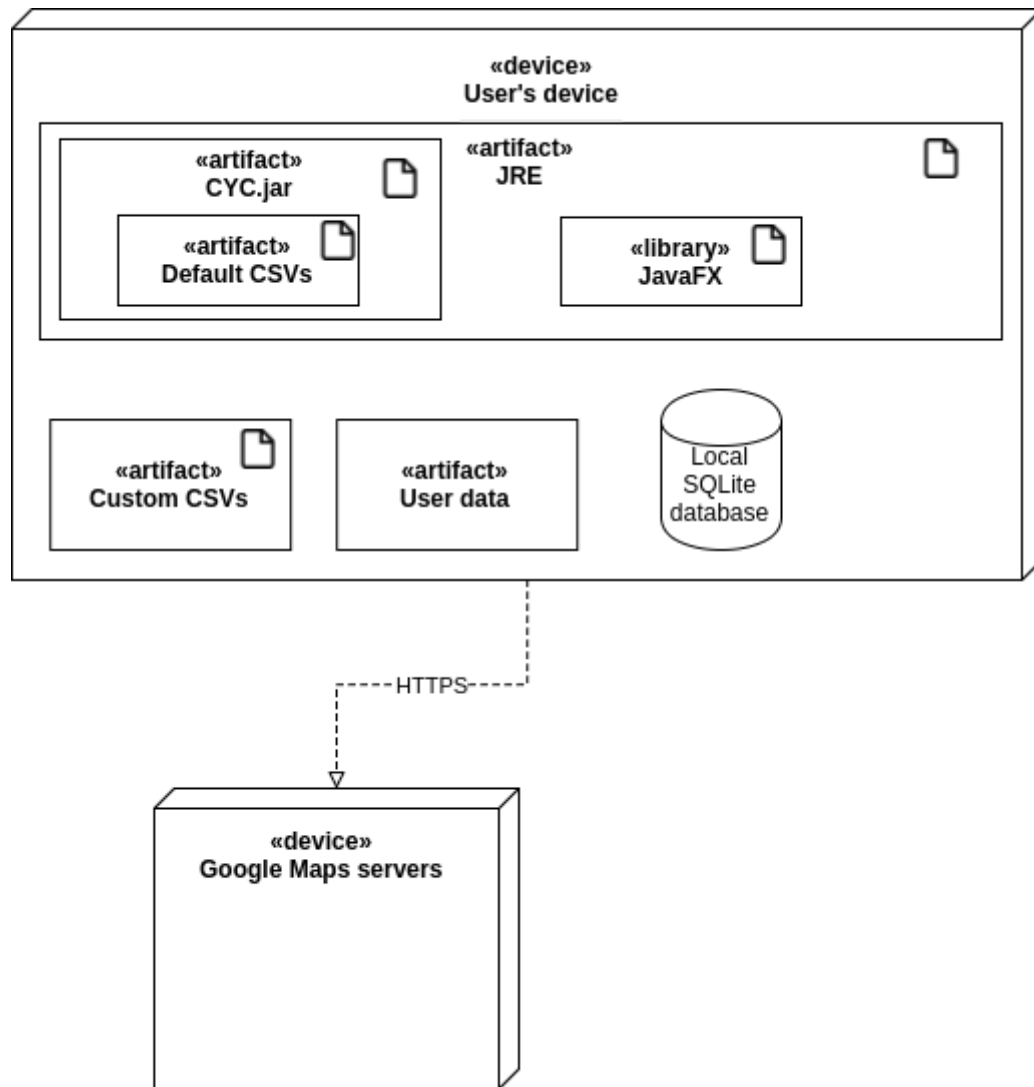


Figure 16: The deployment model for CYC.

6. Detailed UML Class Diagram

We are modelling our project on the MVC (Model View Controller) architectural design pattern. We have chosen to use MVC as it is well suited to GUI applications. It allows us to better avoid redundant code and work on separate components at the same time, without depending on other parts being completed first. An overview of the package structure of CYC is shown in Figure 17.

