Adaptive Cycling Route Planner for Dublin City

Maurice J. Gavin
B.A.I. Engineering
Final Year Project April 2012
Supervisor: Dr. Mélanie Bouroche

School of Computer Science and Statistics

O’Reilly Institute, Trinity College, Dublin 2, Ireland
I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

------------------------------------------  ------------------------------------------
Name                                      Date
I would like to thank my supervisors Dr. Mélanie Bouroche and Dr. Bidisha Ghosh for their guidance, encouragement, patience and time as I undertook this final year project with them.

I want to thank the cyclists who navigated the treacherous roads of Dublin to test Rothaim. Without your valiant input and unquenchable enthusiasm the project could not have become what it is today.

To Neil, Paddy and Jimmy from the Trinity Centre for High Performance Computing, thank you for the multiple impromptu debugging sessions and for always being obliging to my barrage of emails.

I would like to thank Ms. Anneka Lawson for the hours of her time that I stole seeking feedback, papers and advice.

Thank you to Michael Burke, Andy Donnelly, Nick O’Brien and Michael Shine for engaging so completely with the project.

Thank you to my parents for their unwavering support throughout the project and apologies for the constant chatter about the cycling application.

Finally, thank you to the multitude of people who showered me with feedback, encouragement and enthusiasm.
Abstract

The purpose of this thesis was to build an adaptive cycling route planner. An Android application was developed to collect information about cyclist behaviour in Dublin City. The result was the creation of a central repository of cyclist data and an application ready for public release. The goal of creating the adaptive route planner was not fully realised but the net contribution of the thesis is far more valuable to the transport research community than an individual route planner.
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>System Design</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Application Design</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Recording Process UML</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>User Interface Design</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>History, Trip Summary</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>Network UML</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>Internal SQLite Database Design</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>Server Request General Form</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>Server Design</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Proxy UML</td>
<td>16</td>
</tr>
<tr>
<td>2.11</td>
<td>Server Request Handling UML</td>
<td>18</td>
</tr>
<tr>
<td>2.12</td>
<td>Server-Side Database Design</td>
<td>19</td>
</tr>
<tr>
<td>2.13</td>
<td>Control Panel UI</td>
<td>20</td>
</tr>
<tr>
<td>2.14</td>
<td>Database Reader Tool</td>
<td>20</td>
</tr>
<tr>
<td>2.15</td>
<td>Downloaded Cyclist Data</td>
<td>21</td>
</tr>
<tr>
<td>2.16</td>
<td>Parameterized SQL Query</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Progress Bar</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>GPS Triangulation [18]</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>General Form of a Networking AsyncTask</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>Network Alert Dialogue</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>Reason Options</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>Type Options</td>
<td>46</td>
</tr>
<tr>
<td>3.7</td>
<td>Custom Graphics</td>
<td>46</td>
</tr>
<tr>
<td>3.8</td>
<td>Recording</td>
<td>47</td>
</tr>
<tr>
<td>3.9</td>
<td>History Feature</td>
<td>48</td>
</tr>
<tr>
<td>4.1</td>
<td>Data Visualisation 1</td>
<td>53</td>
</tr>
<tr>
<td>4.2</td>
<td>Data Visualisation 2</td>
<td>54</td>
</tr>
<tr>
<td>4.3</td>
<td>Data Visualisation 3</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Graph of Journey Reason Breakdown</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Graph of Route Perception</td>
<td>58</td>
</tr>
<tr>
<td>4.6</td>
<td>Discarding High Inaccuracy Observations and Linear Interpolation</td>
<td>61</td>
</tr>
<tr>
<td>4.7</td>
<td>Main Screen Context Menu</td>
<td>64</td>
</tr>
<tr>
<td>4.8</td>
<td>History Context Menu</td>
<td>64</td>
</tr>
</tbody>
</table>
# Table of Contents

Declaration .............................................................................................................. I

Acknowledgements ............................................................................................... II

Abstract .................................................................................................................. III

Table of Figures ..................................................................................................... IV

Chapter 1: Introduction ......................................................................................... 1

  1.1 Motivation ...................................................................................................... 1

  1.2 State of the Art ............................................................................................ 2

  1.3 Objectives ..................................................................................................... 2

  1.4 Organisation of Thesis ................................................................................ 3

Chapter 2: Design .................................................................................................... 4

  2.1 System Design .............................................................................................. 4

     2.1.1 Client Android Application (Rothaím) .................................................. 7

     2.1.2 Server .................................................................................................. 13

     2.1.3 Auxiliary Tools ................................................................................ 19

  2.2 Target Platform ............................................................................................ 22

     2.2.1 Why choose Android? ...................................................................... 22

     2.2.2 Multiplatform support ...................................................................... 22

  2.3 Software Requirements ............................................................................... 23

     2.3.1 Robustness ...................................................................................... 23

     2.3.2 Extensible ......................................................................................... 23

     2.3.3 Scalability ......................................................................................... 23

     2.3.4 Usability ......................................................................................... 24

  2.4 Security ......................................................................................................... 24

     2.4.1 Security Measures ........................................................................... 24

Chapter 3: Development & Implementation ........................................................... 27

  3.1 Application .................................................................................................. 27

     3.1.1 User Interface ................................................................................... 27

     3.1.2 Services & APIs ............................................................................. 30

     3.1.3 Networking .................................................................................... 35
Chapter 1: Introduction

1.1 Motivation

In recent times, cycling as a healthy and sustainable mode of transport is continuously being encouraged. It is being endorsed by businesses, policy-makers, doctors, transport planners, insurers, the media and the cycling community itself. In 2008, it was estimated that traffic congestion costs Dublin businesses in excess of €2.5 billion each year in loss of productivity [1]. The environmental footprint of a cyclist is much lower than that of a motorist. If more trips were to be taken by bike the impact on nature would be minimised. Occurrences of weight related health problems are on the rise in modern Ireland. The health and exercise benefits of cycling have the potential to reverse this trend. Legislation makers see an uptake in sustainable modes of transport, such as cycling, as a ready solution to these problems. The Smarter Travel Plan 2009-2020 - the Framework Policy for Sustainable Transport sets out the objective to have 10% of all journeys to work taken on bicycles by 2020 [2]. To reach this goal a culture that is supportive of cycling needs to be fostered in Ireland.

Initiatives like the Cycle to Work Scheme launched in 2009 provide financial incentives to people to commute on a bike. These financial perks coupled with national public awareness campaigns showcase cycling as a viable transport alternative to cars, particularly to those living in urban areas. In the five years leading up to 2009 the number of cyclists on Irish roads grew by 30% [3]. These cyclists are becoming increasingly vocal about the dangers of cycling in Dublin and the poor condition of some of the city’s cycling lanes. There is a perception among potential and novice cyclists that the road network is a hostile and dangerous environment for someone on a bicycle. This perception is supported by road death statistics released by An Garda Síochána. Already this year a person has been killed off a bicycle [4].

Campaigns are clearly having a very positive effect at changing the public perception of cycling; this is reflected in its growing popularity. However, despite lavish investment [5] spent on infrastructural upgrades Ireland’s road network remains of great danger to cyclist safety. At present urban planning is steered by information derived from surveys and common sense. This can result in cycling lanes being built along lesser used routes when in fact it could have made more of an impact were it located on a busier or less safe route. Unfortunately it is difficult for the decision makers to implement the optimal infrastructure when they lack the information to identify it. Traditionally, formal network based transport planning has only been applied to vehicles and pedestrians with cycling being overlooked.

The obvious solution to this problem is to create a network of this sort for cycling. To achieve this goal a large volume of high quality cycling data needs to be amassed. The method by which this data can be collected is the subject of this project investigation. Once
a network model exists cycling planning decisions will no longer rely on presumptions and suppositions but rather on verifiable scientific data.

1.2 State of the Art
Most route planners, web and mobile alike, try to integrate all transport modes in their suggestions but cycling is currently rarely considered. The few existing cycling-specific route planners only return routes where the path is optimised for least distance or least time. In a road network, cyclists are the group exposed to the maximum amount of risk. As a result their optimal route is often influenced by other factors like level of expertise, perceived or actual road risks, personal decisions, weather conditions and so on as opposed to least path or least time.

The San Francisco County Transportation Authority recently funded the development of an application for Android and iPhone. The goal was to identify the routes most frequently travelled by cyclists in San Francisco. The city authorities have set a target, similar to the one outlined in the Irish Smarter Travel Plan, that by 2020 cycle trips would account for 20% of all daily journeys [6]. In addition to routes, the application also captures route information and the experience level of the cyclist.

TrailBlaze is another recently released Android application which tracks the route taken by a cyclist. It was developed by Aqualab. The website blurb reveals that they are conducting their research with the intention of making the city of Chicago more cyclist friendly. To this end they want users to record their favourite routes so the city planners are aware of these popular routes [7].

IBM are running a project out of their new Smarter Cities Technology Centre to investigate the impedances to people moving about the city. Factors from road width to the availability of public transport are being considered. They are using mounted sensors around the city to feed data back to a model which they hope will yield conclusions that can reduce congestion and pollution in the city [8].

1.3 Objectives
This final year project is the enabler for a number of transport related research projects. The principal goal is to develop a system with the ability to collect a large volume of cycling route data and journey meta-data. By collecting this data with software this project signals a departure point from the traditional time intensive methods of gathering the same information, such as meeting with users to discuss their trips. This autonomous system, where users do all of the data collection, will be achieved by creating a mobile application to facilitate the recording of data and a server to store this data securely.

Another goal is for users to embrace the mobile application to the extent that they would choose to use it outside of the study. Successful completion of this goal will complement the first as users will be gathering more data due to their continued use of the software.
A final wish for this project would be that, at some point in the future, the data collected would guide targeted development of infrastructure with the intention of minimising the risks to and fatalities of cyclists on Irish roads.

1.4 Organisation of Thesis
Chapter 1 introduces the project. It discusses at length the backdrop of the state of cycling in Ireland today and some of the challenges that are faced. It goes on to describe similar studies and mobile applications and what they aimed to achieve. Finally the project objectives are clearly stated.

In Chapter 2 the technical design of the software is presented. The system architecture is described beginning at the highest level. Key components are discussed in further detail. A list of general software requirements which are desirable for this project finish the chapter.

Chapter 3 looks at the actual development and implementation of the project. The components highlighted in the design section are discussed with respect to the technologies used, the challenges encountered and the solutions applied. This chapter ends with a roundup of the features which affect the usability of the product for both users and researchers.

Chapter 4 is where the results of the project are presented. The user testing period section includes a discussion on ethics in software produced for public consumption as well as the processes of distribution and data collection. Later in the chapter the data that was collected during this period is analysed, with attention given to how this information was retrieved from the central server, the information gleaned from the data, actions arising from user feedback and a short discussion the quality of the data collected. The chapter closes by drawing a comparison between different types of testing and an evaluation of how well the application met the software requirements remit.

In Chapter 5, the final chapter, the thesis is concluded. A critical assessment of the work done is delivered. Some thoughts on how a personalised route planner would be implemented are proffered for consideration. Potential areas of future work are outlined for posterity. The thesis ends with a summation of the investigators personal experience of the completed project.
In this chapter, the system architecture will be presented. The system will be segmented into its component parts and the most important of these will be described in detail. After a thorough analysis of the system design we will look at the process that led to choosing Android as the target platform for the proposed mobile application. The requirements which affect the performance and operation of the software will then be outlined. Finally, the chapter will close by presenting the security considerations take into account when designing the product.

