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

26 September 2017

Table of Contents

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.).

1.2 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.

1.5 System Context

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

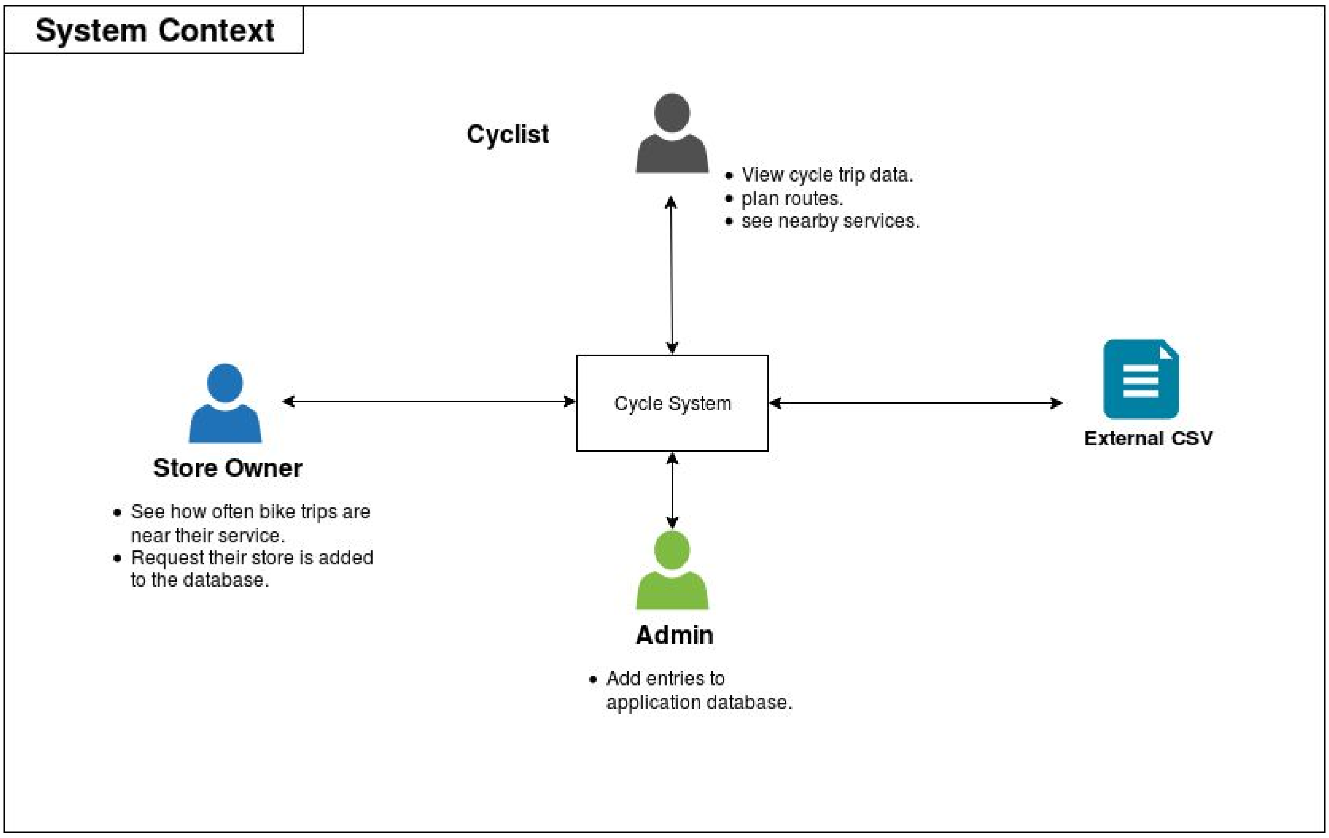

System Context
Cyclist
• View cycle trip data.
• plan routes.
• see nearby services.
Cycle System
External CSV
Store Owner
• See how often bike trips are near their service.
• Request their store is added to the database.
Admin
• Add entries to application database.

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.

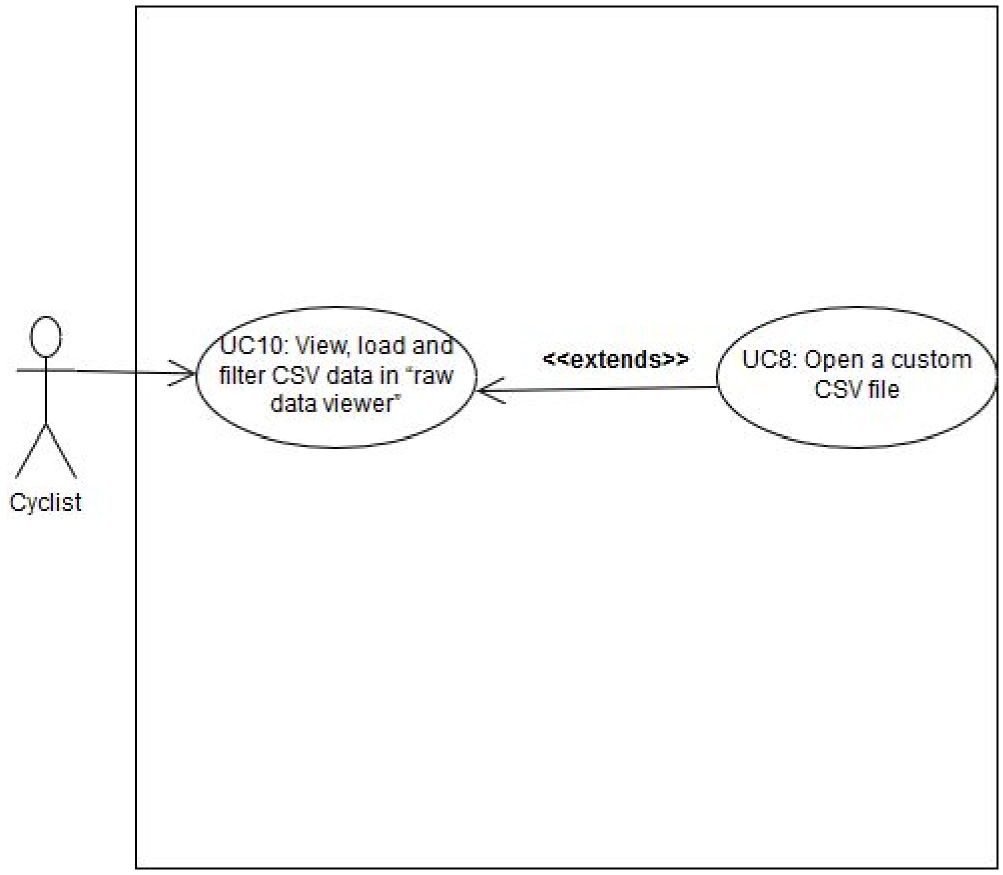

Cyclist
UC10: View, load and filter CSV data in "raw data viewer"
<<extends>>
UC8: Open a custom CSV file

Figure 3: Use cases for CSV Files.