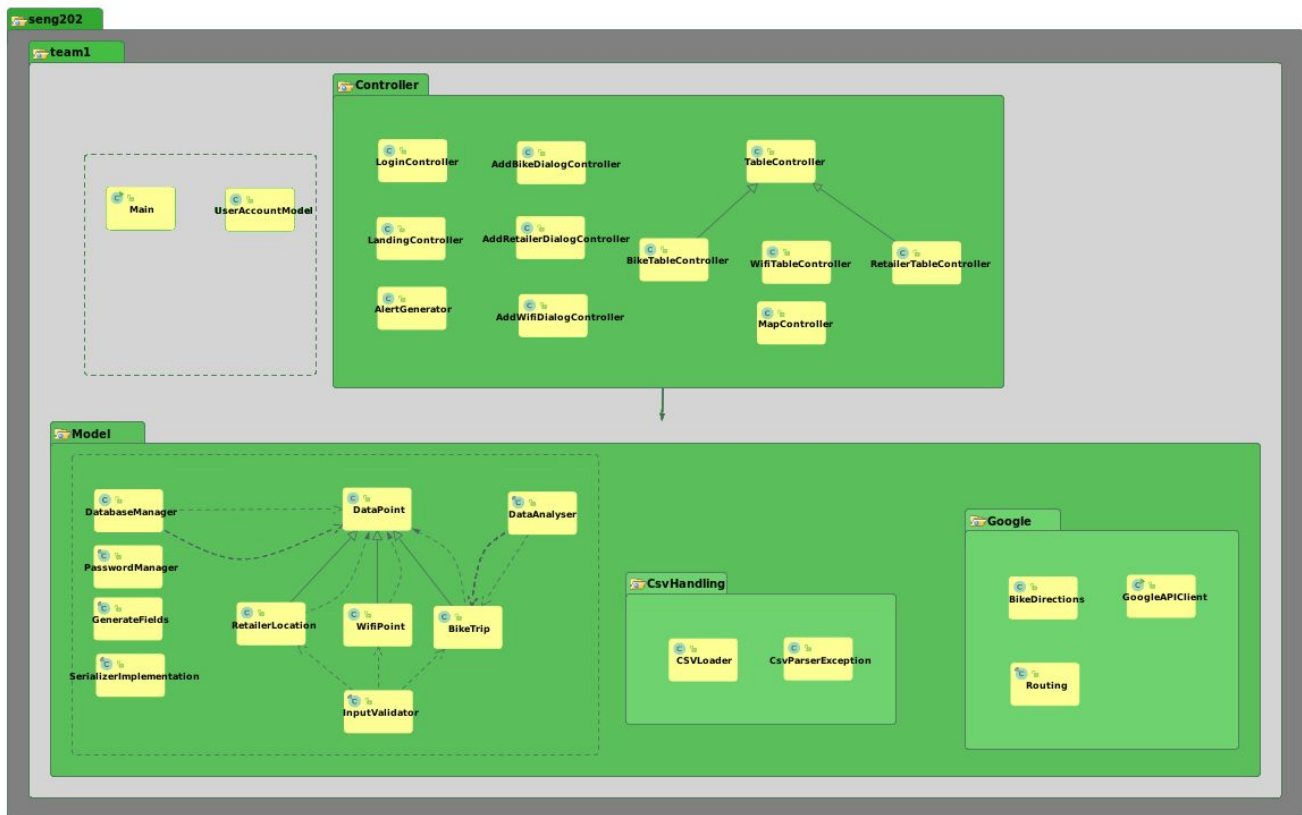


Figure 17: Package Overview

Below that are detailed diagrams of the packages included in the above diagram. These diagrams show the relationships between the classes within the packages. Of particular note is the use of inheritance in the data types within CYC, shown in figure 19 below. Also the lack of a large amount of dependency between classes and packages demonstrates the low coupling within CYC, meaning that new Views could be created in top of the model with relative ease.