2.1 System Design
The software used to create this project can be decomposed into three distinct packages. The system design can be seen in Figure 2.1 below. The Android application is the application run by the cyclists. It collects data which is stored in the server. The data on the server is accessible from both the application and a variety of auxiliary tools. Each component has its own specific responsibilities and when made to interact with each other together they form a system which fulfils the project objectives.
Figure 2.1: System Design

Server - running Red Hat Enterprise Linux 5.6 - provided by Trinity Centre for High Performance Computing

Client Android Application

User Interface

Network Controller

Main Servlet - responds to doGet and doPost requests

Proxy

Request Specific Classes

MySQL Database

External Services and Data

GPS Satellite Readings

Google Maps Service

Local Database

Database Reader Tool

Server Admin Control Panel

Directory of Auxiliary Tools

Internet communications to be secured using HTTPS

Main Servlet - responds to doGet and doPost requests

Proxy

Request Specific Classes

MySQL Database

External Services and Data

GPS Satellite Readings

Google Maps Service

Local Database

Database Reader Tool

Server Admin Control Panel

Software environment diagram
2.1.1 Client Android Application (Rothaím)

This is the application that cyclists can install and run on the Android devices. Its primary responsibilities are to facilitate the collection of trip data and to present an attractive front end to the user. In order to function completely it depends on the availability of two externally run services; namely GPS satellite readings and a connection to the Google Maps Service. In the wider context of the system each cyclist’s application is just one of many clients which commit data to a central database. Below is a more detailed look at the key components of the Android application itself.

2.1.1.1 Recording Design

It has been mentioned that the core purpose of the application is to collect location data. To achieve this it needs a service to first read this data from its source. In the application the user will indicate that they want to begin recording a trip by launching the MainActivity screen (in Android most UI classes have the word Activity appended to them). The class constructor will read from the device’s SharedPreferences to determine whether or not the application is already recording. This can be the case if a user started recording and then navigated away to other applications. If no recording is underway then a new RecordingController is created to manage the recording process. It then begins a new RecordingService and binds it to the current application context.

The RecordingService registers with the operating system that it would like to receive location updates. When updates are received it notifies its observers, namely the RecordingController. When the RecordingController is notified it updates its Trip object. The trip object adds the location to a buffer which flushes these location observations to disk when the internal SQLite database is available. Finally, the RecordingController does some UI updates such as centring the current map view on the latest location and updating the distance travelled.

Above is just one scenario. Since users tend to use many applications at once on their devices it is necessary that the RecordingService continues to work in the background. If a user launches Rothaím while recording is already in progress the application must behave the same as if it had never been closed. Upon resume a new RecordingController is immediately created and attempts to bind the already running RecordingService to this instance of the application. Knowing whether or not a trip is being recorded is determined by reading simple name value pairs from Android useful SharedPreferences.

In addition to the core functionality described above the user can change the state of the RecordingService by using the controls provided in the MainActivity to pause, resume and stop recording. The difficulties surrounding the recording process will be discussed in detail in the next chapter.
2.1.1.2 User Interface Design

Figure 2.4: User Interface Design

Figure 2.3: Recording Process UML
When a user launches the application the first screen they visit is the splash screen. This displays the application name, logo, version and author. After a short animation the application automatically transitions to the main screen. On this screen the user can configure the recording of a journey by inputting the reason for their trip, their perception of their route, then pressing record. This brings them to the recording screen which shows a map (centred on the user’s current location), dynamic statistics about the trip and a panel of buttons to control the state of the recording service.

When a user is finished recording their trip they are shown the final statistics for their trip, including time taken, distance travelled, average speed, maximum speed, average pace, maximum pace and an estimation of the calories burned.

Alternatively, if the user does not want to record a trip they have some other options accessible from the main screen. By pressing Android’s menu button a user can open a context menu. This menu offers a variety of options to the user. They can read the application documentation in ‘Help’, they can view information about the application in ‘About’ or they can change their user preferences in ‘Settings’. Another option available to them is ‘Login’. This brings them to a screen where they can sign into their account by supplying their username and password credentials. New users have the option to register a new account by pressing the signup button instead. This will uncover additional fields necessary for registration. Once a user has an account and is logged in they can now access other features which require authentication like uploading trips to the server.

Whether or not a user is logged in they can still view the journeys stored locally on their phone. To do this they choose the ‘History’ option from the main screen’s context menu. This brings up a list scrollable list where each entry has a summary of basic trip information and a clickable icon indicating whether or not the trip has been uploaded.

![Figure 2.5: History, Trip Summary](image)
2.1.1.3 Networking Design

The backbone of the network design consists of a NetworkController and three task specific classes. The NetworkController is responsible for sending requests to the server, catching responses from the server and publishing notifications to the UI. To achieve all of these tasks it delegates many of these tasks to threaded objects and simply provides static helper functions.

The AccountNetworkHandler, DownloadData, UploadData classes all extend Android’s extraordinarily useful AsyncTask class. It greatly simplifies the process of creating, monitoring and releasing threads. In addition to that it also provides overridable callback functions for publishing a task’s progress to the screen and a post execute function also.

The only connections that the application makes to the network are to perform interactions with the server. By default all outgoing connections are HTTPS, meaning that data sent across the network is encrypted. Furthermore, when a connection is made the application verifies the server certificate against a locally held copy. HTTPS can be disabled by users if they wish through the Settings screen. When a user tries to upload a trip the active NetworkController will create a new instance of an UploadData thread to perform the request. The same happens if the request is to download or to authenticate, the only difference being that the controller creates a DownloadData or AccountNetworkHandler thread respectively.

Upon creation the thread statically requests a client for the request from the NetworkController class. This client object is then written and read to in turn to complete the desired operation. The NetworkController uses the current application context to create a NetworkView. As it receives progress updates and messages from the thread it can choose to publish this information to the user interface using its NetworkView. An example of this is the progress bar seen when a trip is uploaded in Rothaím. Depending on the result of the network request, a message indicating success or the type of error will be returned.

In the next chapter we will examine the technologies used to implement this design.
2.1.4 Database design

This application uses a SQLite Database to store trips and location observations in a structured format. Android's implementation of SQLite restricts other applications from accessing the databases of each other making this an accepting secure location to store trip data. The variables of interest were identified for storage and tables were created accordingly. They are as follows:

2.1.4.1 Trips Table

- **ROWID**: a unique index to identify the trip
- **REASON**: the reason that the user gave for this trip
- **TYPE**: the perception that the user has of their intended route
- **STARTTIME**: the time at which the journey was begun
- **ENDTIME**: the time that the journey ended
- **PAUSETIME**: the total time that recording was paused
- **DISTANCE**: the total length of the route
- **LATHI/LATLO**: the geographically highest/lowest longitudes
- **LGTHI/LGTLO**: the geographically highest/lowest longitudes
- **NODES**: the number of observations associated with the trip

The total trip time can be calculated using these three values.

Used to zoom the map to an appropriate scale.
• UPLOADED: a Boolean indicated whether or not the trip has been uploaded
• STATUS: the current status of the trip, i.e. INCOMPLETE, ABORTED, FINISHED, SAVED

2.1.1.4.2 Coords Table
• ROWID: a unique index to identify the location observation
• PARENT_TRIP: a foreign key to reference the associated parent trip in the ‘Trip’ table
• TIME: a timestamp of when the location was observed
• LAT: the latitude of the location observation
• LGT: the longitude of the location observation
• ACC: the estimated accuracy of the location observation
• ALT: the altitude of the location observation
• SPEED: the perceived cyclists speed at the time the observation was made

![Internal SQLite Database Design](image-url)
2.1.2 Server

The server architecture can be explained as three components. There is a proxy for authentication purposes. The ‘Main Servlet’ is the core of the server. It executes all valid requests to the server and manages the responses. Behind the server there exists a MySQL database with three tables. This database is used for the safe storage and retrieval of user and trip data. An Apache Tomcat 6 container enables the proxy and main servlets, written in Java, to run on the server.

When a request reaches the server the proxy servlet attempts to verify its validity. If the request mode is to ping, login or register then the ‘AuthenticationServlet’ handles the request itself. Alternatively if the request is valid but the mode is not recognised the ‘AuthenticationServlet’ will forward the request to the ‘MainServlet’.

The MainServlet deals with all non-authentication related requests, such as uploading, downloading, providing the admin control panel, etc. Both of the aforementioned servlets have access to the backend database. The proxy uses it to verify user credentials and to create new users and the ‘MainServlet’ uses it for uploading and downloading trips.

The general form of any server request is given in Figure 2.8.

<table>
<thead>
<tr>
<th>Request</th>
<th>Mode</th>
<th>Session Key</th>
<th>Additional Params</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>ping</td>
<td>xxxxxxxxxxx</td>
<td>username</td>
<td>Observation_0</td>
</tr>
<tr>
<td>data</td>
<td>login</td>
<td>dddddd</td>
<td>password</td>
<td></td>
</tr>
<tr>
<td></td>
<td>register</td>
<td>dddddd</td>
<td>tripid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>download</td>
<td>ttttttttttttttttttttttttttttt</td>
<td>trip_reason</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>trip_type</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.8: Server Request General Form
Apache Tomcat Container

- running Red Hat Enterprise Linux 5.6
- provided by Trinity Centre for High Performance Computing

Apache Tomcat Container

Main Servlet
- responds to doGet and doPost requests

Authentication Servlet

Session Key Manager

Upload Handler

Download Handler

Request Forwarding

Database Adapter

Using JDBC driver

MySQL Database

Database Adapter

Server Logs

Logging Utility

Debug

Server Response

Server Request

Server Logs

Server Request

Server Response

Server Response

Server Response

Figure 2.9: Server Design
2.1.2.1 Servlet Design

To be able to execute code on a server it is necessary to have some container in which to run it. As stated previously, this project implements Apache Tomcat 6 to provide this service. This container is responsible for managing the lifecycle of each servlet that it creates. Connections intended for the servlets are must connect to a port on which Tomcat is listening. Since the Rothaím application can use either HTTP or HTTPS connections Tomcat has two default ports, Port 8080 for HTTP and Port 8443 for HTTPS.

The servlets themselves are simply Java classes which can respond to requests. The servlets used for this project are HTTP Servlets; that is, servlets which respond to HTTP requests. When a request is caught by Tomcat it checks its thread pool and if a thread is available it assigns a servlet to deal with the request.

This server implements two servlets, both with very different purposes. One is a simple proxy servlet while the other handles all other requests.

2.1.2.1.1 AuthenticationServlet (Proxy)

The ‘AuthenticationServlet’ is the application’s only way of connecting to the server. It is responsible for servicing all authentication requests. The ‘AuthenticationServlet’ maintains a static list of valid session keys. Each session key has an ID, an expiration time and the name of the user to which it was assigned. All keys are valid for a 15 minute period. When the servlet is created it also creates a ‘SessionKey Manager’. Calls are made to this object when the servlet needs to do something with session keys. Expired session keys are routinely cleaned by a regularly scheduled thread. When a user wishes to perform any operation anywhere on the server they must first provide a valid session key otherwise their request will be immediately dropped and an error message returned in the response. There are two exceptions to this rule, signup and login requests.

If a user attempts to login it is expected that they do not have a valid session key. The servlet parses the request and extracts the supplied credentials. The username and password are checked against the user table in the MySQL Database and if a match is found then it does one of two things. It checks its static list of session key’s to see if the user already has a valid session key. If they do then this is simply returned. Otherwise a new session key is created, inserted into the list of valid keys and finally returned.

For all other requests however, a valid session key is needed to proceed. Once the servlet validates a session key it checks the header for the request’s mode. This is a String which indicates the type of request. At the time of writing the proxy servlet’s supported modes are:

- Ping: this mode is simply accompanied by a session key. All it does is check to see whether or not the session key provided is valid. This might be done by the application if a supposed valid key (with an in-date expiry) has been refused by the
server. This problem can arise if the server is unexpectedly reset causing it to lose its list of valid keys.