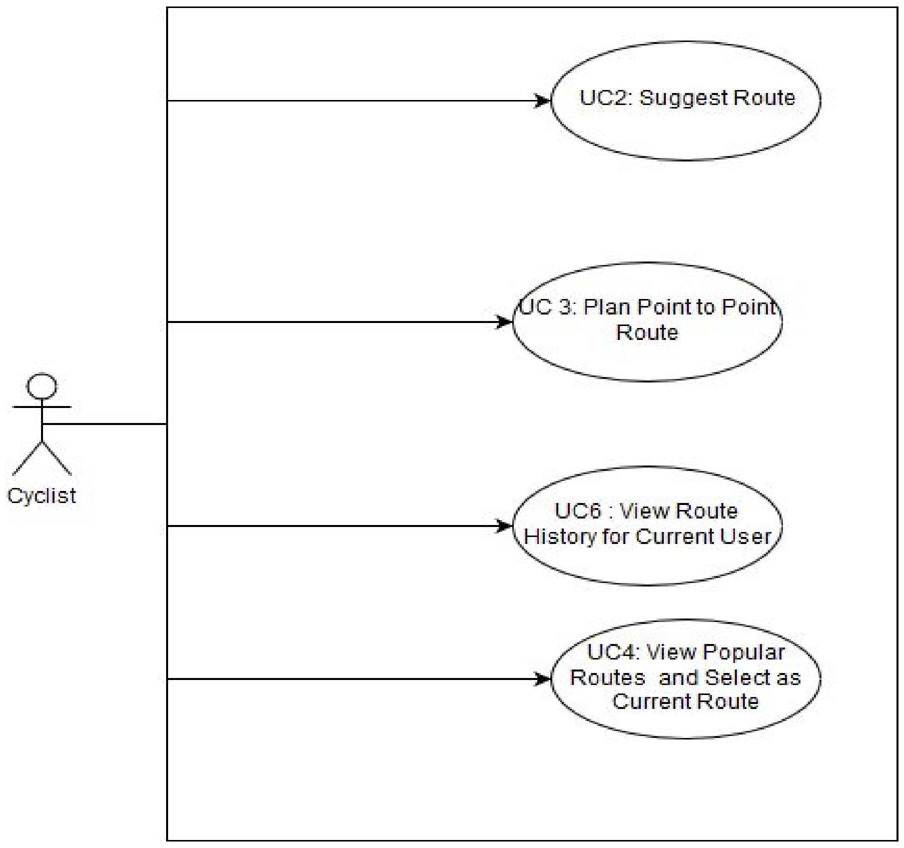

UC2: Suggest Route
UC 3: Plan Point to Point Route
Cyclist
UC6 : View Route History for Current User
UC4: View Popular Routes and Select as Current Route

Figure 2: Use case Diagram for Route Planning

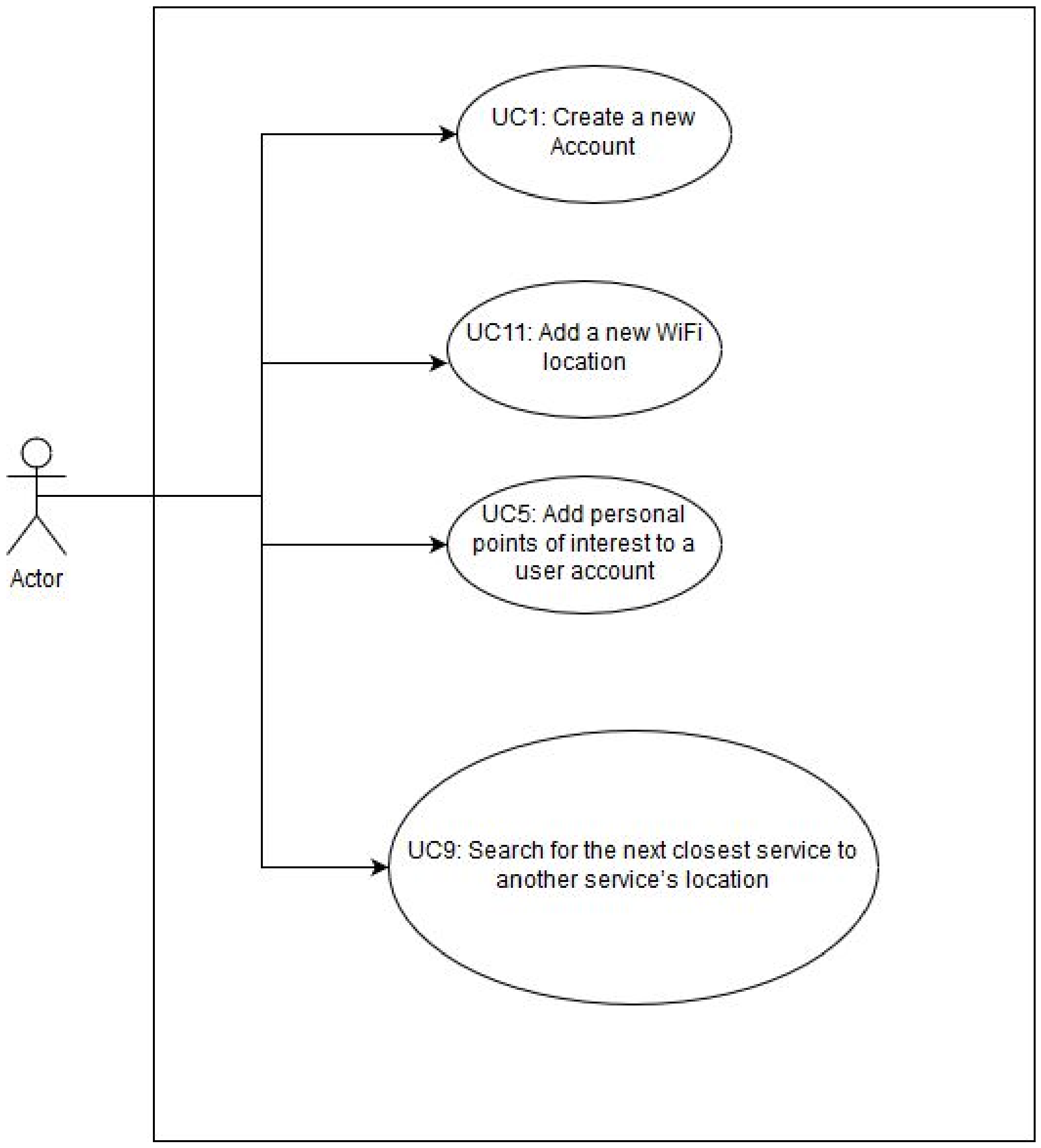

UC1: Create a new Account
UC11: Add a new WiFi location
Actor
UC5: Add personal points of interest to a user account
UC9: Search for the next closest service to another service's location

Figure 3: Use Case Diagram for Account Lifecycle

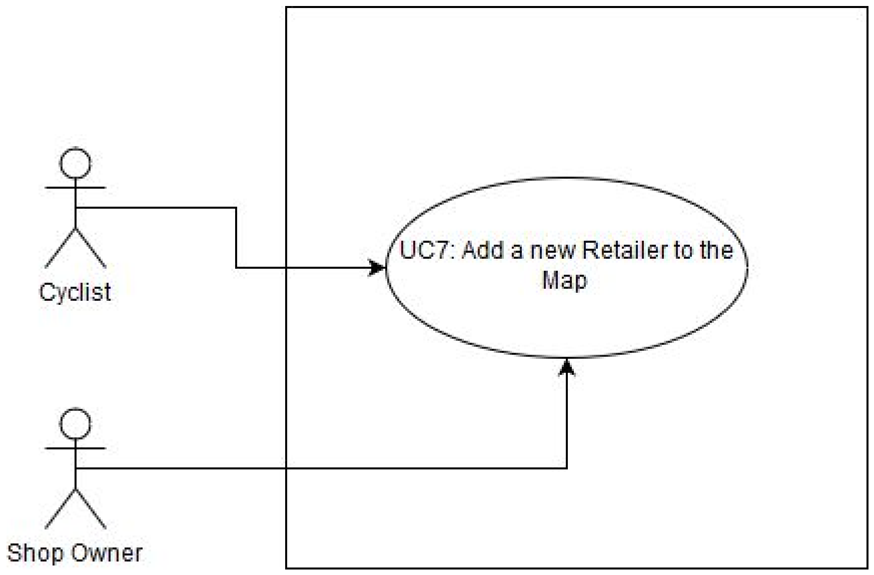


Figure 4: Adding a Retailer to the Map View Use Case Diagram

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	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.	1. Request is sent to the account handling method 2. Validity or uniqueness check fails 3. User is shown appropriate error message
UC2	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.	1. Location has no routes available to show. 2. User is informed.
UC3	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.	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	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.	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	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.	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	1. User requests history 2. Database is queried 3. Last 10 trips are returned	User history is available to the application to be displayed.	1. User requests history 2. Database is queried 3. No history is found and user is alerted

UC7	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.	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	1. CSV file is opened 2. Data is made available to the display module	Display module is able to manipulate this data.	1. CSV file is unable to be opened 2. Buffer is purged 3. User is shown an error message
UC9	1. User location is retrieved 2. Database is queried and nearest WiFi location is returned	WiFi location is able to be routed to.	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	1. CSV file is opened 2. Data is made available to the filter module	Data is added to the dataset.	1. CSV file is unable to be opened 2. Buffer is purged 3. User is shown an error message
UC11	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.	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: 1. Less than 200 ms for basic UI interaction and single local database fetches 2. Less than 1000ms for single remote database fetches and logins 3. 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: 1. Create an account and login in less than 1 minute 2. Create a route in less than 3 minutes 3. 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: 1. New Zealand 2. United States of America 3. 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 given 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	Admin account is able to add new retailers as waypoints to the application.	Low	UC7

4. GUI Prototypes

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.