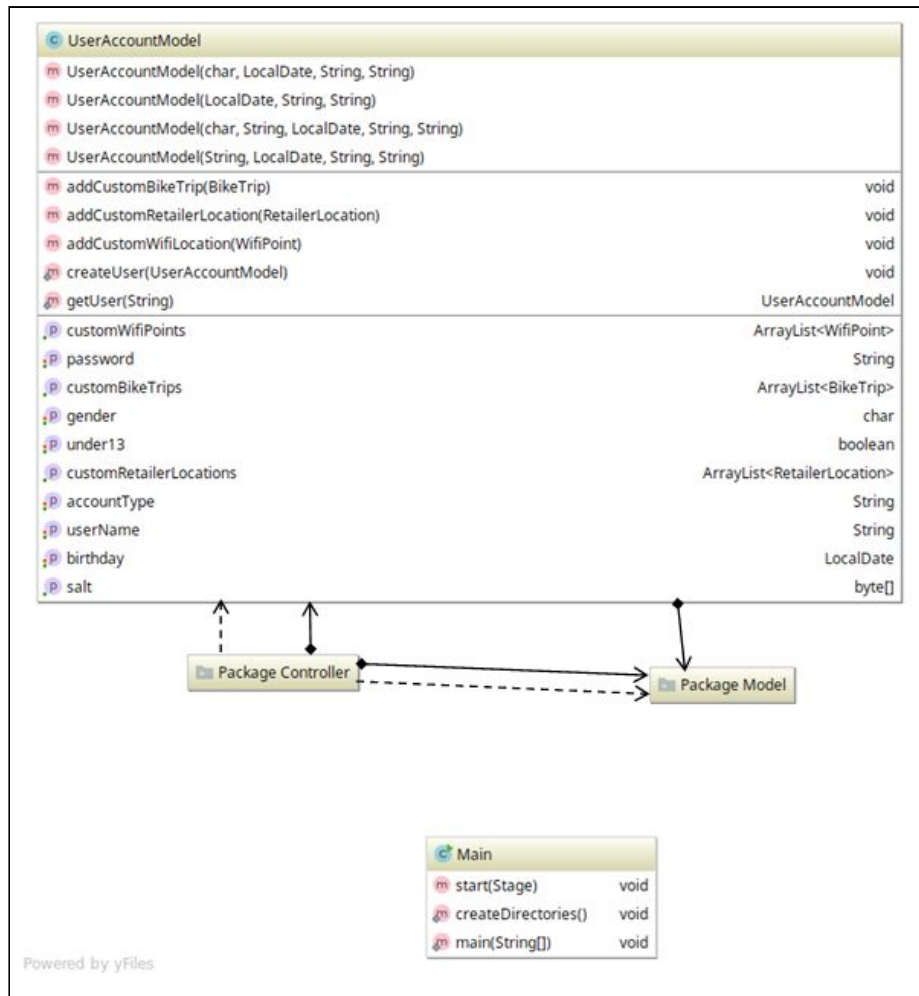


Figure 18: Main Package

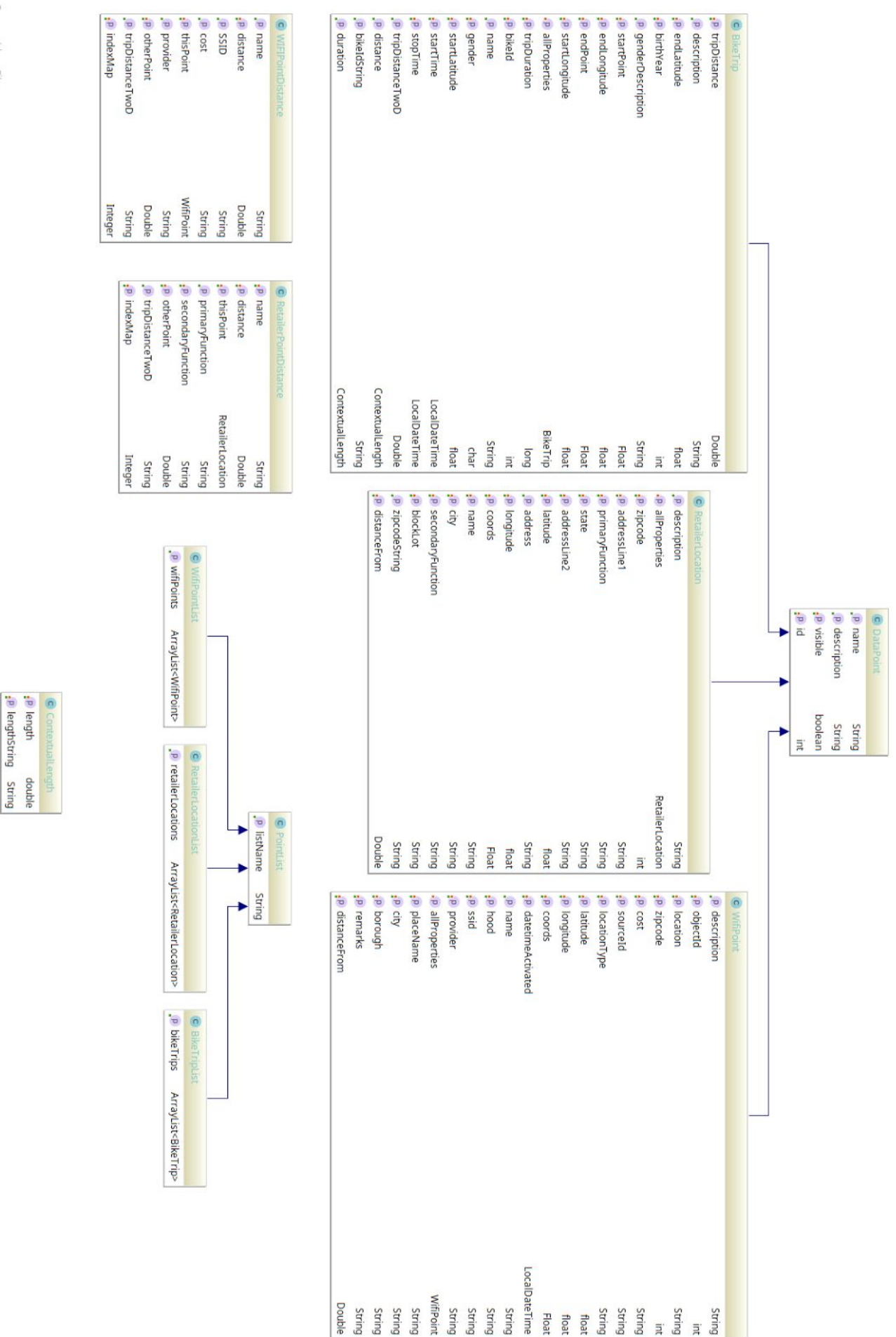


Figure 19: Model Package Part 1

DataAnalyser		
calculateDistBetweenBikeTrips(BikeTrip, BikeTrip)		double
searchBikeTrips(double, double, double, ArrayList<BikeTrip>, boolean)		ArrayList<BikeTrip>
searchBikeTripsCamsMethod(double, double, double, ArrayList<BikeTrip>, double, double)		ArrayList<BikeTrip>
searchWifiPoints(double, double, double, ArrayList<WifiPoint>)		ArrayList<WifiPoint>
searchWifiPoints(double, double, double, ObservableList<WifiPoint>)		ObservableList<WifiPoint>
sortedWifiPointsByMinimumDistanceToRoute(ArrayList<WifiPointDistance>, ArrayList<Float>)		ArrayList<WifiPointDistance>
sortedRetailerPointsByMinimumDistanceToRoute(ArrayList<RetailerPointDistance>, ArrayList<Float>)		ArrayList<RetailerPointDistance>
searchRetailerLocationsOnRoute(ArrayList<Float>, ArrayList<RetailerLocation>, double)		ArrayList<Integer>
searchWifiPointsOnRoute(ArrayList<Float>, ArrayList<WifiPoint>, double)		ArrayList<Integer>
searchWifiPoints(double, double, double, ArrayList<WifiPoint>, boolean)		ArrayList<Integer>
searchRetailerLocations(double, double, double, ArrayList<RetailerLocation>, boolean)		ArrayList<Integer>
searchRetailerLocations(Double, Double, Double, ObservableList<RetailerLocation>)		ArrayList<RetailerLocation>
searchRetailerLocations(Double, Double, Double, ArrayList<RetailerLocation>)		ArrayList<RetailerLocation>
findClosestWifiToBikeRouteStart(BikeTrip, ArrayList<WifiPoint>)		WifiPoint
findClosestWifiToBikeRouteEnd(BikeTrip, ArrayList<WifiPoint>)		WifiPoint
findClosestWifiPointToTrip(BikeTrip, ArrayList<WifiPoint>)		WifiPoint
findClosestWifiToRoute(ArrayList<Float>, ArrayList<WifiPoint>)		WifiPoint
calculateDistance(double, double, double, double)		double
sortTripsByDistance(ArrayList<BikeTrip>)		void
findClosestRetailerToBikeTrip(ArrayList<Float>, ArrayList<RetailerLocation>)		int
findClosestRetailerToWifiPoint(WifiPoint, ArrayList<RetailerLocation>)		int
findClosestWifiPointToRetailer(ArrayList<WifiPoint>, RetailerLocation)		int
findClosestWifiPointToRetailer(ArrayList<WifiPoint>, Float, Float)		int
sortWifiByDistanceFromPoint(ArrayList<WifiPoint>, Float)		void
sortRetailerByDistanceFromPoint(ArrayList<RetailerLocation>, Float)		void
findTripsByBikeId(ArrayList<BikeTrip>, int)		ArrayList<BikeTrip>
findTripsByGender(ArrayList<BikeTrip>, char)		ArrayList<BikeTrip>

DatabaseManager		
open()		void
open(boolean)		void
close()		void
inUse()		boolean
isDatabaseConnected()		boolean
deleteDatabase()		void
populateDatabaseWithDefaultValues(String)		void
getConnection()		Connection
addRecord(DataPoint, String, String)		void
addRawBikeTrip(int, Long, String, String, Float, Float, Float, Float, int, char, int, Double)		void
addRecords(ArrayList<? extends DataPoint>, String, String)		void
getBikeTrips(String, String)		ArrayList<BikeTrip>
getRetailers(String, String)		ArrayList<RetailerLocation>
getListID(String, String, Class)		int
listExists(String, String, Class)		boolean
createNewList(String, String, Class)		void
populateList(String, PointList)		void
getLists(String, Class)		ArrayList<String>
getWifiPoints(String, String)		ArrayList<WifiPoint>
deleteList(String, String, Class)		void
updatePoint(String, String, DataPoint, DataPoint)		void
deletePoint(String, String, DataPoint)		void