- **Login**: as described above this feature allows a user to get a session key by providing a valid username and password pair.

- **Signup**: requests with this mode are accompanied by the details necessary to create a new user. The servlet creates a writable connection the backend database and inserts a new row in the user table. This type of request does not return a session key. The application needs to subsequently login using these new credentials to receive one. (In Rothaím when a user creates a new account the application automatically logs them in also, thereby fetching a session key).

2.1.2.1.2 **MainServlet**

The MainServlet deals with all requests forwarded by the proxy. This means that no further validation needs to be done with the session key. There are a number of requests that the MainServlet needs to be able to handle. Most of these are to provide extra functionality to the application for a logged in user. There are two types of request that are handled, doGet and doPost.

**HTTP GET:**

- **List**: this request returns a list of trips that the logged in user has permission to access. Most users only have permissions to read their own trips. Privileged users, for example those who are accessing the data for research purposes, are able to

![Proxy UML](image-url)

Figure 2.10: Proxy UML
access the trips of other users. This only returns the databaseID and brief summary of the available trips. To download a whole trip the user needs to send a download request containing the databaseID of the target trip.

- Download: the download request contains the databaseID of the trip to be downloaded. The MainServlet will verify that the user requesting the download has sufficient permission to read it. If it does then it retrieves the trip from the database and writes it to the response.

HTTP POST:

- Create: this request is sent by a client which wishes to upload a trip to the server's database. The request header contains the basic trip information like trip reason, perceived route type, the start time and so on. The servlet creates a new row in the trip table to store this data and returns the primary key of the row.

- Update: this request is sent when a client wishes to upload location observations. Each request must also contain the primary key of the parent trip so that rows in the observation table can be associated with the correct trip. These observations are usually sent in chunks so it is not uncommon for an entire upload transaction to have many update requests. This means that if a transmission does not succeed there is less data to retransmit on the network.

- Finish: the client sends a finish request when there are no more observations to be sent. The servlet can now close its connection to the database and sends an acknowledgement to the client to indicate a successful upload.
2.1.2.2 Database Design

At the backend of the server there is a MySQL database. The database has three tables. The user table stores the account information of those users who have registered to use Rothaim. The other two, the trip table and the observation table, store the trip and route data uploaded by cyclists. The user table is written to exclusively by the AuthenticationServlet when new users are created. Similarly the trip table and observation table are only ever written to by the MainServlet. The tables are as follows.

2.1.2.2.1 User Table
- username: the username of the user. This field is read when users try to authenticate themselves with the proxy. This is the tables primary key and therefore all usernames must be unique.
- password: the password associated with the username.
- email: the email address of the user. This can be used to send software updates to the user.
- date_of_birth: this value can be used to determine the age of a cyclist. This may be useful at some point for evaluating the cycling habits of different demographics.
- gender: the gender of the user.
- weight: the weight of the user.
- height: the height of the user.

Note: In this project DOB, gender, weight and height of the user are not collected. We wanted to ensure that the server was secure before taking sensitive user information.

2.1.2.2.2 Trip Table
- id: the unique primary key of the row in the trip table.
- username: the username of the trip owner, foreign key references a user in the user table.
- internalDbTripID: the ROWID of the trip on the device’s internal SQLite database.
- route_reason: the reason for the journey.
- route_type: the perceived route type.
- time_of_day: the time of day at which the trip was recorded (a number between 0 and 23).
- start_time: the time at which the trip began, stored in Unix time.
- end_time: the time at which the trip ended, stored in Unix time.
- upload_time: the time at which the trip was uploaded, stored in Unix time.
- distance: the distance travelled during the trip.

2.1.2.2.3 Observation Table
- id: the unique primary key of the observation in the observation table
• tripID: the row id of the parent trip to which this observation is a child. This foreign key references the parent trip in the trip table.
• latitude: the observed device latitude.
• longitude: the observed device longitude.
• obs_timestamp: the time at which the observation was made, in Unix time.
• speed: the perceived speed at the time of observation.
• altitude: the altitude at the time of observation.
• accuracy: the estimated accuracy of the longitude and latitude readings.

![Database Schema Diagram]

Figure 2.12: Server-Side Database Design

2.1.3 Auxiliary Tools
These additional tools support and expand upon the core services of the project. There are two particularly useful tools that are of interest.

2.1.3.1 Admin Control Panel
The Admin Control Panel is a tool which can be used to view and change the settings of the servlet. Using a privileged username and password one can log in to the control panel to view the server statistics and settings. Settings can be easily changed by passing parameters in the URL. This removes the need to build and export a new version of the server every time a setting, like the server logging level, is to be changed. This tool is not meant to replace the Tomcat Web Application Manager which is a Tomcat plug-in that provides a GUI for starting, stopping, restarting, etc. individual servlets.

```
 ADMIN CONTROL PANEL
 ----------------------------------------
 Server is running version: v1.8.46
 ----------------------------------------
 SETTINGS
 Debug Level: 0
 ----------------------------------------
```

![Admin Control Panel Screenshot]
2.1.3.2 Database Reader Tool

This tool can be used to view a list of accessible trips on the server. Furthermore it can be used to download and export a full file describing the trip in a number of useful formats.

The current version of the tool automatically logs in as a privileged user and synchronises a local list of trips with the available trips which it can access on the server. It shows a summary of each trip, including the time it was recorded, the reason for the trip and the route type. Trips of interest can be double clicked in the list and a further window opens. This window has two tabs, one provides more detailed trip information and the other allows the trip to be exported as a file. At the time of writing, (Database Reader Tool v1.1) there are two supported export formats, .gpx and .txt.

This tool was designed with the intention of being used by researchers to quickly and easily access the structured trip data stored in the server database. This data would be inaccessible without such a tool and as a result is vital to meet the project objectives.
<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 13306355654581.gpx</td>
<td>19/03/2012 13:08</td>
<td>GPX File</td>
<td>1 KB</td>
</tr>
<tr>
<td>34 1331109949833.gpx</td>
<td>19/03/2012 13:08</td>
<td>GPX File</td>
<td>1.45 KB</td>
</tr>
<tr>
<td>97 1331570000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>34 KB</td>
</tr>
<tr>
<td>98 13315700000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>122 KB</td>
</tr>
<tr>
<td>99 133157000000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>138 KB</td>
</tr>
<tr>
<td>100 1331570000000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>95 KB</td>
</tr>
<tr>
<td>101 13315700000000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>98 KB</td>
</tr>
<tr>
<td>102 133157000000000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>122 KB</td>
</tr>
<tr>
<td>103 1331570000000000000.gpx</td>
<td>19/03/2012 13:07</td>
<td>GPX File</td>
<td>138 KB</td>
</tr>
<tr>
<td>106 1331800575693.gpx</td>
<td>19/03/2012 13:09</td>
<td>GPX File</td>
<td>154 KB</td>
</tr>
<tr>
<td>110 1330707778672.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>130 KB</td>
</tr>
<tr>
<td>111 1330803794517.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>53 KB</td>
</tr>
<tr>
<td>112 1330815995252.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>103 KB</td>
</tr>
<tr>
<td>113 1330940194593.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>75 KB</td>
</tr>
<tr>
<td>114 1330967698626.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>144 KB</td>
</tr>
<tr>
<td>115 1331025337914.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>145 KB</td>
</tr>
<tr>
<td>116 1331054684190.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>66 KB</td>
</tr>
<tr>
<td>117 1331112083426.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>66 KB</td>
</tr>
<tr>
<td>118 1331155112582.gpx</td>
<td>19/03/2012 13:05</td>
<td>GPX File</td>
<td>66 KB</td>
</tr>
<tr>
<td>119 1331281865528.gpx</td>
<td>19/03/2012 13:06</td>
<td>GPX File</td>
<td>66 KB</td>
</tr>
<tr>
<td>120 1331309373061.gpx</td>
<td>19/03/2012 13:06</td>
<td>GPX File</td>
<td>66 KB</td>
</tr>
</tbody>
</table>

*Figure 2.15: Downloaded Cyclist Data*
2.2 Target Platform

When designing any application it is important to decide on a target platform early on. The platform will influence the language to be used and the resources available. The developer’s proficiency with the chosen language will also influence the quality of the project. This application, because it needs to be carried with cyclists, needed to be compatible with a mobile platform. The options available were Google’s Android, Apple’s iOS and RIM’s Blackberry OS. Other platforms were immediately discarded due to their lack of prominence (Symbian) or their relative newness (Windows Mobile). Once a shortlist of platforms was established it was necessary to evaluate the various advantages and disadvantages offered by each. In the end it came down to three main factors.

2.2.1 Why choose Android?

The first factor to be considered was whether or not there was a clear bias towards a device among cyclists. After consulting a number of cyclists it was determined that phones running iOS or Android were far more prolific among cyclists than Blackberry devices. The difference between the number of people using Android and iPhone wasn’t sufficiently large enough to favour one over the other at this stage so they were critiqued further.

Android applications are programmed in Java, with the option to code portions in C++. iOS applications on the other hand must be written in Objective C. For a developer with limited experience of either platform, Android would seem to have the gentler learning curve of the two platforms. This perception was supported by opinions voiced on many online app development forums. At a glance the supporting documentation for Java and Android appeared more comprehensive than that of iOS.

The decision to choose Android as the target platform was finally made when development costs were taken into account. To create applications for iOS it is necessary to use an Apple’s OSX as the development environment. The iOS device emulator bundled with the development tools is far superior, in terms of speed, stability and functionality, to the Android emulator provided in the Eclipse IDE. However the ideal situation would be to use an actual device for testing purposes. The prospect of having to purchase an Apple Mac and the ready availability of an Android phone eventually swayed the decision in the favour of Android.

2.2.2 Multiplatform support

Developing the application to work across multiple platforms was considered as an option. In the end a few factors resulted in the decision not to implement multiplatform support.

RhoMobile [9] was one cross-platform development environment that was considered as it supports all popular mobile platforms. A developer would create the core of the programme using RhoMobile and then implement a different interface for each operating system. However, it was decided that if problems were to arise during development with some of the more involved services, like registering GPS Location Listeners, would be better
documented in the primary languages of the platform. Furthermore it was thought that the effort required to integrate each system interface would cut short the time that could be spent on the core of the project. The concern was that the motivation of the project could diverge from the creation of a cycling application and data collection source to an equally important investigation of the experience of multiplatform development. Unfortunately it was not possible to pursue both avenues given the project time constraints.

2.3 Software Requirements
There are a few properties that are highly valuable in any software implementation. For this project the most relevant software requirements were identified. Below are a brief explanations of each requirement. In the next chapter we will see how choices during implementation affect these general requirements.

2.3.1 Robustness
Robustness is a property of software whereby it should not stop working for any reason. It should have a high fault-tolerance such that it would continue to work even if one or more of its components fail. For example, if the user inputs an invalid login password, the application should not crash but instead elegantly identify the problem. If a critical error does occur the system should fail gracefully and inform the user of what has gone wrong. This characteristic is essential for high-availability systems. In this project the goal is to maximise the uptime of the server and minimise the crashes in the mobile application.

2.3.2 Extensible
Extensibility is a measure of how easy or otherwise it is to add additional components and functionality to the core of a program. In this project it is expected that additional features will be added as cyclists feedback their needs and desires. The core of Rothaím is the trip recording functionality. This was the first feature to be completed and then others, such as statistics and history, were added incrementally to complement and expand the application’s functionality. Similarly, the server is designed to allow a developer make it do different things. Its core responsibility is to enable the uploading and downloading of data. In later versions a plug-in was developed provide a control panel for the admin to securely and quickly change server properties such as the detail level of logging. These are examples of extensibility in the project software.