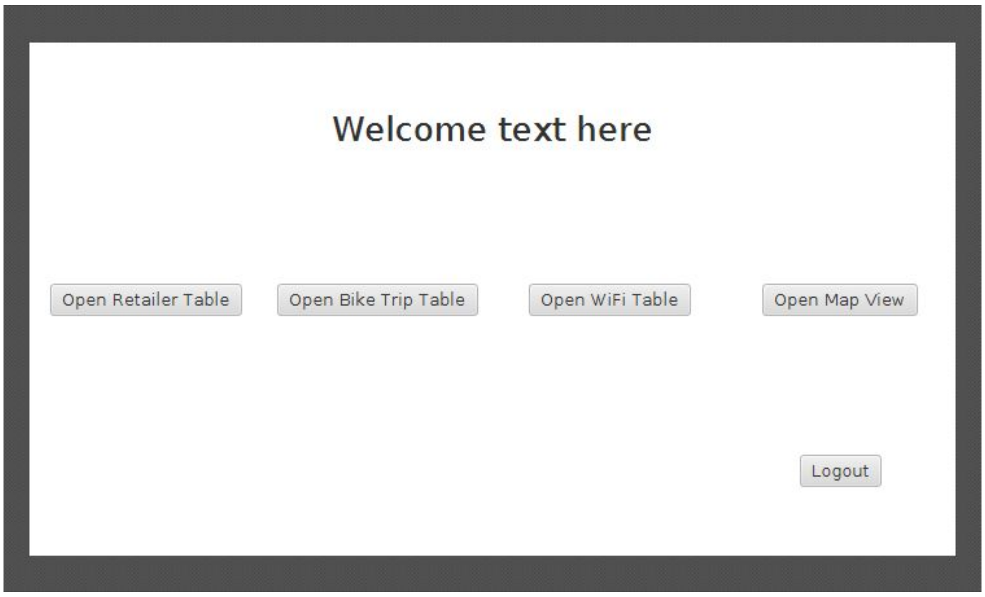
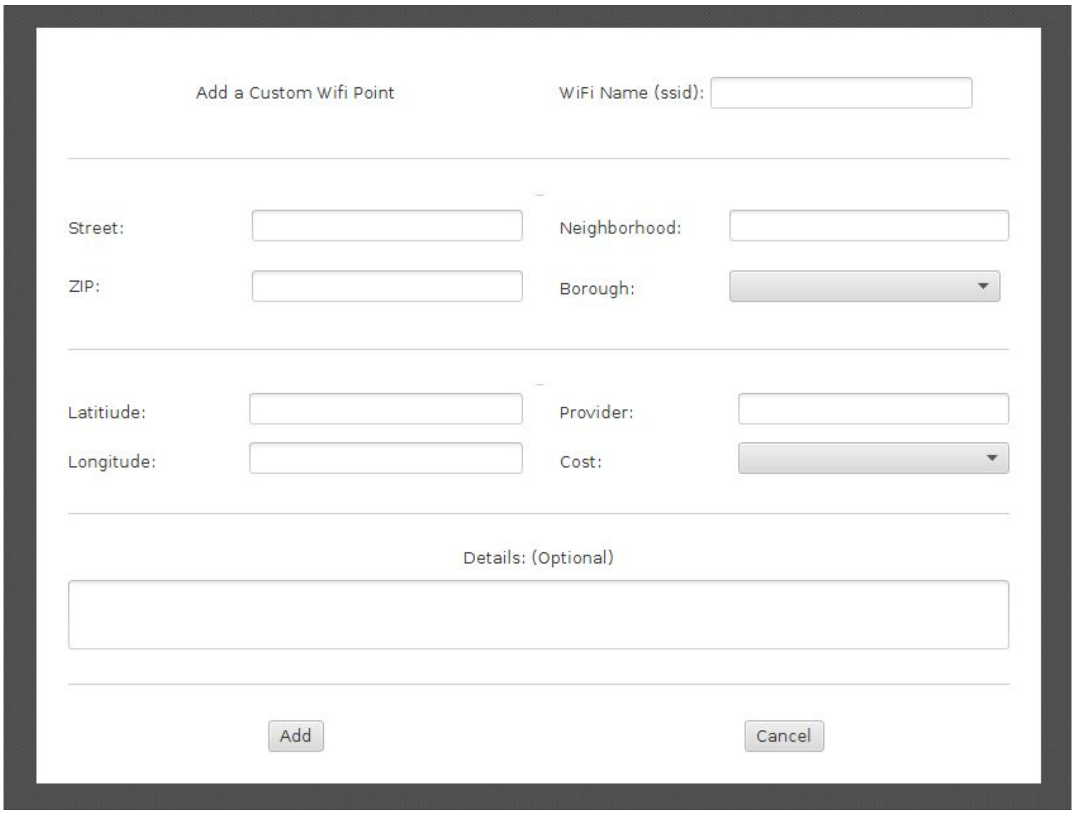


Figure 10: The Initial Application Landing screen

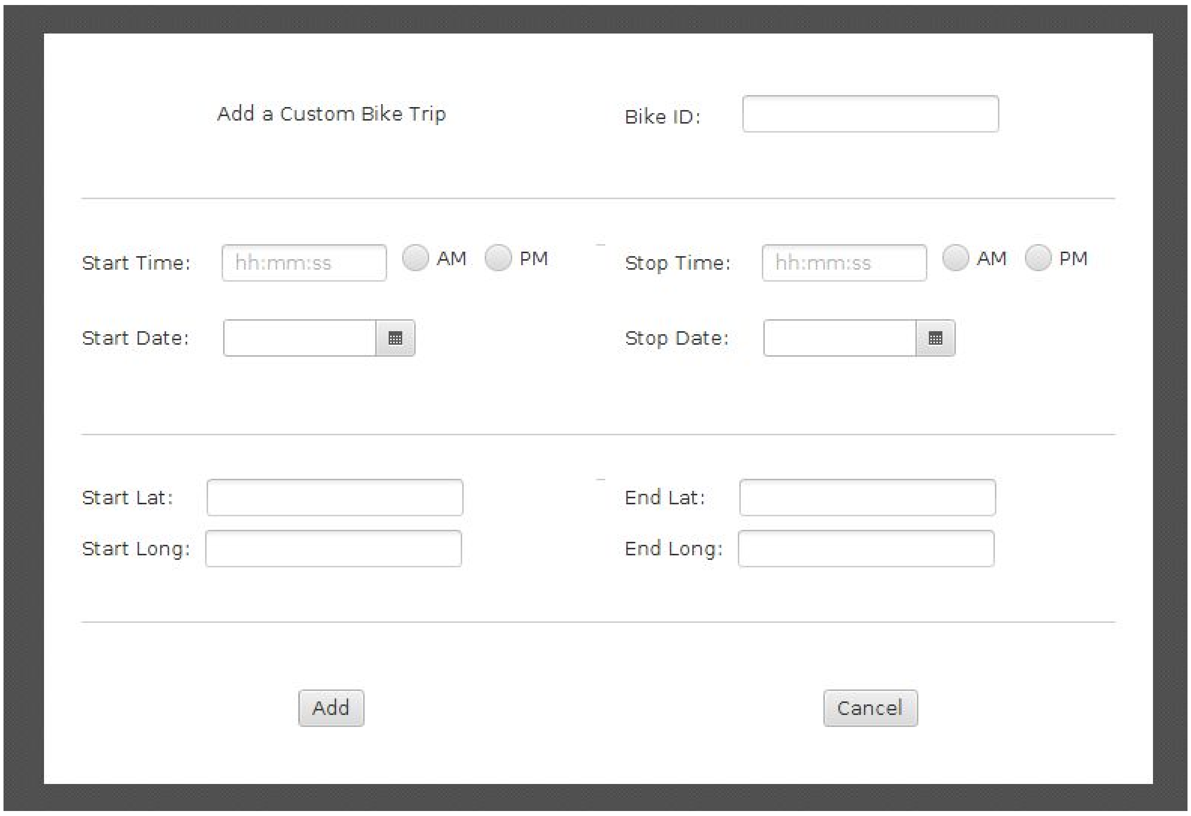


Add a Custom Wifi Point		WiFi Name (ssid):	
Street:		Neighborhood:	
ZIP:		Borough:	
Latitiude:		Provider:	
Longitude:		Cost:	