GenerateFields		
generatePrimaryFunctionsList(ArrayList<RetailerLocation>)		ArrayList<String>
generateSecondaryFunctionsList(ArrayList<RetailerLocation>)		ArrayList<String>
generateRetailerZipcodes(ArrayList<RetailerLocation>)		ArrayList<Integer>
generateListOfSameFunction(ArrayList<RetailerLocation>, String, boolean)		ArrayList<RetailerLocation>
generateWifiTypes(ArrayList<WifiPoint>)		ArrayList<String>
generateWifiBoroughs(ArrayList<WifiPoint>)		ArrayList<String>
findPointsOfSameCost(String, ArrayList<WifiPoint>)		ArrayList<WifiPoint>
generateWifiProviders(ArrayList<WifiPoint>)		ArrayList<String>
findWifiOfSameProvider(String, ArrayList<WifiPoint>)		ArrayList<WifiPoint>

Directory	
ROOT	
USERS	
DATABASES	
directory()	String

InputValidator		
isDuplicateBikeTrip(BikeTrip, ArrayList<BikeTrip>)		boolean
isDuplicateWifiPoint(WifiPoint, ArrayList<WifiPoint>)		boolean
isDuplicateRetailer(RetailerLocation, ArrayList<RetailerLocation>)		lean

Figure 20: Model Package Part 2

C BikeDirections	
m BikeDirections(String)	
m BikeDirections(String, boolean)	
m toString()	String
p dateRetrieved	LocalDate
p points	ArrayList<Float>

GoogleAPIClient	
m googleGeocode(String)	Double
m getRetailerGeocode()	void
m googleGetDirections(double, double, double, double)	
m main(String[])	void

Powered by yFiles

Figure 21: Google Package

CSVLoader	
m loadCSV(String)	ArrayList<CSVRecord>
m loadCSV(String, boolean)	ArrayList<CSVRecord>
m loadCSV(String, int)	ArrayList<CSVRecord>
m loadCSV(String, int, int)	ArrayList<CSVRecord>
m populateBikeTrips()	ArrayList<BikeTrip>
m populateBikeTrips(String)	ArrayList<BikeTrip>
m populateBikeTrips(String, String, String)	void
m populateBikeTrips(String, boolean)	ArrayList<BikeTrip>
m populateBikeTripsIntoDatabase(String, String, String, boolean)	void
m populateWifiHotspots()	ArrayList<WifiPoint>
m populateWifiHotspots(String)	ArrayList<WifiPoint>
m populateWifiHotspots(String, boolean)	ArrayList<WifiPoint>
m populateRetailers()	ArrayList<RetailerLocation>
m populateRetailers(String)	ArrayList<RetailerLocation>
m populateRetailers(String, boolean)	ArrayList<RetailerLocation>

CsvParserException	
m CsvParserException(String)	
p filename	String

CSVExporter	
m exportCSV(String, ArrayList<String>)	void
m exportBikeTrips(String, String, String)	void
m exportBikeTrips(String, ArrayList<BikeTrip>)	void
m exportWifiHotspots(String, String, String)	void
m exportWifiHotspots(String, ArrayList<WifiPoint>)	void
m exportRetailers(String, String, String)	void
m exportRetailers(String, ArrayList<RetailerLocation>)	void