2.3.3 Scalability
This characteristic affects the reliability of software. If a service is getting more requests than it can respond to within a reasonable time then something has to be done to correct this. One way would be to optimise the software and try to minimise the execution time. However, properly written software will already be well optimised. Therefore throwing more hardware at the problem is often the preferred solution. This isn’t a bad idea and usually works out cheaper than doing complex optimisation. However, scaling a system presents a fresh list of problems. The major issues surrounding scalability include difficulties
supporting distributed services, scaling to manage huge hit rates, quality of service concerns, architectural issues and performance of back-end services [10]. While Rothaím is unlikely to be as popular as to experience some of these problems it is entirely possible that the current server, running in a VM with finite system resources, may be insufficient to deal with the traffic and so it is prudent to consider scaling up as an eventuality.

2.3.4 Usability
Usability is one of the most important characteristics of any software. If software is fit for purpose with a reasonably simple interface then it is of little value to anyone.

2.4 Security
The developer is responsible for ensuring that any user details, anything from route data to email addresses, are stored safely and securely. This requires the system design to provide secure storage and transmission of this data.

Security of data is of paramount importance in any software that collects sensitive user information. If the software were ever to be released publicly, say on the Android Marketplace, then the system needs to be more secure than if it were being used exclusively for volunteers. The Data Protection Act, 2003 [11] clearly outlines the rights users and the responsibilities of the data collector.

One of the best ways to protect a user is to educate them. Where user information is collected the application uses labels to identify the data. Furthermore, the application’s Privacy Policy gives information about what data is collected, for what purpose and how and when data it is collected. It also states the publisher’s responsibility and efforts to keep their data secure. Education is the user’s first defence against data loss or theft but the system has to nonetheless secure itself against the worst case scenario.

2.4.1 Security Measures

2.4.1.1 Authentication
As described in the above design section the server makes use of a proxy. The proxy forces all potential users to authenticate themselves with username and password login credentials. User data is only accessible once the proxy has confirmed that the supplied credentials are valid. This ensures that errant requests cannot access the private information of any user.

2.4.1.2 Session Keys
Authenticating each request with a username and password is can be computationally expensive. This is because the proxy has to query the MySQL database each time it wants to confirm that the credentials are valid. A solution to this is to make use of session keys.

Session keys are temporary tokens given to a client so that it can send requests without having to authenticate itself each time. The session keys created in this project are valid for
15 minutes after the time of issue. Once a session key expires a client must re-authenticate to get a new key. This ensures that even if someone intercepts the session key that they will only be able to use it for a short time.

2.4.1.3 HTTPS
HTTPS itself is not a separate protocol to HTTP. Rather it is ordinary HTTP which is used over an encrypted TLS connection. Immediately this means that data transferred across the network is encrypted. The Tomcat container used in the server is configured to accept HTTPS connections on port 8443. In addition to the encryption benefits provided by the Presentation Layer of the OSI Model HTTPS also facilitates server authentication.

When a client receives a response from a server it contains a copy of that server’s public certificate. Since the server certificate is not held by any certificate authority it is necessary for the client application to store a local copy of the server certificate to authenticate against. This setup provides reasonable protection from man-in-the-middle style attacks.

2.4.1.4 Multiple database user accounts
The server’s MySQL database has three available user accounts, each with varying levels of privileges. The lowest level of access is READ-ONLY, the next up is READ-WRITE and finally one has ROOT privileges. When the server uses the JDBC driver to interact with the database it uses the account with the minimum privileges required to complete the operation. The ROOT account is not used anywhere in the software and no record of the credentials are stored on the server. If an attacker breaches the system this makes little difference, but as protection against SQL injection attacks using multiple accounts is a reasonable precaution.

2.4.1.5 SQL Injection
A SQL injection attack involves the insertion a SQL query to a request. A SQL injection exploit can return sensitive user data from the database or modify the database. To perform SQL injection the attacker inserts a different condition in the input in order to effect the execution of predefined SQL commands.

Example:

```sql
SELECT * FROM user_table WHERE username='sUsername'
```

could be exploited by setting the sUsername variable to be

```plaintext
sUsername = 'anyone' OR 'true'='true
```

Instead of returning a single user’s password, email and other personal information as the original statement intended it would return the details of all users stored in the table.

According to The Open Web Application Security Project (OWASP), SQL injection is the most common threat to application security [12]. Luckily, knowing about this type of vulnerability makes it easy to defend against. This project makes uses Prepared Statements for all queries
made to the database. Using these parameterised queries prevent SQL injection. If an attacker realises that prepared statements are being used and restructure their injection to pass arguments the JDBC driver will detect this and drop the query [13].

```java
String statement = "INSERT INTO trip_table VALUES (0, ?, ?, ?, ?, ?, ?, ?, ?, NULL, NULL);";
statement = databaseConnection.prepareStatement(insertStatement);
statement.setString(1, username);
statement.setInt(2, locDB);
statement.setString(3, reason);
statement.setString(4, type);
statement.setInt(5, time_of_day);
statement.setLong(6, start_time);
statement.setLong(7, end_time);
statement.executeUpdate();
```

Figure 2.16: Parameterized SQL Query
Chapter 3: Development & Implementation

In this chapter, we look at how the proposed system design was actually implemented. The technologies used to create each of the components are outlined and the process of development is briefly described. The challenges that arose when implementing each component are presented. The solution used to overcome each challenge is revealed and a comment is made on how it impacts, positively or otherwise, on the software requirements. Finally, the chapter ends by highlighting the key features of the software that was developed.

The Eclipse IDE was used to develop all Java and XML files for the project. Notepad++ was used to create all other code, such as SQL queries, .bat files, etc.

3.1 Application

This section describes the implementation of the Rothaím application. The application is developed for the Android operating system. Its target SDK version is 7 (Platform 2.1 Eclair) but is backwards compatible as far as SDK version 4 (Platform 1.6 Donut). At the time of writing the application can be run on 99.6% of active Android devices [14]. Android applications are developed using the world’s two most popular objected oriented languages [15], Java and C++. Most applications are developed using Java. They are run using Android’s Dalvik virtual machine. However, if the performance of some code needs to be improved a developer may choose to use the Native Development Kit (NDK) to develop using C++. Rothaím had no need for such optimisation and was implemented entirely in Java. At the time of writing the application had 5,736 lines of executable code in 84 classes and 7 packages.

3.1.1 User Interface

The user interface is the part of the application that the user can view, navigate and interact with on their screen.

3.1.1.1 Technology

User interfaces in Android are created by defining the desired layout in XML. Alternatively, if required, they can be programmatically generated. Rothaím uses combines both of these approaches. Simple layouts are created using XML exclusively. However, more complex screens, such as screens with maps, lists and tabs need to have their contents set programmatically. All static resources, layout margins, static text, font sizes and so on are also all stored in XML files.
3.1.1.2 Development
The Eclipse IDE provides a very useful layout creator. It has a panel of components which can be dragged and dropped to build the layout. The properties of views and objects can also be set with this tool. XML source code is generated in the background. Each new screen in Rothaím extends the DefaultActivity class. This provides the Activity class that every new screen requires anyway as well as some static variables and utility functions used by multiple user interfaces.

For a user to interact with the interface the screen needs a whole arsenal of listeners on buttons, lists, icons, events and so on. To add a listener to buttons and views defined in the XML layout the Activity's onCreate method must be overridden and the resource imported. This application achieves this by grabbing the memory address associated with the resource's id from the automatically generated R.java file. It is then possible to add listeners to the object programmatically.

3.1.1.3 Challenges
The only programming challenge of interest involved with the user interface was with creating lists and tabbed frames. One difficulty was that these content containers were completely different to the previously implemented views.

Furthermore, once the list views were created correctly with the listeners being applied a fresh problem was discovered. Once a list element is no longer visible it is unbound from the view and the row item is allocated for new contents. This led to a problem where the row's listeners didn’t correspond to the contents of the row.

Another problem arose when users began testing the application. When they would perform operations such as uploading large trips they would sometimes think that the application had frozen or crashed when the uploading ring did not disappear in a timely fashion.

Finally, a problem which didn’t present itself immediately was the complexity of supporting the application on a range of devices from many different manufacturers. With Android phones being made by Google, HTC, Samsung, Motorola and so on there exists a huge variation in the resolutions, aspect ratios and screen dimensions in target devices. This made it very difficult to design an interface that scales well to each device.

3.1.1.4 Solution
The solution to this was simple. Android offers a ListActivity and TabbedActivity to the developer to simplify the process of embedding content inside of the parent containers. Custom layouts needed to be created to define how a single list entry would be laid out. The HistoryActivity was made to extend the ListActivity class and a CustomHistoryAdapter set as its contents. This custom adapter class extends Android’s BaseAdapter class, which is the standard was of storing and interacting with list contents, and implements the Filterable class, which as the name suggests facilitates filtering of results. The adapter had to be
Further customised to add an onClick listener to the entire row, which when clicked shows more details about the trip. However it excludes the upload status icon at the end of each row. This icon is reserved for a different listener which when fired attempts to upload the trip to the server.

Although it was immediately apparent that there was a disparity between row contents and listeners on the History screen it was moderately difficult to identify the cause of the problem. Android recycles list items for performance purposes. Reusing them allows the ListView to scroll more smoothly. After looking again at where values were assigned to the ListView, the code was rewritten to initialise the new listeners only when content is imported into the recycled list items.

The problem of an unresponsive user interface was solved by threading all background processes separately so that they could run concurrently. This allowed user triggered events to be handled by the interface and the background processing to continue behind the scenes. This itself gave rise to another problem. Without disabling the user interface users got the impression that network interactions completed instantaneously. Therefore an elegant progress bar was implemented using the AsyncTask class’s call back functions. The program calculates how much data is to be transmitted and using the buffer size it calculates the percentage of the total transmission each successful update request fulfils. This results in an accurate indication of how long it takes for the operation to complete.
Supporting an assortment of screen sizes was a time intensive task. The solution used was to render separate images for different screen sizes and pixel density scenarios. Every custom graphic created for the application had to be saved in three sizes (36x36 pixels for low density screens, 48x48 pixels for medium density screens and 72x72 pixels for high density screens). The downside of this solution is that it increases the size of the application, but that was the compromise necessary to make the application that scale appropriately to a diversity of screens.

3.1.2 Services & APIs
Rothaím makes uses some external services and APIs to enhance the functionality of the application. It is primarily reliant on GPS location updates and Google’s maps service.

3.1.2.1 Recording
Recording is the process whereby the device detects and stores its position on the surface of the Earth at a specific time. This ability is at the very core of the applications functionality.

3.1.2.1.1 Technology
There were a number of options available when it came to choosing a position tracking finding technology. Undoubtedly the most popular and recognised of these is the Global Positioning System (GPS). This system makes use of 31 active satellites in medium Earth orbit to determine the location of the receiver on the ground. Each satellite continuously transmits two signals, on different carrier frequencies, containing its position in space, the time of its internal clocks, and other information [16]. One of these signals is encrypted and is only for the use of the U.S. Military. The other, lower resolution signal, is can be used for civilian applications such as route trackers.

Nearly all Android devices come equipped with GPS receivers. To determine the location these need to lock on to a minimum of three satellites in a short timeframe. The device can then calculate its position using triangulation [17]. Briefly, the steps are:

- Read signals from three satellites.
- For each satellite calculate the distance from it.

\[ \text{Distance} = \text{speed} \times \text{time} \]

Where \( \text{speed} \approx 3\times10^8 \text{m/s} \) and \( \text{time} = \text{time received} - \text{time transmitted} \)

- Draw a sphere around each satellite, taking the satellite location as the centrepoint and the distance from as the radius.
- With one satellite the device knows that it is somewhere on the surface of the sphere.
• With two satellites (provided they are sufficiently close together) the device knows that it is somewhere on the circular plane where the spheres intersect.
• With three satellites, with intersecting bounding spheres, the device can determine that its location is one of two points common to all three spheres.
• Whichever point is on the surface of the Earth is taken as the location.