Details: (Optional)

Add Cancel

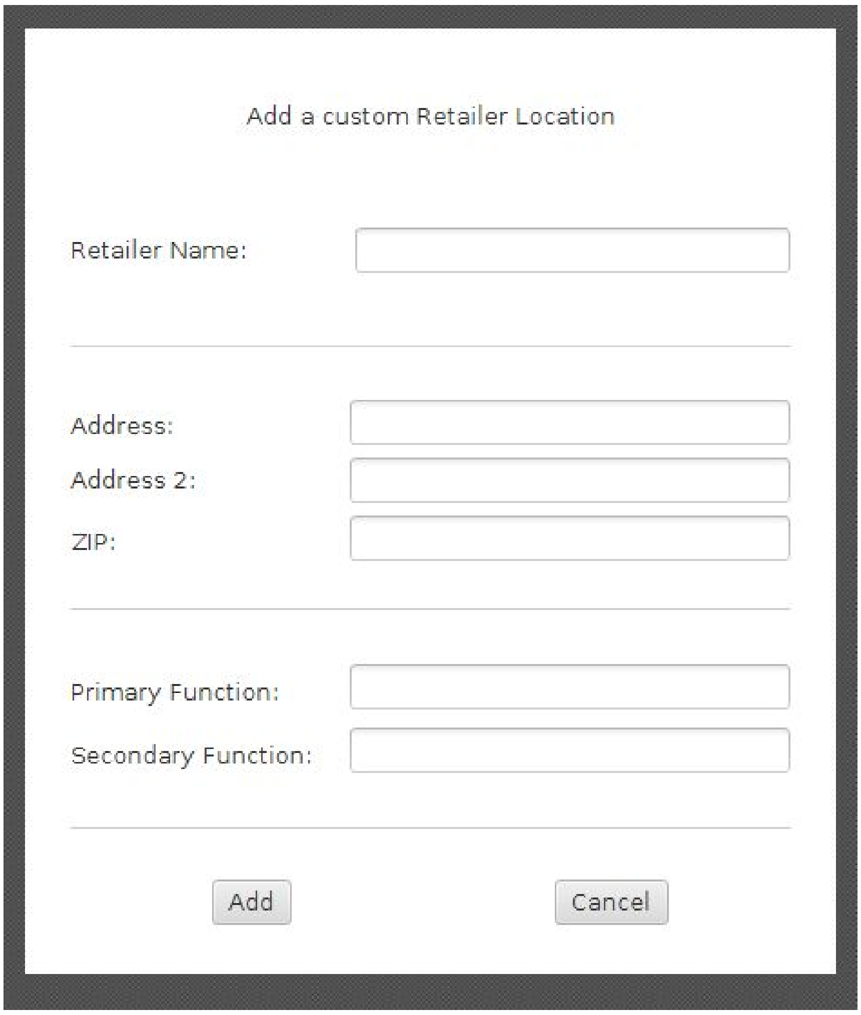
Figure 11: Add Custom Wifi Point



Add a Custom Bike Trip		Bike ID:	
Start Time:	hh:mm:ss AM PM	Stop Time:	hh:mm:ss AM PM
Start Date:		Stop Date:	
Start Lat:		End Lat:	
Start Long:		End Long:	

Add Cancel

Figure 12: Add Custom Bike Trip


Add a custom Retailer Location

Retailer Name:

Address:

Address 2:

ZIP:

Primary Function:

Secondary Function:

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 13.

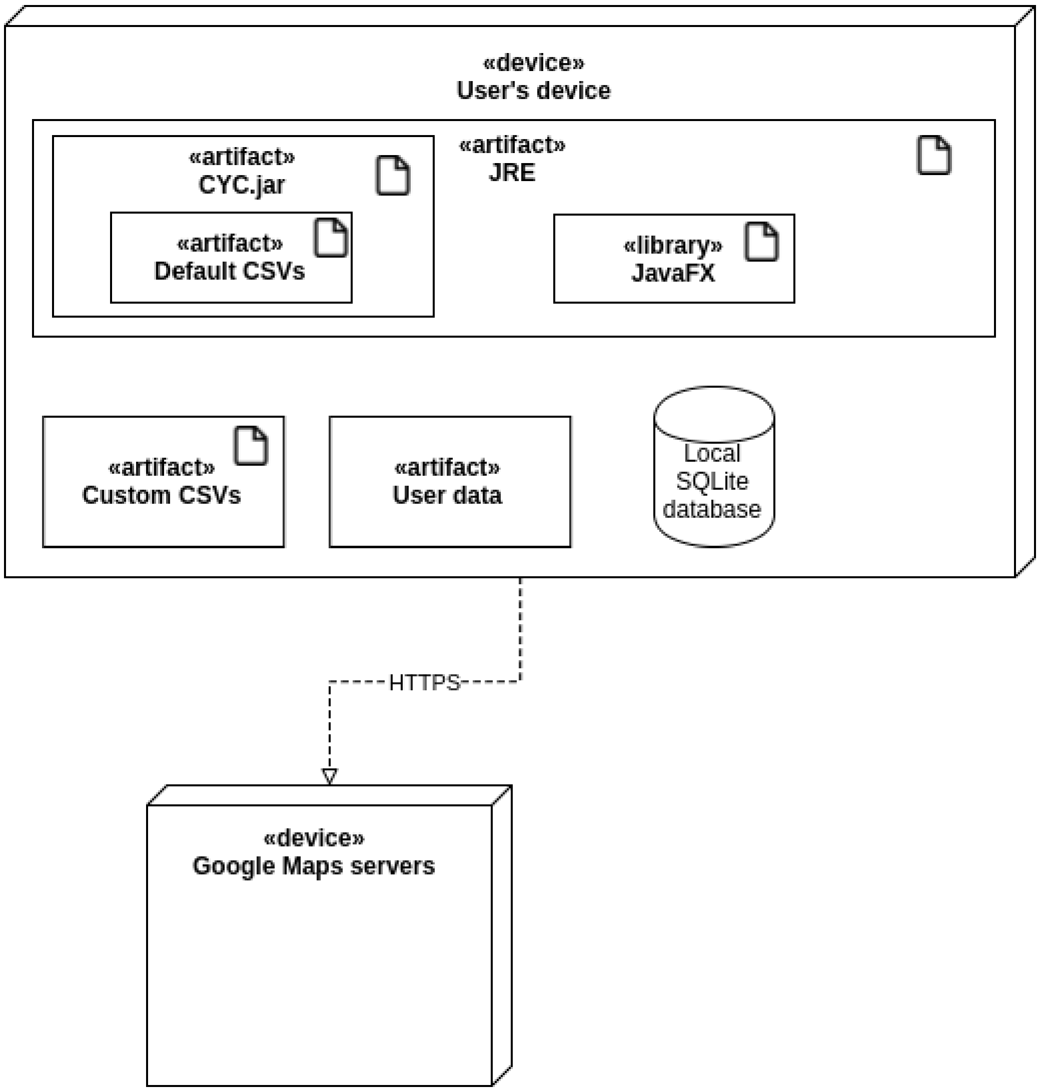


Figure 13: The deployment model for CYC.

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 16.