Powered by yFiles

Figure 22: CsvHandling Package

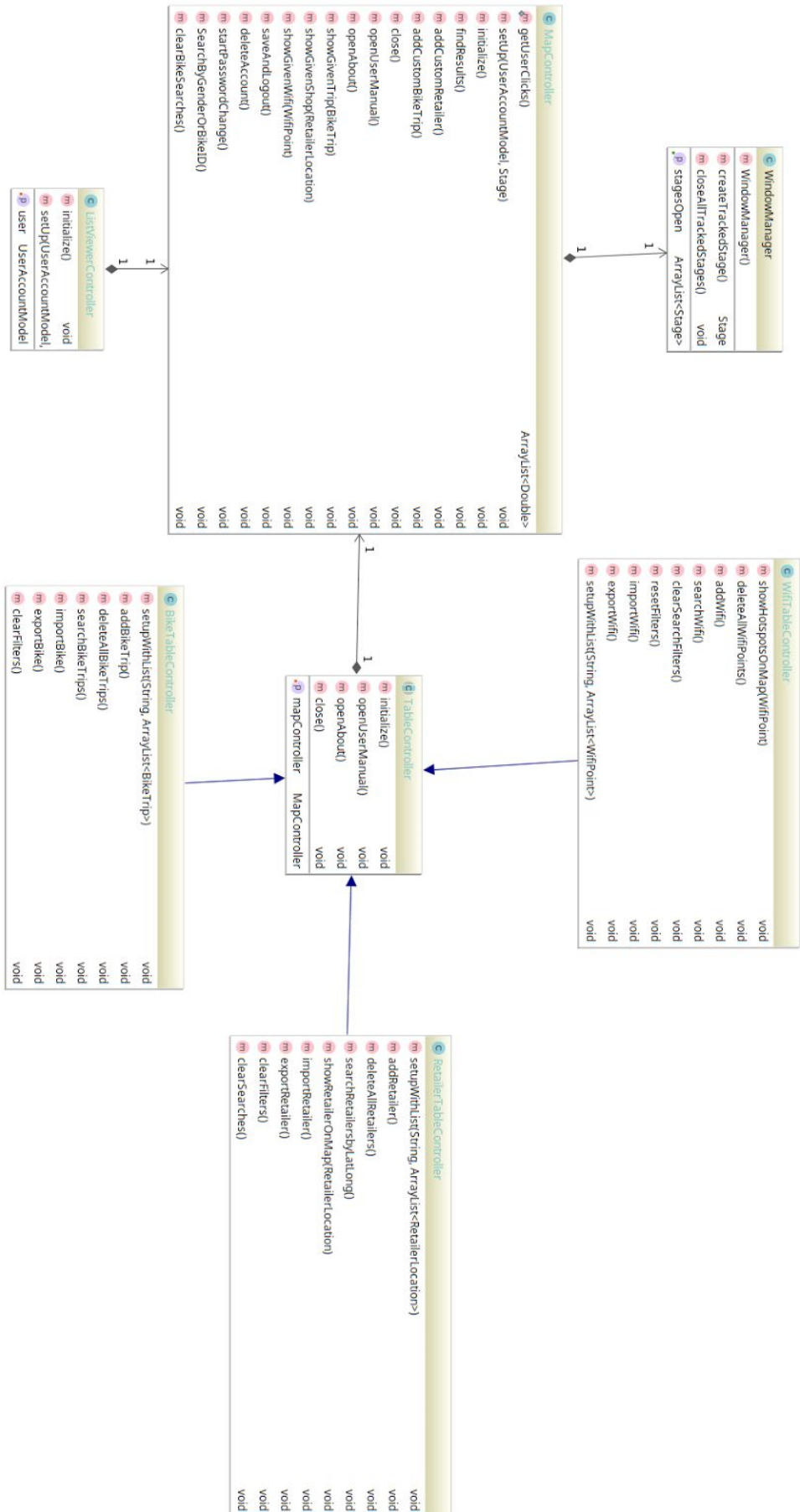


Figure 23: Controller Package

7. Risk Assessment

Risks are ranked for impact and likelihood. All risks identified are the responsibility of all team members to manage. These risks are shown in Table 8.

Table 7: Risk assessment for CYC.

ID	Description	Impact	Likelihood	Consequences	Prevention
R1	Three or more sick team members	High	Medium	Depending on timing in phase, can either result in increased workload to missed feature or late/incomplete submission.	All members to pay attention to their health and notify the team if they begin to feel sick at busy times. Team members to consider not attending meetings if they have a contagious illness.
R2	Broken code is committed to the main branch	Low	High	Application may become uncompileable or major application breaking bugs may be introduced.	All developers are to ensure that code passes local unit tests. Branch is to be tested as a whole before being merged to the main branch. Continuous integration is to be built daily. If the main branch is broken it must be fixed as soon as practical.
R3	Accidental loss of code	High	Low	Depending on timing could be severely impacting.	Members to ensure correct use of Git. Double check before deleting code. Developers to only remove branched they have merged
R4	One or two sick team members	Medium	High	Depending on timing in phase, can either result in increased workload to missed feature.	All members to pay attention to their health and notify the team if they begin to feel sick at busy times.

R5	Overestimation of amount of work able to be completed in a timeframe	Medium	Medium	Potential for multiple features to be unimplemented at submission times.	Members to notify team if they feel their workload is too much. Weekly workrate goals to be set. Weekly progress reviews to ensure project is meeting milestones.
R6	Underestimation of difficulty in implementing feature	Medium	Medium	Feature may not be possible to implement. Increased workload for other members.	Members to ask for help as early as possible if needed. Features to be basically researched before being committed to, to ensure technical feasibility.
R7	Team member unenrolling for SENG202	Medium	Low	Increased workload for remaining members.	Members to inform team as soon as possible if they are planning on unenrolling.
R8	Low quality code	Low	Medium	Increased workload for other members, rewriting and editing poor code.	Style conventions to be adhered to. Members to ask for help as early as possible if needed. Peer programming and co-locational working to be undertaken when practical. Merge requests to be undertaken by each developer once per deliverable.

8. Project Plan

8.1 Major Milestones

Deliverable 1 is due on the 15th of August.

Deliverable 2 is due on the 26th of September.

Deliverable 3 is due on the 10th of October.

A product demonstration will be conducted between the 13th and 20th of October (Learn, 2017).

8.2 Minor Milestones

Code and documentation freezes will be in place from:

- The 11th of August to the 16th of August
- The 22nd of September to the 27th of September
- The 6th of October

The feature packs as described in the assignment outline are able to be implemented as seven milestones, with each pack building on the functionality of the previous feature pack. Furthermore, additional requirements from the identified functional requirements can be added to these packs as applicable.

A full prototype version will be available for deliverable 2. Team availability is limited in the first week of term four. Weeks are prioritised based on other course commitments of the team. Four of the five team members have the same courses. Dates marked in red indicate assignments that are due for other courses (Learn, 2017), reducing the amount of available time in that week. Yellow indicates deliverable due dates and orange indicates document and code freezes. The exception to this is week seven which features three midterm tests and an assignment (Learn, 2017); this whole week will be considered unavailable. This is shown in Table 9.

Each feature package is dependent on the package before it. This will require the basic and intermediate feature for each package to be added linearly. The advanced and application specific features for each package can then be extended while the next package is developed.

Table 8: Project calendar for development of CYC.

Week	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday
1	15/07	16/07	17/07	18/07 Lecture 1	19/07	20/07 Lab 1	21/07 Lab 2
2	22/07	23/07	24/07	25/07 Lecture 2	26/07	27/07 Lab 3	28/07 Lab 4
3	29/07	30/07	31/07	01/08 Lecture 3	02/08	03/08 Lab 5	04/08 Lab 6
4	05/08	06/08	07/08	08/08 Lecture 4	09/08	10/08 Lab 7	11/08 Lab 8
5	12/08	13/08	14/08	15/08 Lecture 5 Deliverable 1	16/08	17/08 Lab 9	18/08 Lab 10
6	19/08	20/08	21/08	22/08 Lecture 6	23/08	24/08 Lab 11	25/08 Lab 12
	26/08	27/08	28/08	29/08	30/08	31/08	01/09
	02/09	03/09	04/09	05/09	06/09	07/09	08/09
7	09/09	10/09	11/09	12/09 Lecture 7	13/09	14/09 Lab 13	15/09 Lab 14
8	16/09	17/09	18/09	19/09 Lecture 8	20/09	21/09 Lab 15	22/09 Lab 16
9	23/09	24/09	25/09	26/09 Lecture 9 Deliverable 2	27/09	28/09 Lab 17	29/09 Lab 17
10	30/09	01/10	02/10	03/10 Lecture 10	04/10	05/10 Lab 19	06/10 Lab 18
11	07/10	08/10	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	17/10 Lecture 12	18/10	19/10 Lab 23	20/10 Lab 24

8.3 Progress Reviews

In addition to the weekly class standups, progress reviews shall be completed by the team every Thursday. Action points from these will be used for goal setting and planning. Specific minor tasks to be assigned in these to ensure that all major tasks for the week are achieved.