Figure 3.2: GPS Triangulation [18]

Similar technologies that could have been implemented include the Russian GLONASS or positioning using WiFi hotspots and cell towers. GPS was chosen not primarily for its strengths but that it was the most accurate, most widely used and easily implementable option available. GLONASS is only supported on some mobile devices, specifically those with Qualcomm Snapdragon S2 and S3 CPUs, since December 2011 [19]. And the location returned by coarse network location techniques using cell towers and WiFi are only accurate to between 50m and 100m.

Assisted GPS (A-GPS) technology can improve the “Time To First Fix” (TTFF) by using an available network connection. It downloads orbital data about the location of satellites in the sky to which usually helps get a faster fix. Results vary depending on when the device’s GPS was last used and hence the integrity of its cached satellite information.

3.1.2.1.2 Development
Receiving location updates in Android is not particularly difficult. The process is very well documented on the Android Developers website [20]. Firstly, during installation, the application must seek permission at from the user to be allowed access to the device’s “Fine Location” privileges. Subsequently, when the application wishes to get location updates it must register its interest with the OS using a LocationManager to begin receiving location updates. In Rothaím updates are requested to be received twice per second.
When a location is received by the device it is propagated all listening applications. The RecordingService notifies its observers, namely the RecordingController, that a location update has been received. The RecordingController adds the location observation to a temporary Trip object. The Trip calls a function to connection to the internal SQLite database and adds a new row in the ‘Coords’ table.

When the user ends a trip the associated rows in the database set flags to indicate the applications transition from the RECORDING state to the SAVING state. The RecordingService unsubscribes from OS location updates, is stopped and unbound from the application context.

### 3.1.2.1.3 Challenges

Two of the greatest obstacles faced in this project pertained to GPS locations.

The first issue was with the length of time that it takes to receive the first location fix. This “Time To First Fix” (TTFF) is a common problem in GPS receivers. From testing carried out during the project it was found that the TTFFs of Assisted GPS (A-GPS) devices were between 5-30 seconds. For ordinary GPS devices, and all phones without a network connection, it can took anywhere from a few seconds to tens of minutes. This problem was only exacerbated if the receiving device began to move while still awaiting its first fix.

The second problem faced was ensuring that the data collected had acceptably low degree of error. GPS uses electromagnetic waves to transmit its signals and therefore cannot pass through metal. Unfortunately most cycling of interest for this project takes place in an urban area and as a result waves can be received after reflecting off a building as opposed to coming direct from a satellite. This sort of unavoidable phenomenon introduces inaccuracies into the location data.

Another problem, which was entirely the result of poor design, was that of saving. In the first implementation attempt all observations would be buffered to be written to memory only when the trip was completed. This implementation resulted in the application crashing on long trips because it was running out of memory.

### 3.1.2.1.4 Solution

No suitable solution was found to improve the TTFF of phones without a network connection. An option available was to use to cell towers and in-range WiFi networks to determine a coarse location and use that until a better approximation was available with GPS. However, it was decided that a location that was at its very best accurate to 50m was of no value for data collection purposes as a user could be on one of many streets in an area. For users with an available network connection a small packet is downloaded containing the latest positions of the satellites. It was found that this dramatically decreased the TTFF.
There was a lot more room for innovation with regard to the accuracy issue. The solution decided upon in the end was to increase the original sampling rate of 1 location every 5 seconds (12 locations/minute) to 2 locations per second (120 locations/minute). This solution was arrived at after investigating the effects of varying sampling rates on battery life. It was found that a user could comfortably complete two trips in a day and still have sufficient charge remaining for normal use.

Increasing the frequency of location updates does not improve the accuracy of the updates. Rather the additional readings can be aggregated to generate a better estimation of the true location. The volunteer testing later confirmed that despite the high sampling rate that battery drain was not having an undue negative effect on usability.

The issue with crashes occurring on long trips required the recording and saving design to be revisited. The process was re-implemented so that observations would be periodically flushed to memory when buffers became too full. Prior to this change, a user who saved a trip might have to wait a few seconds as the entire list of recorded observations was written to the database. Now when the user saves a trip a single flag is set in the database to indicate that it is marked as saved. Observation points which have been written to the database but not saved are cleaned by background threads which periodically verify the integrity of the data.

3.1.2.2 Maps
Maps are a used frequently in Rothaím. While recording a map shows the current user location. The history overlays the recorded route on a map so that the user can clearly see where they travelled.

3.1.2.2.1 Technology
Google API’s Google Maps add-on was chosen to provide the maps for the application. OpenStreetMap was considered as an alternative to Google Maps because the documentation pertaining to implementing Google Maps in Android was perceived to be superior to that of OpenStreetMap.

Neither of the above map APIs uses static maps. They both dynamically download their maps upon request. As a result any updates to the maps are automatically available on the phone application without the need for a software update. In hindsight, a major advantage of OpenStreetMap is the ability to cache maps for offline use. Unfortunately, to do so with the Google Maps API would violate the Terms of Use. Having the use of cached maps when no network connection is present would add significant value to the application for a user with no data connection.

3.1.2.2.2 Development
The Google Maps API is not included in the standard Android library. As a result the application must package Google API’s maps.jar alongside standard android.jar. Once the prerequisite library is included in the project, MapViews can be embedded in XML layouts.
This was an occasion when the layout creator tool did not provide drag and drop objects and so knowledge of the underlying code was needed to proceed.

An Activity that wishes to use a map must also extend the MapActivity class. Once it uses the layout as its content it can retrieve a reference to the MapView as it would any other button or text field. A MapController is implemented to programmatically pan and zoom. This functionality is required on startup and each time a location is received to keep the map centred on the user’s location.

The final step is to attain functioning maps is to register for an API key. For this a Google Account and MD5 fingerprint of the application’s certificate are required. A debug certificate is automatically created by the Eclipse IDE and a key generated from this certificate can be used while debugging. However, when the application is released it needs to be signed otherwise no device will install it. Java provides the keytool command to manage keys and certificates. After creating a certificate for the release build of the application its MD5 fingerprint is used to generate a new API key. This key then needs to replace the debug key for maps to be available on the application [21].

3.1.2.2.3 Challenges
The only major problem with the maps was with overlays. Overlays are markers supported by the API which can be placed on the map. The application creates an overlay for each observation and connects the marker to that of the next observation resulting in a plot of the trip. This worked fine for a small number of overlay markers. However, once the quantity of observations exceeded a few hundred the map interface began to slow down. The framerate dropped as the user panned and zoomed on the map. For trips with thousands of observation points it could take tens of seconds for the route to be drawn on-screen and then if the user panned or zoomed they would have to be redrawn all over again. This made the maps feature utterly unusable to view trips of any significant length.

3.1.2.2.4 Solution
Reading forum posts revealed that this was not an uncommon problem with the Google Maps API in Android. Unfortunately there was no easy solution. The problem doesn’t manifest itself on iPhone as the API is implemented differently for that platform. It was decided that reducing the resolution of the data drawn would improve the situation. So where there were, say, 2,000 observations to be drawn only every fifth one, 400, would be visible. This had two effects. The first and immediate one was draw time was reduced. Where a route would have taken 10 seconds to refresh it now took two. The second was that the integrity of the route was lost. The loss in data resolution meant that when zoomed in the overlay no longer fairly represented the recorded route. Despite these problems, this solution might have been appropriate except for the fact that it was not scalable. If a user recorded a trip of 10,000 observations by applying the above solution the application still had to trudge through drawing 2,000 markers.
A better solution was later found. It involved determining the maximum and minimum values for latitude and longitude of a trip. The map is zoomed out so that it can fit the entire trip and the data resolution is scaled to optimise performance. At this level of abstraction the missing data points are not noticeable. As the user then zooms in the resolution of the drawn data must increase, otherwise the lack of data points become apparent. However, as the map is zoomed the latitudinal and longitudinal ranges contract, therefore observation points which are no longer visible do not need to be drawn. This solution scales well to trips of all lengths.

3.1.3 Networking

The networking components of the application are used for communicating with the central server. The primary classes in the networking packages are NetworkController, AccountNetworkHandler, UploadData, DownloadData, and NetworkView. The CustomSSLSocketFactory class is implemented to improve the security of the data transmission but users may choose not to use it by disabling HTTPS on the Settings screen.

3.1.3.1 Technology

The networking classes make use of the Apache HttpComponents library (version 4.1.2) to assist in networking tasks. It provides a toolset of Java components for HTTP and its associated protocols.

3.1.3.2 Development

3.1.3.2.1 Network Controller

The NetworkController’s primary responsibilities are to behave as a manager for network connections and to provide utility functions to the threads it creates. When the application initialises the NetworkController the only thing it does it create a NetworkView to which it can publish UI updates. It does nothing additional until it is called again, usually due to some user action like clicking the upload button. On such an occasion the NetworkController will create a threaded class, which extends Runnable, to perform the desired operation.

During the course of the threads lifetime it will require the utility functions offered by the NetworkController. The static getNewHttpClient() function creates a new HttpClient object and configures it. Timeouts are set to define how long the client should wait for initial connections and for data on sockets. The character set and supported protocols (HTTP and HTTPS) are also defined. Finally a ThreadSafeConnection is initialised to manage multiple threads sending HTTP requests. The resulting HttpClient is returned to the calling object. The other two popular static functions that the NetworkController offers are the get() and post() functions which handle the transmission of requests to the server.

3.1.3.2.2 Account Network Handler / Download Data / Upload Data

Each of these classes extends Android’s AsyncTask, a child of Java’s Runnable class. They are tasked with performing a specific job, AccountNetworkHandler is responsible for logging in or registering a user, DownloadData downloads trips from the server and UploadData
uploads them. Because of their common parent class they are all structured the same way. They have the doInBackground(), onProgressUpdate() and onPostExecute() functions overridden to perform their custom tasks. The bulk of the work is done in the doInBackground function, which uses functions to complete sub-tasks. When completed, these sub-tasks call publishProgress() to trigger a chain of callback functions which results in the UI being updated. Just before the AsyncTask exits it runs the onPostExecute() function which is where the final result of the request is returned to the NetworkController. If the operation was not a success the NetworkController will check the result to see if it corresponds to one of its 16 recognised error codes. Below is a pseudo-code example of AsyncTask's three functions being applied in the UploadData class.

```java
int doInBackground(Object... params)
{
    configureClient();  // Get a HTTPClient from NetworkController
    authenticate();  // Verify session key with server
    uploadBasicTripInfo();  // Send trip reason, type, time, etc.
    while (observations_remaining)  // While not all points sent
    {
        uploadSection(chunked_observations);  // Send next chunk
    }
    return finish();  // Send trailer information to server
}

void onProgressUpdate(int progress)
{
    // Notify the NetworkController which in turn updates the UI
    controller.updateProgress(progress);
}

void onPostExecute(int result)
{
    // Return the operation result
    // It will either be a success or 1 of 16 unique error codes
    controller.setResult(result);
}
```

Figure 3.3: General Form of a Networking AsyncTask

### 3.1.3.2.3 Network View

This class is solely responsible for creating on screen messages to reflect the condition of the network. Toast messages, the least intrusive messages, briefly overlay some semi-transparent text before fading away. Messages using Android’s AlertDialog display the message in a dialogue box with one or more buttons to control the response. These are used to show all error messages; they will remain on screen until dismissed by the user. Where the result of an operation is a recognised error the NetworkController will instruct the NetworkView to provide additional buttons which the user may select to attempt to fix the problem. The final type of user interface element provided by the NetworkView is the
ProgressBar. Functions are periodically called to update it so the user doesn’t begin to think that the application has crashed.