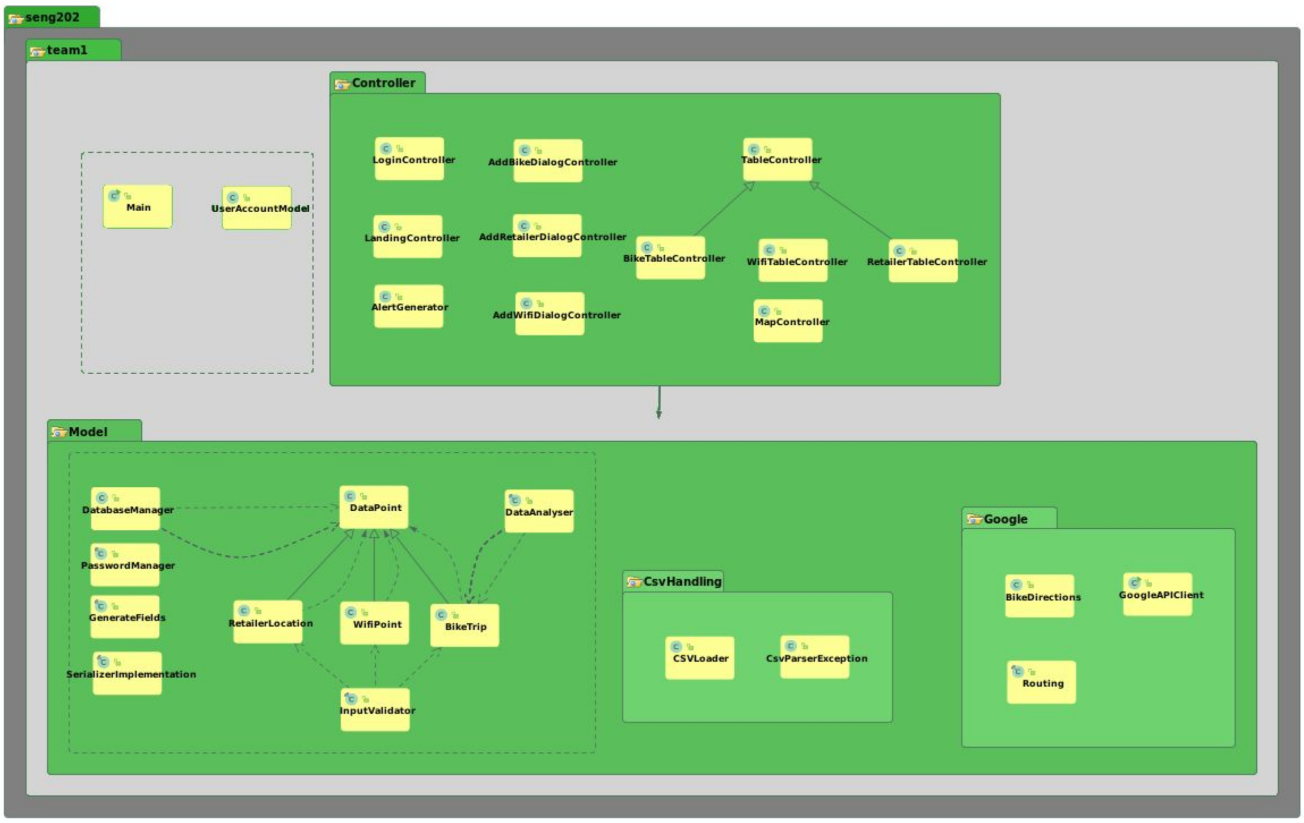
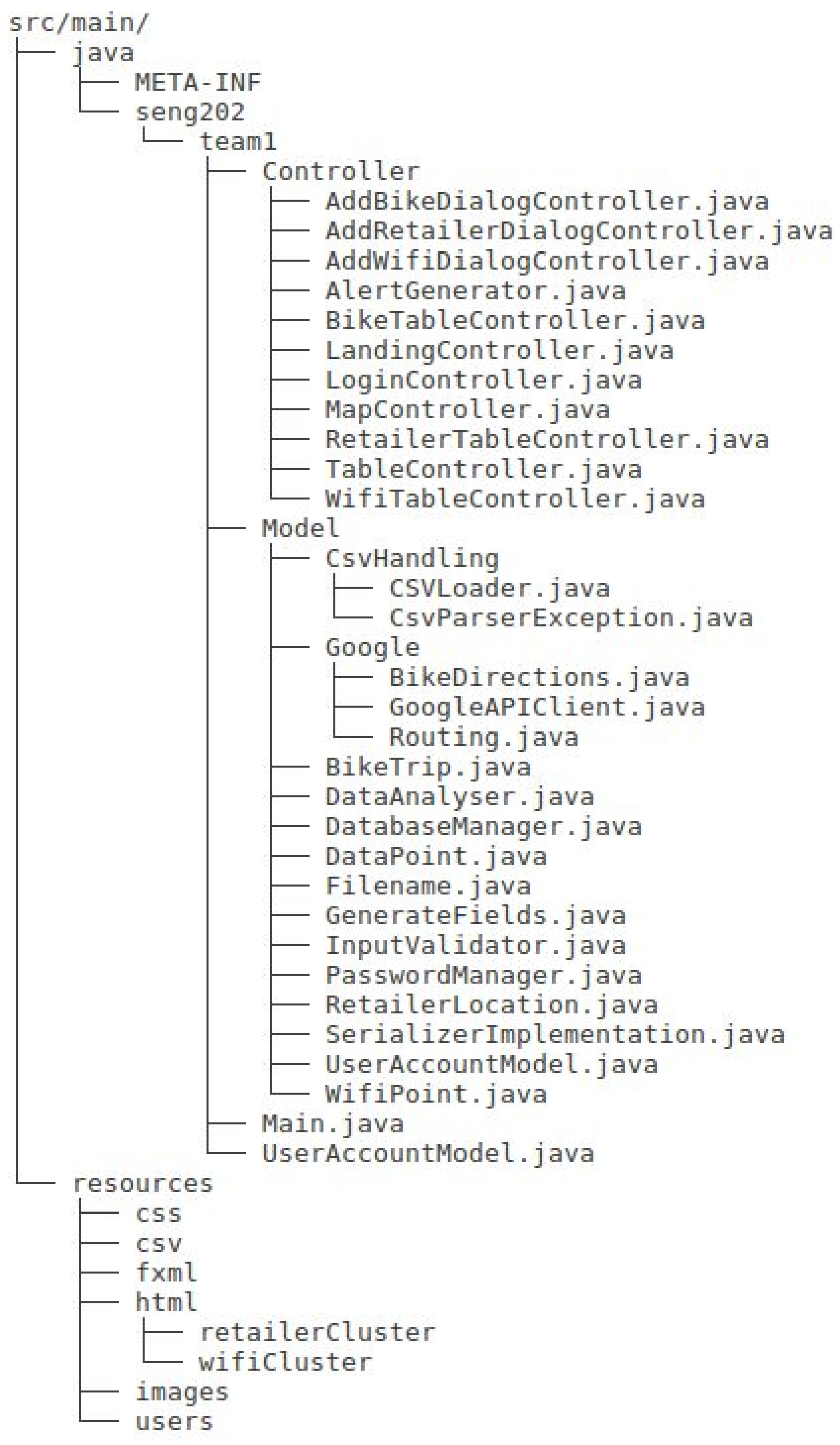


Figure 13: Package Overview

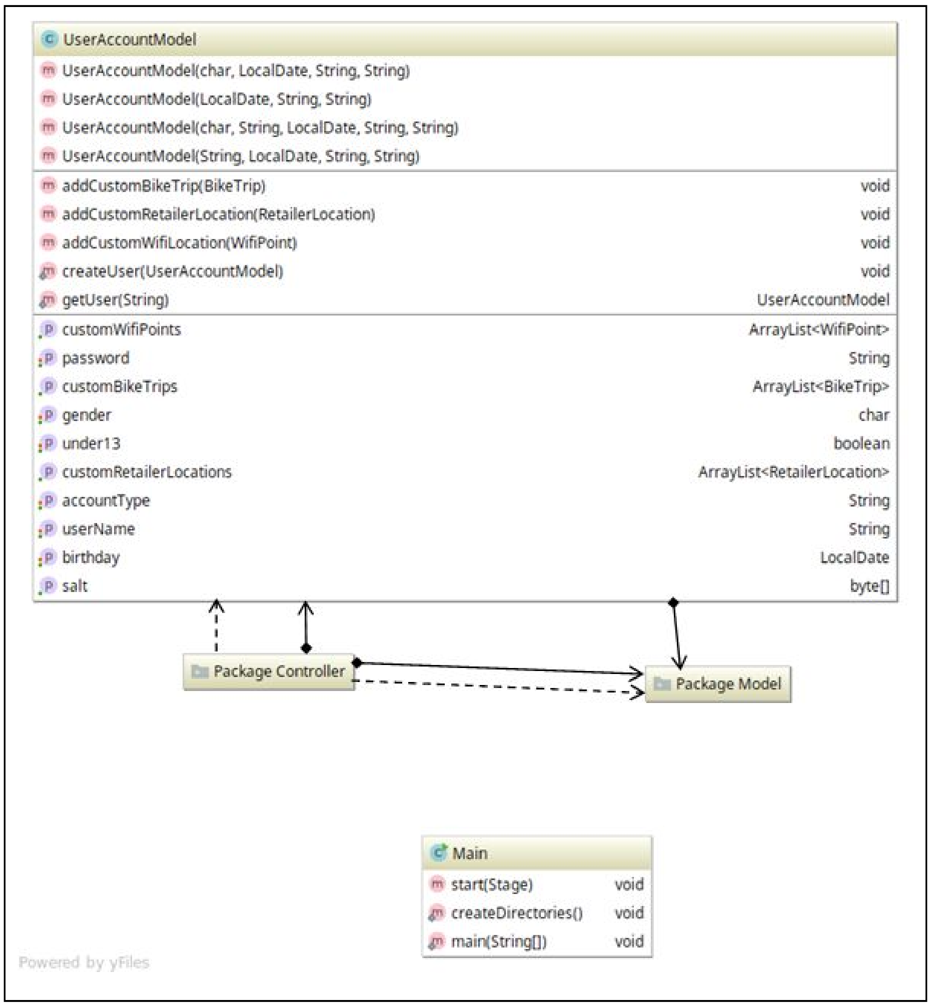
The folder hierarchy of the CYC system is provided below in figure 15, to give a clearer understanding of the package structure shown above.