All tasks will be tracked on Trello, using the checklist function. This allows a rapid view of remaining work to be seen.

All code will be stored on the group GitLab, with code to be briefly checked by all team members weekly. All commits to the main branch must pass the continuous integration tests.

8.4 Deliverable 1

The first deliverable of this project is a planning and scope document. This phase consists of four weeks and can be broken down into 8 sub tasks. Due to the iterative nature of these tasks, they will be completed concurrently with priority given to certain tasks. As tasks will change in nature, the tentative schedule in Table 10 is used.

Table 9: Weekly plans for Deliverable 1.

Week 2	Business context, use cases, Stakeholders and requirements
Week 3	Business context, use cases, acceptance tests, stakeholders and requirements, risk assessment
Week 4	GUI prototype, deployment model, project plan for later deliverables, UML diagram and use cases
Week 5	Document freeze, proof reading and buffer time.

This schedule allows the scope and understanding of the project to grow, while allowing work done in earlier weeks to be refined. Tasks are to be divided up to individual team members earlier on with collaboration and group work to develop from week 3.

8.5 Deliverable 2

From the 16th of August the project will move out of planning and into production. This gives seven weeks to implement the required packages for deliverable 2. Time will be divided as discussed in Table 11, tasks are split from Wednesday to Wednesday as submissions are due on Tuesdays.

Table 10: Weekly plans for Deliverable 2.

Week beginning	Key tasks
16th August	Basic GUI and data import to be completed.
23rd August	Extended viewing and data filtering. Basic analysis and relations of data.
30th August	Extended loading and user data input. Persistent data storage.
6th September	Visualisations and database integration. This includes the first week of term 4.
13th September	Visualisations and database integration. This includes the first week of term 4.
20th September	Code freeze on the 22nd with Submission of deliverable 2 on the 26th. This week is used as a finishing week and over run buffer.

8.6 Deliverable 3

After the submission of deliverable two all the remaining quality requirements will be implemented. The Gantt chart shown in table 11, breaks down the remaining tasks. These have been assigned as one task per team member per day. As many of the tasks are already implemented in the back end of the project this is achievable. This leaves 11 days for work on the design document updates and as buffer time for any difficulties that occur.

Table 11: Gantt chart for deliverable 3

Functional requirement	Pre Requisite	Tu 26	We 27	Th 28	Fr 29	Sa 30	Su 1	Mo 2	Tu 3	We 4	Th 5	Fr 6	Sa 7	Su 8	Mo 9	Tu 10
FR4	-															
FR8	-															
FR18	-															
FR21	-															
FR24	FR21, FR11															
FR7	-															
FR11	FR21															
FR25	-															
FR26	FR21															
FR27	-															
FR30	-															
FR33	-															
FR34	-															
FR35	-															
FR37	-															
FR38	FR37															
FR41	-															
FR45	-															
Design Document																
Buffer time																

9. Testing Protocol and Procedures

9.1 Functional Testing

In Table 12 the classes tested are shown with their coverage, this was measured with the built in IntelliJ coverage tool using tracing. Total coverage is 25% for methods and 40% for classes, this is low for the reasons discussed in the discussion below, however coverage on the tested classes is sufficiently high.

Table 12: Coverage Statistics

Class Name	Class %	Method %	Line %	Comments
Full Application	40% (24/60)	25%(170/670	30% (1350/4390)	
BikeTrip	100% (1/1)	59% (25/42)	68% (112/164)	Is an instance class containing many getters and setters.
CSVLoader	100%(1/1)	100% (7/7)	97% (91/93)	
DataAnalyser	100% (4/4)	64% (20/31)	59% (160/271)	
DatabaseManager	100%(1/1)	87% (21/24)	87% (463/530)	
GenerateFields	100%(1/1)	66%(6/9)	71%(65/91)	
InputValidator	100%(1/1)	100%(3/3)	93%(15/16)	
PasswordManager	100%(1/1)	80%(4/5)	80%(16/20)	
RetailerLocation	100%(1/1)	55% (20/36)	59% (68/115)	Is an instance class containing many getters and setters.
SerializerImplementation	100%(1/1)	100%(2/2)	76%(33/43)	
WifiPoint	100%(1/1)	52% (23/44)	55% (72/130)	Is an instance class containing many getters and setters.
CSVExporter	100% (1/1)	100% (7/7)	97% (91/93)	

9.2 Quality Testing

The following acceptance test have been performed on the exported .jar file.

Item	Description
SUT	System
Test	AT1
Description	User is able to add an account and access the application
Result	User made account in 2 minutes and 10 seconds
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC1

Item	Description
SUT	System
Test	AT2
Description	Suggests route to a place from a place
Result	Two points on the map could be selected, route was appropriate. Additional waypoints could not be added.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC2 UC3

Item	Description
SUT	System
Test	AT3
Description	Data can be viewed and filtered in the table view.
Result	Data was in the table. Filters accurately and quickly changed data
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC10

Item	Description
SUT	System
Test	AT4
Description	CSV files can be loaded and displayed in a reasonable time.
Result	Csv was selected. Had at least 1000 result in the file. CSV opened with no noticeable delay
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC8

Item	Description
SUT	System
Test	AT5
Description	View popular routes and then accept that as the current route.
Result	Routes can be searched by point and routes from there can be seen
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC4

Item	Description
SUT	System
Test	AT6
Description	Add personal points
Result	Point was added, could be seen in the dataset but did not appear on the map.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC5

Item	Description
SUT	System
Test	AT7
Description	User can view personal route history.
Result	User can create a Unique Bike Id and view routes for that bike
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC6

Item	Description
SUT	System
Test	AT8
Description	User can add custom Wifi locations
Result	Point was selected in both views. Was added and shown on the map and table.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC6

Item	Description
SUT	System
Test	AT9
Description	Additional retailers able to be added
Result	Retailer was correctly added, searchable and able to be modified later. Was correctly shown on map.
Pass/Fail	Fail
Tester	Josh Burt
UC / Requirement	UC7

Item	Description
SUT	System
Test	AT10
Description	View popular routes and then accept that as the current route.
Result	Functionality was not implemented, no point was able to be returned
Pass/Fail	Fail
Tester	Josh Burt
UC / Requirement	UC9