![Network Alert Dialogue](image)

Figure 3.4: Network Alert Dialogue

### 3.1.3.3 Challenges
The challenges experienced with respect to networking were self-inflicted. In the first implementation, a poorly thought through design decision led to File Transfer Protocol being used to transmit trip data. This archaic protocol provided unreliable transmission over the network. The original idea was that the backend would be much easier to configure if a simple, out-of-the-box FTP server could be deployed to deal with requests. While this perception was true the resulting system didn’t come near to meeting the projects software requirements for usability, robustness or security. A better solution had to be found.

### 3.1.3.4 Solution
The FTP networking was scrapped and HTTP was implemented instead. The HTTP connection to the server didn’t exhibit FTP’s problem of dropping the connection mid-transfer. The reason for this is that FTP must maintain two connections to the server, a control connection for commands and data connection for the file contents. The control connection is usually idle and as a result the firewall, which is attempting to clean up unused TCP/IP connections, frequently closed it before the transfer on the other connection was complete [22].

HTTP connections, on the other hand, only require a single port to transmit requests and receive responses. For large files in particular, using HTTP can be slower due to the
overhead of each request and response needing some header information. However, against the improved reliability, which comes with not having idle connections dropped, this trade-off is more than acceptable. This greater robustness of the software means that fewer errors occur which in turn positively impacts on usability, making the users interaction with the application less frustrating. A HTTP proxy server was developed to help improve the security but unfortunately the solution was insufficient to prevent eavesdropping of man-in-the-middle attacks. And so a final iteration was necessary.

HTTPS was implemented to solve the security issue. HTTPS is HTTP used over a SSL/TLS connection. To implement this on the client-side the CustomSSLSocketFactory class was created. Objects of this class used when the NetworkController creates a new HttpClient. Upon creation an instance of this class will read the local copy of the real servers public certificate. From this it extracts the server’s public key. When a response is returned from the server the CustomSSLSocketFactory verifies that the server is genuine and not a rogue server performing a man-in-the-middle attack. To do this is reads the unverified server’s X.509 Certificate from the response and first of all ensures that it is in data. If it has not expired the application then verifies it using its copy of server’s public key. The transfer of data will only occur if the server can be verified by the application. Furthermore the SSL/TLS connection employs asymmetric key encryption so that the data on the network is not sent in plaintext. Implementing HTTPS is not without its drawbacks. The server’s certificate is self-signed. As a result if the server certificate is ever changed or additional servers are added to deal with traffic load the application will require updating to add the new certificates to the local keystore. This presents difficulties with regard to the project’s scalability. Aside from this there is also a performance hit due to the added encryption and server authentication. However, the security of a user’s data took priority and it was decided that the benefits of HTTPS greatly outweighed the downsides.

3.1.4 Database
The internal SQLite database provides secure and structured storage of all trip details and location observations.

3.1.4.1 Technology
When choosing the technology for the database to most commonly used forms of data storage in Android applications were examined [23]:

- SharedPreferences: data is stored as key-value (name-value) pairs. The application uses SharedPreferences extensively for storing user preferences and data that must be persistent between Activities. The main obstacle to using this for storing trip and observation data is that each key can only have one value associated with it at any time. Since Android API level 11, functionality has been added to SharedPreferences to allow arrays of data to be associated with keys. Rothaím doesn’t make use of this and as a result is backwards compatible as far as API level 4.
- Internal Storage: this type of storage requires the programmer to create a file in internal memory and write to it using an OutputStream. However, unless meta-tags are manually inserted, there is no structure to the data. Reading the data in at a later stage incurs a large parsing overhead.

- External Storage: a file is created on the phone’s SD card. While most devices have SD cards for external memory, this is not a guarantee. It is also the slowest of all the data storage types. Its only advantage is that if files are written to external memory, a user can remove the file themselves and use it in a visualisation programme like Google Maps. Rothaím contains hidden functionality which enables the publishing of trips to external memory in the popular GPS Exchange Format (.gpx) to external storage.

- SQLite Database: data is stored in a structured database. The database is private and cannot be accessed by other applications ensuring the security of the data. SQL queries are used to insert and extract information from the database.

3.1.4.2 Development
The DatabaseAdapter class is used to create and handle all subsequent interactions with the underlying application database. Upon initialisation the DatabaseAdapter checks to see if the database exists. The first time it runs, it will need to create it. The database is comprised of two tables, the Trips table and the Coords table. The PARENT_TRIP field in the Coords table references the ROW_ID of its parent trip in the Trips table. When a trip is removed from the Trips table, for example when a user chooses not to save it, all corresponding observations in the Coords table are automatically deleted.

The DatabaseAdapter class presents an interface of twenty functions to allow facilitate everything from creating a new trip in the database to fetching observations in a defined range for chunked uploading of data. They are used by the user interface for displaying routes and statistics, by services for recording, by the maps API for trip visualisation, and by the various networking components for retrieving information to upload or to store downloaded data.

<table>
<thead>
<tr>
<th>General Database Functions</th>
<th>Recording Functions</th>
<th>Other Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>open()</td>
<td>createTrip(Trip)</td>
<td>deleteAllCoordsForTrip(long)</td>
</tr>
<tr>
<td>openReadOnly()</td>
<td>firstFix(long, long)</td>
<td>fetchTrip(long)</td>
</tr>
<tr>
<td>close()</td>
<td>updateTrip(long, float)</td>
<td>fetchAllTrips()</td>
</tr>
<tr>
<td>cleanTables()</td>
<td>updatePauseTime(long, double)</td>
<td>fetchAllSavedTrips()</td>
</tr>
<tr>
<td></td>
<td>updateStatus(long, int)</td>
<td>fetchAllCoordsForTrip(long)</td>
</tr>
<tr>
<td></td>
<td>updateUploaded(long, int)</td>
<td>fetchCoordsInRange(long, int, int)</td>
</tr>
<tr>
<td></td>
<td>addCoordToTrip(long, Location)</td>
<td>fetchMaxSpeed(long)</td>
</tr>
<tr>
<td></td>
<td>setNodes(long, int)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>finish(long, int, long)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: DatabaseAdapter Methods
## 3.2 Server

The section describes the development of the project’s server. At the time of writing the server had 1,083 lines of executable code, in 12 classes and 6 packages.

### 3.2.1.1 Technology

The server itself is supplied by the Trinity Centre for High Performance Computing. It is a Virtual Machine running a distribution of Linux called Red Hat Enterprise Linux version 5.6. It has the Security Enhanced Linux feature enabled to prevent modification of other parts of the server outside of the home directory. This means that ROOT privileges were not available during server configuration and development. Issues arising from this configuration are outlined later in this section of the thesis.

The web application framework chosen was Apache Tomcat version 6. In the application discussion above the project invested time had already deploying a FTP server and found that it didn’t meet the requirements. The alternative technology to Tomcat would have been Glassfish. Glassfish, created and maintained by Oracle, is the newer of the two technologies and comes with many benefits and add-ons. However, it was decided that for Tomcat’s minimalistic design (i.e. it supports Servlets and Java Server Pages and not much else) was sufficient for this project’s needs.

The backend database is an implementation of MySQL. The alternative here was to use Oracle. Having had previous experience with Oracle databases the developer had a bias towards using MySQL in order to learn new skills. Bias aside an analysis of the merits and demerits of both technologies were still conducted. The main attractions of MySQL were its performance and its price. Oracle’s free version only allows databases to grow to 4GB. MySQL has no such imposed limitation. Looking at other projects which make use of databases small scale companies and applications tend to lean towards MySQL whereas larger enterprises used Oracle. All things considered MySQL was chosen as the technology upon which the server’s database was built.

In MySQL a developer can use a variety of engines for storing their tables. New tables default to the MyISAM engine unless otherwise specified. In this server, however, it was decided that the InnoDB engine better suited the project needs. Table 2 below shows a summary the factors which of the decision making process [24].

<table>
<thead>
<tr>
<th>MyISAM</th>
<th>InnoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proven Design</td>
<td>New design – not as well tested</td>
</tr>
<tr>
<td>Simple</td>
<td>More Complex</td>
</tr>
<tr>
<td>Faster than InnoDB</td>
<td>Transaction safe (strict data-integrity)</td>
</tr>
<tr>
<td>Table locking</td>
<td>Row locking (faster when many transactions are occurring on the same table)</td>
</tr>
<tr>
<td></td>
<td>Foreign Key Constraints</td>
</tr>
<tr>
<td></td>
<td>Better crash recovery as it keeps logs of</td>
</tr>
</tbody>
</table>
Data integrity and foreign key constraints were the main attractions of the InnoDB engine. In particular, being able to cascade deletions in one table to the related entries in child tables reduces the volume of code that needs to be written later on to manage the integrity of the database. Finally, from understanding how the server would interact with the database, it was realised that MyISAM may not necessarily be the faster technology to use in this implementation. The most common database operation will be either reading or writing to the observation_table in the database. If MyISAM were to be used the entire table would be locked when a row was being inserted or retrieved. This would prevent another servlet from accessing the table until the lock is released. Say, for example, two users were simultaneously uploading trips to the server. They would both be writing blocks of observations to the observation_table at the same time. To prevent conflicts locking the table is a reasonable measure however it will likely impact on the performance. Now add a user who wants to download some observation points and the system suddenly presents a scalability problem. However, if the same table is using InnoDB only the row in the observation_table is locked and the only time that a lock will be contested is if a user tries to simultaneously read the same trip from the server on two devices.

3.2.1.2 Development
The larger part of time spent of server development was spent on configuring the Tomcat container. There were a multitude of settings that needed to be tweaked and set to get it to work. Firstly, Tomcat had to be configured to listen to incoming HTTP requests on port 8080 and later re-configured to listen for HTTPS requests on port 8443. The next step was configuring the filepath map correctly to the server. The above configuration took place in Tomcat’s server.xml and context.xml configuration files. Furthermore, once the request reached the correct server implementation within Tomcat the URL had to be remapped to the correct servlet within the project. This mapping is defined in the server project’s web.xml configuration file. Further configuration needed to be done so that Tomcat would include the JDBC driver library when it launches the JVM.

The implemented system adhered closely to the proposed design with the AuthenticationServlet behaving as a proxy service at the front-end of the server. When forwarding requests to the MainServer it was important to understand the subtle, yet poorly documented, difference between a forward and a redirect. It was found that a redirect does not allow the header of the response to be changed by the other servlet.

The MainServlet makes use of two helper classes, much like the application’s NetworkController, to perform specific requests. The UploadHandler and the DownloadHandler classes deal with upload and download requests respectively. They both initialise a DatabaseAdapter to use an interface between themselves and the back-end
database. The UploadHandler provides functions to deal with the primary upload modes, namely create, update and finish. The DownloadHandler offers variety of download specific functions, list all of a user’s trips and download to fetch a single trip. For each forwarded server request Tomcat creates a threaded instance of MainServlet eliminating any further need to thread the helper classes.

The MySQL database was developed through a PuTTY shell. Scripts were written in the Notepad++ text editor and executed on the server using the command below. This approach allowed the tables to be quickly modified if required.

```
mysql > source create.sql
```

Subsequently it was discovered that foreign key constraints and cascades were not being enforced. No errors were reported by MySQL and it was eventually discovered that in this MyISAM was the default table engine. The database had to be recreated where the tables were defined to be powered by the InnoDB engine. This solved the foreign key constraint problem.