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 17 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.



```
src/main/
├── java
│   ├── META-INF
│   └── seng202
│       └── team1
│           ├── Controller
│           │   ├── AddBikeDialogController.java
│           │   ├── AddRetailerDialogController.java
│           │   ├── AddWifiDialogController.java
│           │   ├── AlertGenerator.java
│           │   ├── BikeTableController.java
│           │   ├── LandingController.java
│           │   ├── LoginController.java
│           │   ├── MapController.java
│           │   ├── RetailerTableController.java
│           │   ├── TableController.java
│           │   └── WifiTableController.java
│           ├── Model
│           │   ├── CsvHandling
│           │   │   ├── CSVLoader.java
│           │   │   └── CsvParserException.java
│           │   ├── Google
│           │   │   ├── BikeDirections.java
│           │   │   ├── GoogleAPIClient.java
│           │   │   └── Routing.java
│           │   ├── BikeTrip.java
│           │   ├── DataAnalyser.java
│           │   ├── DatabaseManager.java
│           │   ├── DataPoint.java
│           │   ├── Filename.java
│           │   ├── GenerateFields.java
│           │   ├── InputValidator.java
│           │   ├── PasswordManager.java
│           │   ├── RetailerLocation.java
│           │   ├── SerializerImplementation.java
│           │   ├── UserAccountModel.java
│           │   └── WifiPoint.java
│           ├── Main.java
│           └── UserAccountModel.java
└── resources
    ├── css
    ├── csv
    ├── fxml
    ├── html
    │   ├── retailerCluster
    │   └── wifiCluster
    ├── images
    └── users
```

Figure 14: Folder Hierarchy of CYC



UserAccountModel

UserAccountModel(char, LocalDate, String, String)
UserAccountModel(LocalDate, String, String)
UserAccountModel(char, String, LocalDate, String, String)
UserAccountModel(String, LocalDate, String, String)

addCustomBikeTrip(BikeTrip)	void
addCustomRetailerLocation(RetailerLocation)	void
addCustomWifiLocation(WifiPoint)	void
createUser(UserAccountModel)	void
getUser(String)	UserAccountModel
customWifiPoints	ArrayList<WifiPoint>
password	String
customBikeTrips	ArrayList<BikeTrip>
gender	char
under13	boolean
customRetailerLocations	ArrayList<RetailerLocation>
accountType	String
userName	String
birthday	LocalDate
salt	byte[]

Main

start(Stage)	void
createDirectories()	void
main(String[])	void

Figure 15: Main Package

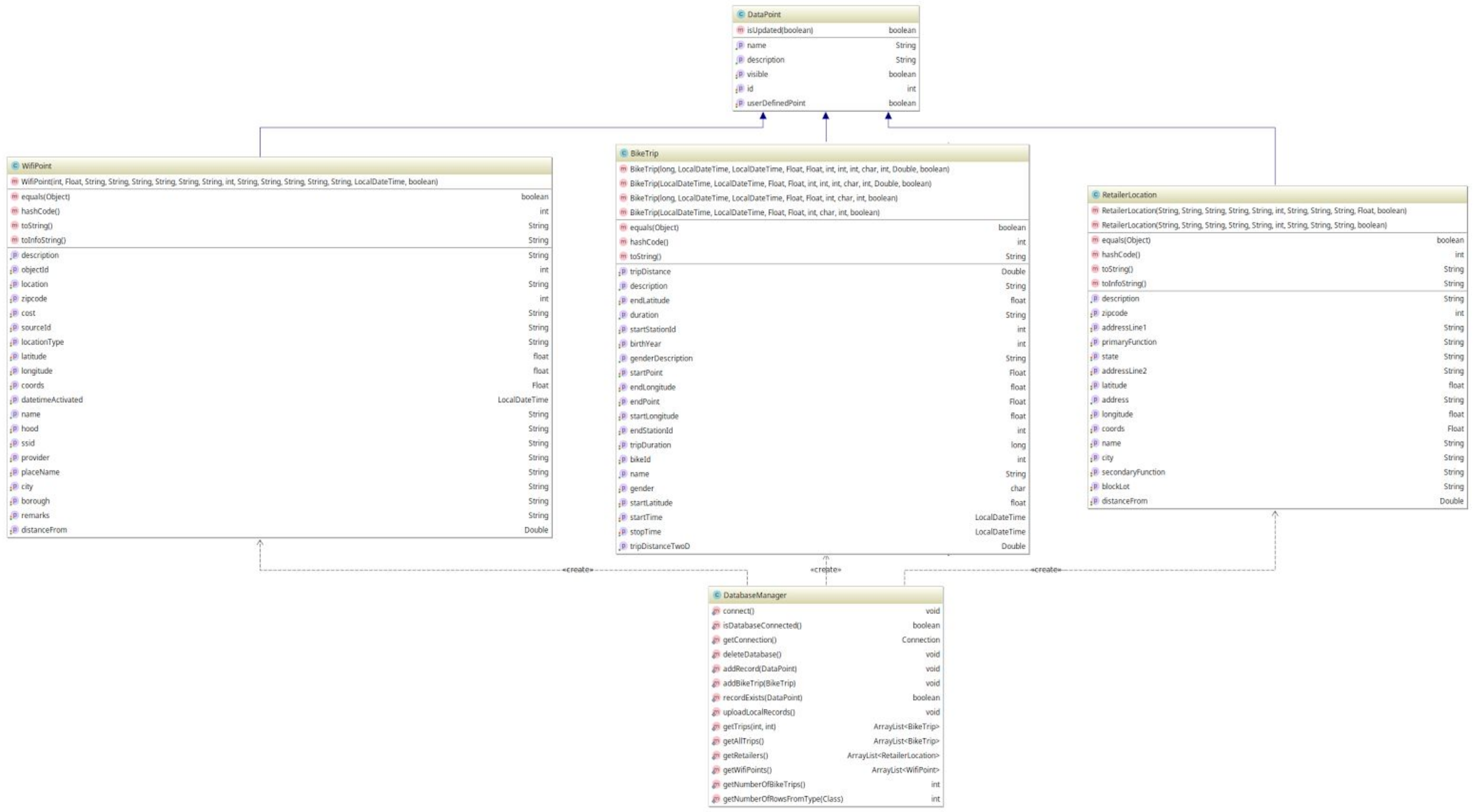
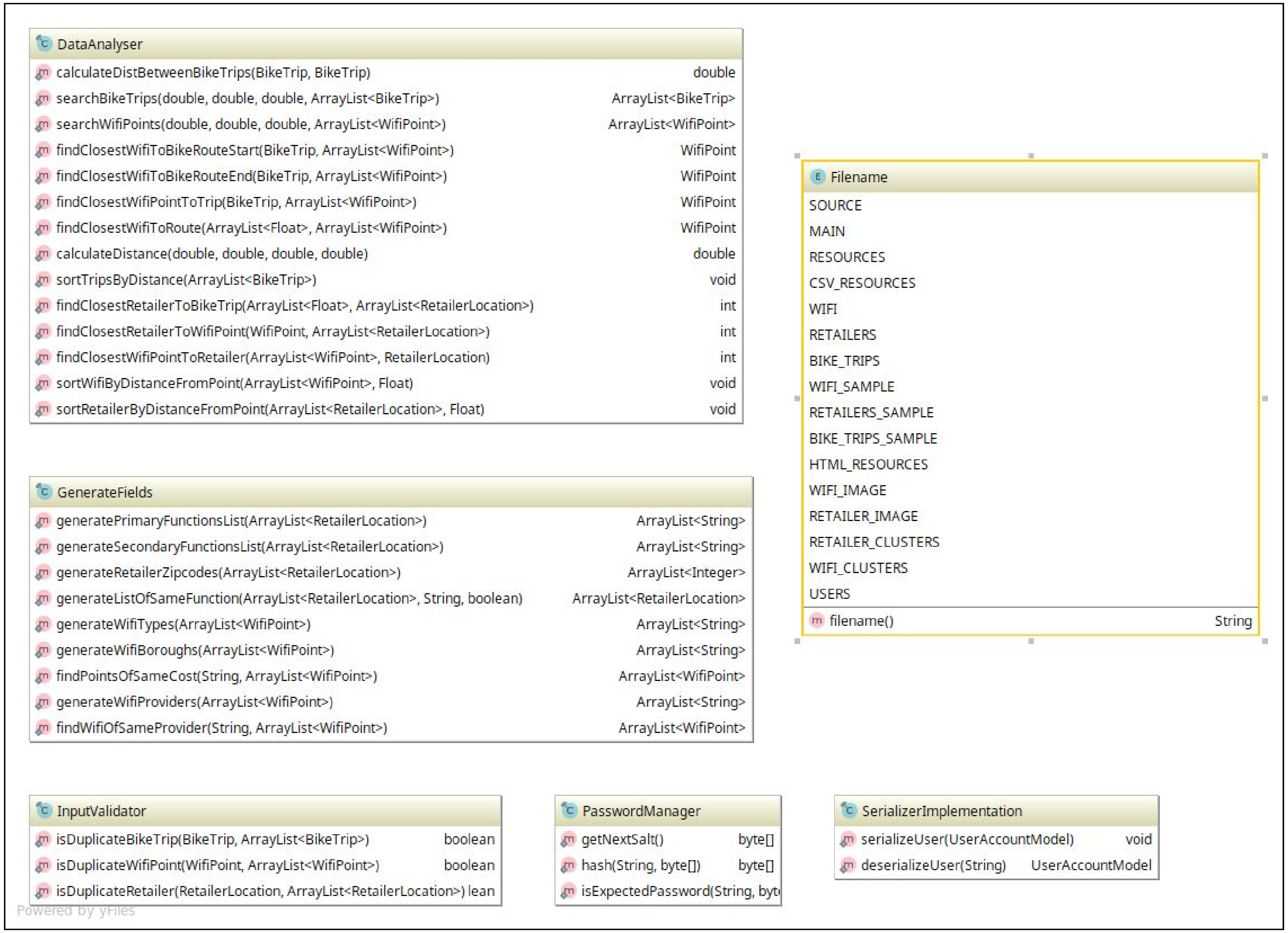