Item	Description
SUT	System
Test	AT11
Description	Data Points can be modified
Result	Table view was opened correctly. Data point was selected. Point was modified. This was persistent.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC5

Item	Description
SUT	System
Test	AT12
Description	Data Points can be deleted.
Result	Data Point was selected from the table view. Data Point was deleted persistently. All data points were deleted, on selection of this option, this was persistent.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC5

Item	Description
SUT	System
Test	AT13
Description	Lists can be created
Result	A list was successfully created. Could be selected. User could populate this list with a CSV. User could populate this list with custom points.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC8, UC10

Item	Description
SUT	System
Test	AT14
Description	Lists can be selected
Result	A User in the map could select a List. Data changed itself in real time. A list viewer could be opened and a list selected. This was displayed correctly in the viewer.
Pass/Fail	Pass
Tester	Josh Burt
UC / Requirement	UC8, UC10

9.3 Discussion

All acceptance tests have been performed. Thirteen passed and one failed. The Failed test was medium priority, however this functionality is not critical to the usage of the application and a new user will not notice it's non-implementation. All high priority acceptance tests have been implemented and passed.

All view and controller classes are not complemented by unit testing. There are two reasons for this, the view classes relate exclusively to the GUI, the controller classes implement little practical change to the data. The view classes are more accurately tested through acceptance testing and visual inspection, every effort to introduce faults while testing the GUI has been made. The controller classes only pass data between the model and the view. Testing for these has been by visual inspection and logging requests as they are passed to the controller. This allow the developers to see that the controllers are being called and that data is flowing between all three components.

Model classes have appropriate unit tests. These test were written, as per team protocol, at the same time as the class was created. This has allowed the integrity of the application to be verified, as unit tests that should have already passed could be checked after a merge or substantial change. Further all commits submitted to Gitlab were built and tested in the continuous integration environment.

10. Current Product Version

10.1 Requirements and Features Implemented

The following high priority functional requirements have been implemented; this is shown in Table 13.

Table 13: High Priority Functional Requirements that have been implemented

ID	Description	Stakeholders	Priority	Use cases(s)
FR1	The application can load data about bike trips by parsing CSV files.	S1, S2, S6	High	UC10
FR2	The application can load 'WiFi locations in NYC' data by parsing CSV files.	S1, S2, S6	High	UC10
FR3	The application can load 'Lower Manhattan Retailers' data by parsing CSV files.	S1, S2, S6	High	UC10
FR5	User can calculate the distance between the start and end points of a route, as the crow flies	S1	High	UC3
FR6	Given start and end points a possible route is generated and displayed to the user	S1	High	UC2
FR12	Duplicate records are not imported to a list. The user is notified of the duplicates.	S1, S2, S6	High	UC8, UC10
FR13	User can add new trips to the list of trips.	S1	High	UC5, UC10
FR16	User can update existing records.	S1,S2, S6	High	UC10
FR17	User can either log in to an existing user account or create a new account.	S1, S2,S6	High	UC1
FR19	Accounts have default 'bike trip', 'WiFi' and 'retailer' lists when created.	S1, S2, S6	High	UC1
FR20	The data visualization screen has a window that displays a map of New York City.	S1	High	UC3, UC4, UC6
FR22	WiFi locations can be displayed on the map.	S1	High	UC3, UC9
FR23	Retailer locations can be shown on the map.	S1, S2	High	UC3,UC9

The following medium and low priority requirements have been implemented

Table 14: Medium Priority Functional Requirements

ID	Description	Stakeholders	Priority	Use Case(s)
FR14	User can add new WiFi locations to the list of WiFi locations.	S1	Medium	UC5, UC11
FR15	User can add new unique retailers to the list of retailers. Duplicate retailers will not be accepted.	S1, S2	Medium	UC5, UC7
FR28	The raw data viewer lists data for bike trips, WiFi hotspots, and retailers, each in their own separate table.	S1, S2	Medium	UC10
FR29	For each record on each list, basic details are shown by default. Upon opening a record, further details are displayed.	S1, S2	Medium	UC10
FR31	User can filter WiFi location data by borough, type, and provider.	S1, S2	Medium	UC10
FR32	User can filter retailer data by street and ZIP code.	S1, S2	Medium	UC10
FR36	User can sort bike trips based on distance.	S1	Medium	UC10

FR40	When a user adds data to their lists, it is persistent.	S1	Medium	UC6, UC10
FR44	When a user attempts to log in to an account, the password they have entered is checked against the hash of the password for that account; if successful, they are logged in.	S1	Medium	UC1

Using the results shown in the tables in the Requirements and Features Not Implemented section, we can see the following use cases in table 14, have either partial support, total support or no support. Total support is defined as all requirements relating to the use case being implemented. Partial support is defined as some of the requirements of a use case being implemented. No support is defined as no requirements for the use case being implemented.

Table 15: Support Level of Use Case Diagrams

ID	Use case	Support Level
UC1	Create new cyclist account	Total Support
UC2	Suggest route	Total Support
UC3	Plan point to point route	Total Support
UC4	View popular routes and select it as current route	Partial Support
UC5	Add personal points of interest to a user account	Partial Support
UC6	View route history for a current user	Partial Support
UC7	Add a new retailer to the map	Partial Support
UC8	Open a custom CSV data file	Partial Support
UC9	Search for the next closest service to another service's location	Partial Support
UC10	View, load and filter CSV data in "raw data viewer"	Total Support
UC11	Add a new WiFi location	Partial Support

10.2 Features We Are Most Proud Of

One feature we are proud of is the map view, clustering and points of information placement. The clusters look nice and enhance the usability of the application. The map view is easy to use and the points of information are well placed. The hover over for more information provides more functionality and the filters work well.

Show on map from the tables. The right click interface is easy to use. Being able to see a route, Retailer or WiFi Hotspot on the

The final feature we are most proud of is general ease at which the application can be used. The GUI design is functional, responsive and straightforward. As usability is our highest ranked key driver, it was important to the stakeholders that a easy to use application was delivered.

10.3 Requirements and Features Implemented In Phase 3

The high priority requirements shown in table 16 have been implemented in Phase 3.

Table 16: High Priority Requirements to still be Implemented.