On another occasion a problem was discovered where the database was losing the precision of floating point numbers, observation latitudes and longitudes. At this time the user trial had already begun and some users had already registered accounts and uploaded data. To guard against accidentally deleting their accounts and information when modifying the observation table the mysqldump tool was used to create a file which could be used as a restore point. The necessary table modification was trivial (change the data type of latitude and longitude to DOUBLE) but creating backups of the database contents became a regular fixture.

### 3.2.1.3 Challenges

The first challenge associated with hosting a server in Trinity College was connecting to it. The college network administrators observe strict policy regarding allowing external devices to communicate with a server on their network. This meant that it was impossible for any mobile device which was not connected to the college network (which at the time of writing is not officially supported) to communicate with the server due to firewall.

Configuring Tomcat and MySQL was another difficulty. As mentioned in the introduction to this section the server uses SELinux to jail the user. Without ROOT privileges it was impossible to install additional software and editing configuration files in protected directories was equally impossible.

Debugging a remote server is a complex procedure compared to debugging a local server for a developer who is new to both remote debugging and the technology to be debugged. In hindsight, a little understanding of how Tomcat is configured and operates (and unrestricted firewall) would have made it possible to debug the server through the Eclipse IDE from the beginning.
3.2.1.4 Solution

The existence of firewall was an extremely problematic issue and prevented any meaningful testing of the software. A firewall exception request was submitted to the network administrators, justifying the need for an exception, but no action was taken or response received. Therefore the decision was made to side step the “red tape” and to directly approach Trinity Centre for High Performance Computing in the hope that they would be more understanding of the project’s needs. Fortunately they were more than willing to be of assistance and not only did they fast track the firewall request but they also provided the server. When it became apparent that opening port 21 for FTP and port 22 for SFTP it was necessary to submit a second proposal to open ports 8080 for HTTP and port 8443 for HTTPS.

For security reasons it was not feasible to get ROOT privileges for on the server to configure the needed components. However, the TCHPC team obliged to implement any specified changes that were submitted. On a number of occasions, while configuring the server, debugging sessions were held to attempt solve outstanding problems. It could be argued that having to work with the TCHPC team rather than be granted ROOT privileges slowed the development process. However, any delay was overshadowed by the information gained from the experience, particularly related to servers and the Linux operating system.

The problem of debugging too was eventually overcome. The Utility class was developed to provide support for multiple levels of logging on the server. The Admin Control Panel discussed in the last chapter was initially developed to dynamically change the logging level of the server eliminating the need for a new build to be deployed. Arising from these changes a .bat script was written to speed the process of deploying a new server version on the server. The Tomcat Manager plug-in was setup to facilitate swift restarting of the server through a browser (although it requires admin credentials for security). Prior to this the only way to restart the server was through the command line and required ROOT privileges. Versioning was strictly enforced to ensure that it was simple to verify whether or not the build and server restart were a success.

From the perspective of a developer, the multitude of tools which now surround the server render the development environment markedly more usable than it was to begin with. The implementation of the server itself meets the project security requirements, as it uses a proxy for authentication, guards against SQL injection attacks and provides support for HTTPS connections with its self-signed certificate. Finally, the Database Reader Tool was designed and developed after the server was built. Nonetheless it integrates seamlessly with the server nicely demonstrating the extensibility of the server implementation.
3.3 Key Features

3.3.1 For Users
Since this project is primarily concerned with the collection of cycling data it is imperative that users enjoy the entire user experience. Furthermore if the product adds significant value to the user then they will be inclined to use it more frequently. With this in mind we identified three of the most important features would add value Rothaim for a cyclist.

3.3.1.1 Interface
An intuitive and streamlined interface is one of the most important features of any consumer or enterprise product. This principle is so important that it isn’t even specific to software development. A poor interface is one where users find it difficult to perform common tasks or where there is an unnecessarily steep learning curve. This type of interface eventually frustrates users and in the worst case scenario they may stop using the product entirely. On the other hand, a product that is simple to use can integrate itself into the lifestyle of a user. With this in mind careful consideration was given to usability at the design stage.

It is clearly undesirable to have the user pause to configure a number of fields before starting each journey. If you are about to depart on a journey the last thing that you want is to have to pause to configure a number of fields in a phone application. Therefore a concerted effort must be made to minimise the steps and time required to perform an operation. In the Rothaim application this is achieved in a number of ways for the various functions. On the plan route screen a user indicates the reason for their trip and their perception of the route that they intend to take. To streamline the process two lists of preset options are offered to the user. The benefits of this are twofold. The first is that the user doesn’t need to waste time typing out a line of text. Secondly, when it comes to analysing the aggregated data it will be easier to associate trips with similar properties because of the constraints imposed on the user.
Another example of simplicity of performing operations is with the upload feature. The user can upload at different points in the application. The default behaviour of the application uploads the trip at the time of saving. It is expected that making the user upload each time they save should improve the total number of trips uploaded. However, if a user chooses to upload at a later time the process is equally simple, requiring them to simply press a large icon next to the trip record in their history.

The visual design of the application is not vitally important for a closed testing group. However if the application were to ever be released publicly potential or first time users get their first impressions by the theme and graphics of the application. It isn’t necessary to have exceptional artistic talent in order to achieve this. Graphics needed for specific operations can be created or obtained from public domain image repositories. Android provides a wealth of stock resources that can be harnessed by a developer. Using the default menus, buttons, layouts and so on leads to the application being instantly familiar to users who are accustomed to using Android devices.
The importance of descriptive error messages cannot be overstated. Error messages allow a user to troubleshoot their own problems without having to seek assistance from the developer. For example, if a user tries to upload data without first authenticating themselves they will not have a valid session key to present to the server. Obviously the user doesn’t need this technical information and it is enough to simply offer the solution. In this case it would be to log in before retrying the last operation. Nowadays most users would expect these sorts of features as standard.

3.3.1.2 Recording a Trip
The recording feature is the core of the application. Through it the application gains further value by providing useful and interesting statistics about the user’s behaviour that they want to view. While recording a trip the user can see the time elapsed since they first began their journey, their current speed and the distance that they have travelled. The application can also be used for navigation purposes. A user who is lost can find their bearings by seeing their current location clearly overlaid on a local map. The map allows panning and zooming so that the map can be explored.

Once the user has begun recording a trip they still have some control over the application behaviour. Using simple buttons at the base of the screen they can pause recording their journey at any time. This grants users further control over the privacy of their data and reduced the perception that they are constantly being monitored. To resume their journey they simply press the Resume button. This starts up a background service to respond to GPS location fixes and thus recording is effortlessly resumed.
3.3.1.3 History
The history needs to provide an overview of all saved journeys. When scrolling through many journeys a user should be able to see the most important information about a trip at a glance. However, if they want more information about an individual trip they can explore it further and view more detailed information.

A well-structured history interface is the feature that will give longevity to the application and maintain users’ interest. Statistics that interest users include the total journey time, the distance travelled, the average speed, their maximum achieved speed, the calories burned and so on. As users log more trips they will look back through their history and compare their most recent performance with their past performances. The more engaged cyclists will attempt to beat their top speeds or vary their routes to achieve a shorter journey time for their commute.

If a user is less performance orientated they can still utilise the history. It is possible to visualise recorded routes as overlays on an on-screen map. Users can review their history to see where they have been travelling and what landmarks they may pass along the way. This could cause users to realise that the street that they’ve never travelled before might actually be a potential shortcut for one of their popular routes. Another outcome could be that they see a place of interest marked on the map which just off the route that they regularly travel, such as a church or shopping centre, prompting them to stop and explore the next time they pass by.

This behaviour will result in the emergence of new behavioural patterns in these users.

3.3.2 For Researchers
Assuming that the application is successfully implemented and becomes a popular tool with the cycling community then researchers will have access to a rapidly growing repository of
information about cyclists that has previously been impossible to come by in large quantities. However, this project is also responsible for ensuring that the data collected is rich, accurate and accessible. The project can be classified as a failure from a research perspective if the cycling data lacks any one of these three characteristics.

3.3.2.1 Accurate Location Data
The location data that is collected from cyclists should be accurate enough for the purpose of analysis. It is no use recording data if the latitude and longitude are not fine enough to indicate which road a user is on. Ideally we would like to pinpoint the exact location of the cyclist on the road at any given time but this may not be possible in practice. In Chapter 4 we will see that accomplishing this task demanded additional thought.

3.3.2.2 Trip Meta-Data
As interesting as it may be to track the routes that users travel everyday it is even more interesting, particularly from a behavioural analysis point of view, to know why they have chosen these routes over another. Knowing the motivation for a user’s travelled route development of all sorts of hypothesis, such as factors which influence route choice depending on trip reasons. Things become really interesting when these theories are then matched against the users own perception of that route. A user may travel a route because they perceive it as being safer than an alternative. Having auxiliary data about accidents and so on from other sources allows an investigator to either prove or disprove the validity of that perception. By simply gathering two meta-data fields, trip reason and trip type, from a user research can depart from traditional path analysis to explore new areas such as identifying patterns in the behaviour of cyclists, both individually and as a transport group.

3.3.2.3 Usable File Formats
It is unrealistic to expect that everyone who will be using this data for the research purposes should have to learn the MySQL database language to access it. Most transport researchers are experts in their own field and not in computer science. It is important to them therefore that the data they download is stored in a location file format with which they are familiar. These include but are not limited to .gpx, .kml, .xls and variations of .xml schemas. A lot of time is dedicated to converting data sources to the correct file format. Software which exports files which are instantly usable for analysis is of high value to a researcher.
Chapter 4: Evaluation

In this chapter, we will evaluate the suitability of the implemented solution for fulfilling the project objectives. To do this we first discuss the testing period where volunteer cyclists tried the Rothaím application. The ethical obligations of a software developer are discussed and the steps taken to ensure that this project observes these obligations. The need for additional tools to assist analysis will be justified and the tools described. The user feedback to the trial period will be discussed, as will the insights gained from the data itself. The software will be evaluated to see if it adheres to the general software requirements proposed in the design. The differences between a BETA testing and debugging are contrasted against the applications release to volunteer testers. Finally, the entire system arising for this research will be critically analysed to appraise whether or not it is fit for its intended purpose.

4.1 User Testing
This project included a three week testing period where the application was distributed to a group of volunteer cyclists.

4.1.1 Ethical considerations
The primary responsibility when releasing software must be the safety of the user. It would be crassly negligent to publish an application which may harm users.

4.1.1.1 User Protection
This project involves an application which is used by cyclists. Already they are a group who are exposed to a high level of risk on the roads. They do not need software which will detract from their attention while they are cycling. As a result the Rothaím application does not encourage users to interact with the application while they are cycling. It achieves this by providing minimalistic features during the course of a trip.

Encryption standards have been compromised in the past and therefore the security of user data cannot be guaranteed. Recognising that fact in this project was empowering as it forced the implementation to strive to protect to the best of ability. This was achieved by using the many security features discussed in previous chapters, such as not publishing trips to the device’s external memory and using the industry standard security and encryption protocols.

A practical measure which improves user security is to only collect the minimum quantity of personally identifiable user information. For example, the application and research have no need for a user’s address or mobile number so they are not collected. In the unlikely event that an attacker was to breach the server database they would only find personally identifiable information that was collected, like usernames and passwords. It should be
stressed that every precaution has been taken to ensure that an attack like this cannot occur.