Figure 16: Model Package Part 1



```
DataAnalyser
calculateDistBetweenBikeTrips(BikeTrip, BikeTrip)                                double
searchBikeTrips(double, double, double, ArrayList<BikeTrip>)         ArrayList<BikeTrip>
searchWifiPoints(double, double, double, ArrayList<WifiPoint>)      ArrayList<WifiPoint>
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
sortWifiByDistanceFromPoint(ArrayList<WifiPoint>, Float)                            void
sortRetailerByDistanceFromPoint(ArrayList<RetailerLocation>, Float)                 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>
```

```
Filename
SOURCE
MAIN
RESOURCES
CSV_RESOURCES
WIFI
RETAILERS
BIKE_TRIPS
WIFI_SAMPLE
RETAILERS_SAMPLE
BIKE_TRIPS_SAMPLE
HTML_RESOURCES
WIFI_IMAGE
RETAILER_IMAGE
RETAILER_CLUSTERS
WIFI_CLUSTERS
USERS
filename()                                                 String
```

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

```
PasswordManager
getNextSalt()                    byte[]
hash(String, byte[])             byte[]
isExpectedPassword(String, byt
```

```
SerializerImplementation
serializeUser(UserAccountModel)             void
deserializeUser(String)         UserAccountModel
```

Figure 17: Model Package Part 2

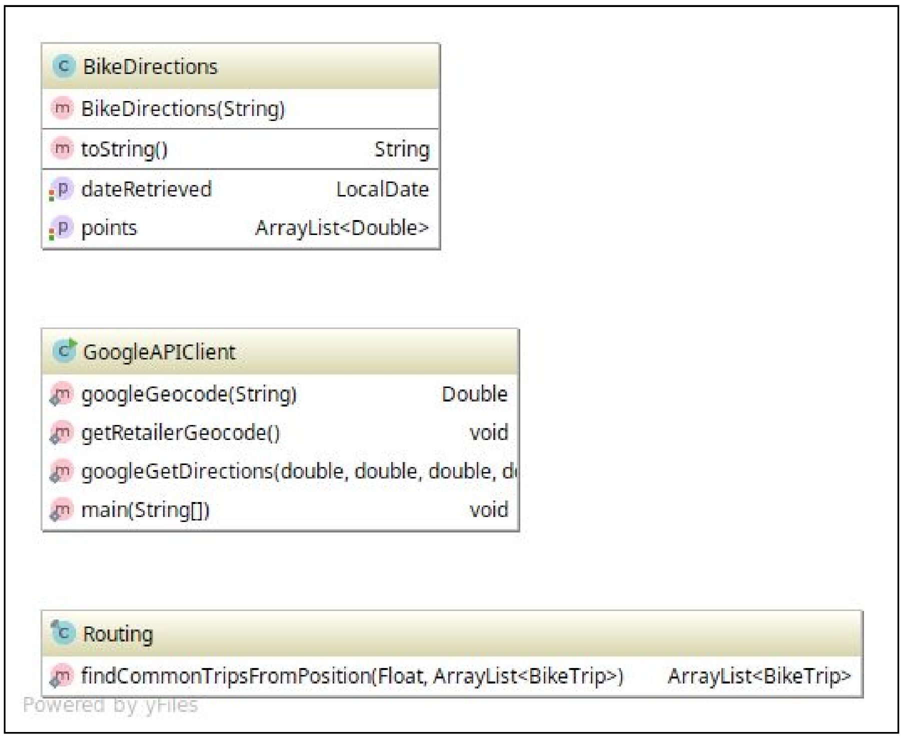

BikeDirections
BikeDirections(String)
toString() String
dateRetrieved LocalDate
points ArrayList<Double>
GoogleAPIClient
googleGeocode(String) Double
getRetailerGeocode() void
googleGetDirections(double, double, double, d
main(String[]) void
Routing
findCommonTripsFromPosition(Float, ArrayList<BikeTrip>) ArrayList<BikeTrip>
Powered by yFiles