ID	Description	Stakeholders	Priority	Use Case(s)
FR4	User can calculate the distance between two waypoints of a bike trip, based on the difference between their longitude and latitude.	S1	High	UC3
FR8	User can enter a start point and request popular routes. The most common routes taken from this starting location should then be displayed for the user. The user can then select one of these routes.	S1	High	UC4
FR18	User can create an account by accepting the terms of service.	S1, S2, S6	Medium	UC1

FR21	Bike trips can be displayed visually on the map.	S1	High	UC3, UC4, UC6
FR24	The retailers along a bike trip can be displayed on the map, with their distances ranked.	S1, S2	High	UC3, UC9

The medium and low priority use cases, shown in table 17, have been implemented in Phase 3.

Table 17: Medium and Low Priority Functional Requirements

ID	Description	Stakeholders	Priority	Use Case(s)
FR7	User can find the closest WiFi hotspot to a given retailer.	S1, S2	Medium	UC9
FR11	User can find the closest retailers to a bike route.	S1, S2	Medium	UC2, UC9
FR25	The WiFi access points near a retailer can be displayed on the map, with their distances ranked.	S1, S2	Medium	UC3, UC9, UC11
FR26	The WiFi access points along a bike trip can be displayed on the map with their distances ranked	S1, S2	Medium	UC3, UC9
FR30	User can filter bike trip data by start point, end point, bike ID, or gender.	S1	Medium	UC10
FR33	User can search bike trips based on	S1	Medium	UC10

	search criteria: either absolute search criteria (e.g. 'start location X'), or ranges (e.g. 'start location between X and Y').			
FR34	User can search WiFi locations based on search criteria: either absolute search criteria (e.g. at location X'), or ranges (e.g. located between X and Y').	S1	Medium	UC10
FR35	User can search retailers based on search criteria: either absolute search criteria (e.g. at location X'), or ranges (e.g. located between X and Y').	S1, S2	Medium	UC10
FR37	When a new data set is imported, the data set can be added to an existing list of its type, or create a new list of said type.	S1	Medium	UC8, UC10
FR38	When multiple lists are associated with an account, the user can select between lists within the information panel.	S1	Medium	UC10
FR45	User can export their lists as CSV files.	S1, S2	Low	UC10

10.4 Requirements and Features Not Implemented

The following features, shown in table 18, were not implemented.

FR27 was not implemented as our CSV parser module, Apache CSV parser did not support reading a single line to the file. We felt changing the entire structure of that module was not feasible in the time period for Phase 3.

FR41 was not implemented as our local database was not implemented in Phase 2. In the two week time period we felt that this was a large amount of work and as it was not central to the design of our application it would likely stop more important requirements from being implemented. This requirement should have been rated low priority.

Table 18: Features

ID	Description	Stakeholders	Priority	Use Case(s)
FR27	User can load CSV data with an alternate number of attributes per entity/line as defined by the user at runtime.	S1	Medium	UC8
FR41	Bike trip, WiFi, and retailer data can be exported over a network to a central database.	S1, S2	Medium	UC5, UC7, UC11

11. Key Lessons Learned

11.1 Cameron Auld

- Use cases are an effective way to get a clear idea of the functional requirements that are actually required for the project.
- When working with new/unfamiliar technologies, in this case JavaScript and the Google Maps API, then you may have revise your requirements based on limitations in the technology. Good research into the technology, in this case the Google Maps API, partly mitigates this.
- When I logged my time, it helped me by making me realise what I should actually be working on and whether I was making good use of my time.

11.2 Josh Bernasconi

- Logging the time I spend on a task greatly increases my productivity and concentration because I don't want to feel like I'm logging time that isn't actual work. I also think that being able to see how much time the rest of the team is putting in helps to keep the work load balanced.
- A large project like this would have been much harder to finish if we didn't have the planning we did. Having concrete FRs to implement and test makes the development more focused and stops a bit of feature creep.
- Keeping an idea of the big picture of the app when developing a small piece of code helps reduce the amount of refactoring needed when a new feature is added and changes how the app flows.

11.3 Josh Burt

- When planning the initial project, half of what you want to do is actually doable in the time. Making concrete Use Cases would have helped narrow down what really needed to be done.
- Functional Requirements are key to knowing what to do. When they are correctly prioritised they make life so much easier. Prioritizing and setting these correctly at the start is a major change I'll make next project.
- Not doing only the project from one angle. I worked mainly on the backend at the start of the project and had to learn a lot when the GUI tasks were more important later on. I should have implemented features fully (backend, GUI) at the start to minimize this.

11.4 Ollie Chick

- It was useful to set standard dimensions for GUI elements to ensure the GUI had a consistent look throughout.
- Concrete code freeze dates help make sure that any bugs can be eliminated before the deadline.

11.5 Ridge Nairn

- Planning a database schema beforehand and considering how it will be used is crucial to make sure that the implementation is efficient for the type of data being stored.
- Communicating effectively what is currently being worked on, what needs to be done and what has been completed is extremely important.
- Explaining thought processes and code to others really helps to visualise abstract concepts, understand better what is going on and thinking about how to improve it.

Change Log Phase 2

The following sections have been changed significantly:

- Executive Summary
- 2 Stakeholders
- 3 Acceptance Tests
- 5 Deployment model

- 6 Detailed UML Diagram

The following subsections have been significantly changed:

- 1.0 Business and System Context
- 1.2 Market Research
- 1.5 System Context
- 1.6 Future Business Context
- 4.1 GUI Reflection
- 8.6 Deliverable 3
- 7.R2

The following sections have been added:

- 9. Testing Protocol and Procedures
- 10. Current Product Version
- Change Log
- Additional Comments

Change Log Phase 3

The following sections have been added:

- 3 Acceptance Tests have been added.
- 6 UML Updated
- 9. Testing Protocol and Procedure
- 10.2 Changed What we are proud of
- 10.3 added
- 10.4 added
- 11. Key Lessons Learned.

References

1. GoldenCheetah. (2017) Retrieved from www.goldencheetah.org
2. Strava. (2017) Retrieved from www.strava.com
3. MapMyRide. (2017) Retrieved from www.mapmyride.com
4. Learn. (2017) Retrieved from www.learn.canterbury.ac.nz