Figure 18: Google Package

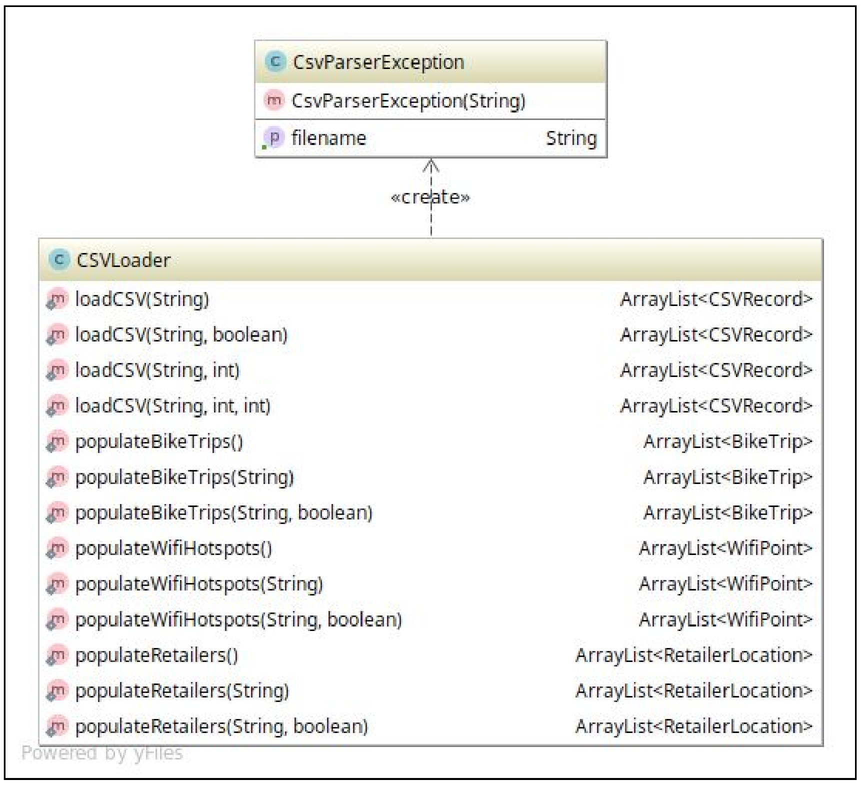

CsvParserException
CsvParserException(String)
filename String
«create»
CSVLoader
loadCSV(String) ArrayList<CSVRecord>
loadCSV(String, boolean) ArrayList<CSVRecord>
loadCSV(String, int) ArrayList<CSVRecord>
loadCSV(String, int, int) ArrayList<CSVRecord>
populateBikeTrips() ArrayList<BikeTrip>
populateBikeTrips(String) ArrayList<BikeTrip>
populateBikeTrips(String, boolean) ArrayList<BikeTrip>
populateWifiHotspots() ArrayList<WifiPoint>
populateWifiHotspots(String) ArrayList<WifiPoint>
populateWifiHotspots(String, boolean) ArrayList<WifiPoint>
populateRetailers() ArrayList<RetailerLocation>
populateRetailers(String) ArrayList<RetailerLocation>
populateRetailers(String, boolean) ArrayList<RetailerLocation>
Powered by yFiles

Figure 19: CsvHandling Package

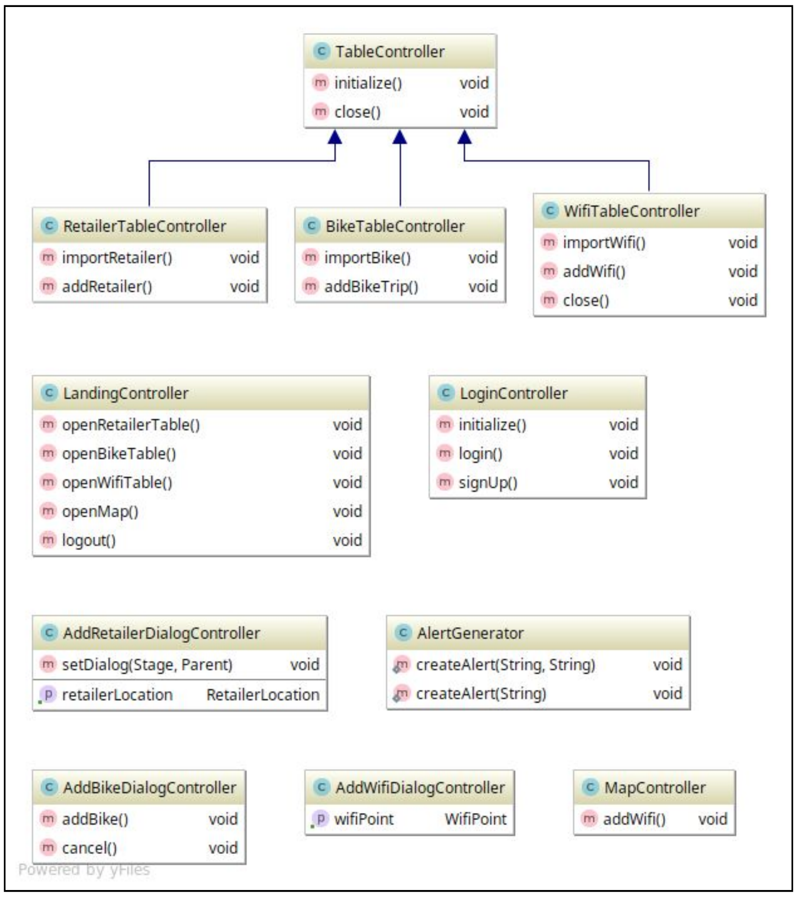


Figure 20: Controller Package

7. Risk Assessment

Table 7: Risk assessment for CYC.

ID	Description	Impact	Likelihood	Consequences	Prevention
R2	Broken code is committed to the main branch	Low	High	Application may become uncompilable 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.

8. Project Plan

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 tak 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	W2 7	Th 28	Fr 29	Sa 30	Su 1	Mo 2	Tu 3	We 4	T h5	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 11 the classes tested are shown with their coverage, this was measured with the built in intelliJ coverage tool using tracing. Total coverage is 26% for methods and 30% 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	30% (16/53)	26%(128/480)	29% (818/2737)	
BikeTrip	100% (1/1)	63% (26/41)	84% (108/128)	Is an instance class containing many getters and setters.
CSVLoader	100%(1/1)	50%(5/10)	82% (106/129)	
DataAnalysier	100% (4/4)	100% (18/18)	95%(146/152)	
DatabaseManager	100%(1/1)	92%(13/14)	85%(189/220)	
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)	58%(20/34)	77%(61/79)	Is an instance class containing many getters and setters.
Routing	100%(1/1)	100%(1/1)	91%(11/12)	
SerializerImplementation	100%(1/1)	100%(2/2)	70%(14/20)	
WifiPoint	100%(1/1)	51%(22/43)	63%(50/79)	Is an instance class containing many getters and setters.

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	Fail
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	Functionality was not implemented, no route was found
Pass/Fail	Fail
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	Fail
Tester	Josh Burt
UC / Requirement	UC5

Item	Description
SUT	System
Test	AT7
Description	User can view personal route history.
Result	Functionality was not implemented, no route history was found
Pass/Fail	Fail
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	Functionality was not implemented, no route history was found
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

9.3 Discussion

All acceptance tests have been performed. Four passed and six failed. Of the six that failed, one was a high priority. However this test only just failed so is considered non critical. The remaining five are all medium priority. All of these have the underlying functionality available in the model classes, this makes these failures non critical. Three of the passing acceptance tests were rated high priority, one was rated low. Failing acceptance tests have been reviewed and the updated project plan has plans in place to ensure all acceptance tests are 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 test 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

From the project description (Learn, 2017), Feature packages 1 and 3 have been fully implemented on the back end of the system. Feature packages 2, 5 and 6 have been partially implemented.

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.

Another feature we are proud of is the filtering in the table view. It is accurate, responsive and works well with large data sets, a large data set is defined as over 1000 entries. Data also loads for much larger files, albeit slowly, and the threading design of this allows the GUI not hang so other operations can be performed simultaneously.

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 Not Implemented

The high priority requirements shown in table 16 are still to be implemented.

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

This leaves the medium and low priority use cases, shown in table 16, as still to be implemented.

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
FR27	User can load CSV data with an	S1	Medium	UC8

	alternate number of attributes per entity/line as defined by the user at runtime.			
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 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
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
FR41	Bike trip, WiFi, and retailer data can be exported over a network to a central database.	S1, S2	Medium	UC5, UC7, UC11
FR45	User can export their lists as CSV files.	S1, S2	Low	UC10

From the project description (Learn, 2017), Feature packages 4 and 7 are still to be implemented. Feature package 2 requires filtering by latitude and longitude on the table to be implemented. Feature package 5 requires three of five tasks to be implemented. Feature package 6 requires six out of twelve tasks to be implemented.

Change Log

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

Additional Comments

We set the finish time for the timelogs as 2pm on 26 September so that we had time to export them and compile the deliverable.

Josh Burt

My work flow for implementation involved writing tests, implementing, research the functionality on difficulties and then Javadocing the method. These are not entered as separately tagged items. All work on the design document was on all sections rather than focussing on one at a time.