4.1.1.2 SCSS Research Ethics Application
To conduct any study involving people the School of Computer Science and Statistics requires the project investigator to submit a proposal seeking permission to carry out the study. The intention of this application is to ensure that the department authorities can judge whether the investigation is of an ethical nature. It also has the benefit of focusing the researcher’s attention on the ethical implications of conducting user trials. The supporting documents had to justify the need for a user trial and the clear benefit for the research and for the user. Documents also had to be written to summarise to the volunteer the objective of the project and their part in it. A consent form had to be signed by each volunteer before they could take part in the study to confirm that they had read the project information sheet provided and agreed to the terms and conditions. As part of the debriefing arrangements they were invited to a demonstration of the project and the developer was made available to answer questions or concerns at any time.

As part of the participant information document a Privacy Policy was written and a Terms and Conditions of use were drafted. The Privacy Policy in particular clearly outlines the entitlements of the user. It lists all information that is collected, what it is to be used for, the times at which it is gathered and the actions taken to protect it.

4.1.2 Distribution
Participants were drawn from the professional and social networks of the principle investigator. People who were known cycle regularly and owned Android mobile phones were targeted specifically. This network of people was harnessed through email, calls, face-to-face contact and social networking sites. Volunteers who agreed to participate were distributed with the application software for their phone upon signing and returning a copy of the Participant Consent Form. A further ten phones which were available to the project were loaned to cyclists who didn’t have Android phones themselves. They had to sign a further document stating that they would return the hardware upon request. Sixteen volunteer cyclists returned consent forms and participated in the three week trial.

4.1.3 Data Collection
4.1.3.1 Collection Period
As stated previously the user trial lasted for three weeks. During this time volunteer cyclists were using the Rothaím application to input the reason for their journeys and the perceptions of their routes. They were then recording the path that they actually cycled. Depending on the presence of a data connection some users registered accounts and uploaded their data to the server. Whereas those with no available internet connection, which is predominantly the users borrowing phones, had to connection to WiFi to upload their trips or returned the device to the project investigator with the recorded trips.
During the user trial and data collection period there were several hundred trips recorded. Visualisations of these trips are shown in Figures 4.1-4.3. However this data was not the only input from the users. The larger part of feedback broadly fell under the categories of crash reports, functionality questions, usability suggestions and new feature requests.

Unfortunately application crashes led to the irrecoverable loss of trips that were recorded early on by some users. The challenge of a phone taking a long time to receive its first location, which was discussed in the last chapter, manifested itself frequently in cyclists with the borrowed phones, as they had no data plan for Assisted GPS. We also mentioned in the last chapter how a problem was discovered regarding latitude and longitude precision.

During the data collection phase the development was primarily focused on supporting the application rather than producing major new features. At this time the Database Reader Tool was being tested against the mass of incoming trip data. It was discovered, as mentioned in the last chapter, that there was an anomaly whereby latitude and longitude precision was being lost. Once the necessary database adjustments were made it was unnecessary for the developer to record cycling information themselves as had previously been the case. The repository was promptly updated with fresh data to test against. This shortened the time elapsed between the moment the problem was identified to verifying that the implemented solution was effective.
Figure 4.1: Data Visualisation 1

Legend: The area within the red square is the image shown in Figure 4.2: Data Visualisation 2.
Legend: The area within the red square is the image shown in Figure 4.3: Data Visualisation 3.
Figure 4.3: Data Visualisation 3
4.2 Data Analysis

4.2.1 Database Reader Tool
This auxiliary tool was developed to make the process of retrieving trips stored on the server rapid, transparent and simple. The design and functionality of the tool was outlined previously in the design chapter so it will not be reiterated here.

For the purpose of data analysis this custom program was employed to download 63 unique trips which volunteers had uploaded to the database. There were many more trips recorded but unfortunately there was no opportunity to upload them due to the crashes in the early version. Even as analysis for this project was being conducted users who had continued to use the application beyond the trial period were still uploading data.

4.2.2 Data Statistics
At the time of writing 204 trips had been uploaded to the server but only 63 were available when the statistics below were created.

Figure 1 shows the breakdown of journey reasons. The bulk of the trips undertaken were for either going to work or coming home. It was expected that deviation between the number of trips to work or school and the journeys home would not be large. The trend for users is that if they remember to record their commute in the morning they are more likely to log a trip on the journey home also. The three recreational cycles were all logged at the weekend and tended to be round trips, departing from a location and returning to the same location.

The two shopping trips were also both round trips. It appears that on both occasions the user neglected to pause their trip upon arriving at the shop and so there is a window of time where no GPS locations were observed. Presumably this is because they were indoors at the time. For cycling data with no meta-data, no journey reasons or route perceptions, this deduction could not be reasonably supported. However, knowing the cyclists intention for the journey we can better understand the anomalies that arise in their trips.
There are interesting insights to be drawn from this data set about cyclist perceptions of routes. 83% of trips uploaded to the server had cited shortest as being their perception of a route. This suggests that short routes are generally to be desired by cyclists. However, if the alternative route types are considered and correlated with the journey reasons even more interesting results are revealed. Although there are few scenic trips it can be seen that none of them correspond to work journeys. Two of them are for recreational journeys and the remainder are or trips home. Similarly the safest and quietest routes were taken during trips home and to the shops.
The final graph, shown in Figure 3, to be presented takes the time at which each journey was begun and graphs the number of trips relative to the time of day. It is clear from the results that most trips take place between the hours of 8am and 10am in the morning and again between 4pm and 6pm in the evening. This is unsurprising as these are the rush hour periods in Dublin city, when most commuters are moving around. The demographic of volunteers used in for the data collection period was biased towards researchers and students due to the process of selection. This may account for the moderate dispersion in evening journeys as researchers and students are often quite late leaving for home.
4.2.3 User Feedback

It was mentioned earlier that cycling data was not the only information received from the user trial. Without the constructive user feedback that was returned during the testing period there is no doubt that the application wouldn’t have realised its current high quality.

Soon after the initial release, reports of “Force Close” crashes (critical crashes) were fed back to the developer. The root cause of the crashes was often difficult to identify because any information about the context had to come from the user’s recollection. To isolate the issue users were quizzed on their activities in the time leading up to when the crash occurred. This process eventually yielded results and led to four software updates for the Rothaím application which greatly improved the stability of the application. The changelog can be found in Appendix B.

Some users favoured the application to the extent that they replaced their other route tracking applications with Rothaím. When asked about their reasons for this they included that the liked the design of the application, the variety in statistics and the ability to check previous trips in the history. One concern that needed to be assuaged was whether or not device battery life was not being consumed at a reasonable rate. The application couldn’t be allowed to use so much power as to hinder the cyclist from using the phone for other tasks throughout the day. However, most volunteers were generally pleased with their battery usage while running the application.

Social changes were observed in the users who were actively engaging the trial. Cyclists claimed that they were monitoring their past commute times through Rothaím’s history. To
try and shorten the time spent travelling some users began to deviate from their regular routes in order to reach their destination sooner. Specifically analysing one user’s routes it can be seen that their most frequently travelled path had changed by the end of the testing period. The compared trips were for travelling to work and both routes had been perceived as being the shortest. The average commute time for the user’s first three uploaded trips is 35 minutes and 29 seconds. When the average commute time for the most recent three trips is computed the result comes in a few seconds under 30 minutes. It is rewarding to see that users are achieving their goals from their interaction with Rothaím. It also supports the perception in current research that a cyclist’s perception of a route is not always correct.

4.2.4 Quality of data

4.2.4.1 Removing noise and errors from a route

Unfortunately the recorded data has a lot of noise in it. Due to the limitations of GPS technology the only way to remove these errors is to do some post processing on the data. Neither the mobile application nor the server does any noise reduction or error correction on the data set. The data is stored as it is recorded so that researchers can work with the raw data themselves.

Nonetheless some thought was given as to how noise may be removed from the data set. Points at which there is a sudden change in gradient could be immediately removed. However, this method could potentially identify sharp corners as erroneous observations. Therefore a more sophisticated solution is required.

Recall that all data stored in the server database has an associated estimated accuracy. Using this information, observations with a large probability of error could be immediately discarded from the data set. The intervening route could be generated by using linear interpolation between the observations which come before and after the discarded point. To make matters worse, usually when one observation has a high error it is followed by a burst of observations with similarly large estimated inaccuracies. This is unavoidable and is down to the physical environment of the cyclist and the position of the satellites at that time.

Figure 3 shows how the above solution would work. The black dots mark the latitude and longitude in a xy plane, the bounding circles represent the region where the true location could actually be and the green line is the recorded route. The location recorded at P4 has a high estimated inaccuracy as shown by the red circle. It is discarded and the route between P3 and P5 is linearly interpolated resulting in a route with reduced noise. Combining both of the above solutions will, in most cases, preserve data integrity better than one on its own.
4.2.4.2 Quantify the accuracy of recorded data

It would be useful to be able to determine how close to the recorded route is to the actual route cycled. Short of actually observing the user throughout their journey it would prove difficult. What can be done, however, is to compare the performance of Rothaím against similar route tracking applications. This would require the cyclist to run multiple route tracking programs during their journey. Since the Rothaím application, and all other tracking applications which adhere to Android’s best practices, receive their location fixes from the operating system no matter which applications are compared they will be gathering the same information. This is not to say that the on-screen visualisation will not vary. An application can choose to do processing on the data before it presents it on the screen. Depending on the complexity of the processing a negative effect could be had on power consumption. Unfortunately, due to time constraints, it was not possible to run comparative tests to investigate further.
4.3 Results and Evaluation

4.3.1 Testing: Internal development vs. BETA release
There is a marked difference between closed developer testing of software and a user trial. It is not possible for an individual, particularly if they are also the developer, to effectively test the application on their own. The individual will follow a similar pattern each time. Therefore having a period where a small group of users independently test the application is necessary to detect the maximum number of problems. Rothaím’s user trial of 16 volunteers was a great opportunity to solve bugs as the users were doing the work of detecting them and the developer only had to fix them. Each individual’s unique behaviour and the different environments to which they exposed application led to a discovery rate of bugs and crashes that was far more rapid than any other stage in the development process.

The most critical crash was one which caused the application to stop working every time it was launched. The software design had naively assumed that the application would always be properly closed, with system services being unbound and persistent flags reset. In practice this is not always the case. Operating the application in new settings led to untested sequences of events. For example, if the phone was recording but lost battery power mid–journey then the application would be closed immediately without the being cleanly shutdown. Upon restart the application would believe that it is still recording (because of dirty flags) and transition straight into the recording screen. This would result in uninitialized memory being accessed and a crash occurring every time on launch.

Incidentally this bug was patched out in version 1.1.0. The solution was to add assertions on key parameters shortly after launching the application. If any of them were null the application could surmise that it was incorrectly shutdown and begin repairing itself.

It would have been difficult to identify, let alone fix, this bug and many like it if the testing of the application had relied solely on one individual.

4.3.2 Evaluation of software characteristics
This section evaluates whether or not the software implementation achieved the requirements outlined in the design chapter.

4.3.2.1 Robustness
Robustness is a measure of how tolerant the system is to faults. Leading on from the above discussion it is fair to say that the early versions of the application were not sufficiently robust. However, issues have been removed by subsequent software updates and at the time of writing there are no known critical bugs which cause the application to close. As a result the robustness of the application software has been significantly improved.

Robustness can also be applied to the availability of a service. This viewpoint can be adopted to evaluate how well the server meets the remit. This project’s design and implementation attempted to provide the maximum uptime of the server. The finite thread
pool available for requests promises that if a connection manages to be assigned a threaded servlet that it will have sufficient resources to run. If the number of allowed concurrent threads was unbounded then the availability of shared system resources would be tougher to guarantee.

The robustness objectives of the project were largely met. To further improve this characteristic future user bug reports should continue to be patched and the server should be monitored to identify unexpected behaviour that may arise during periods of high load.

4.3.2.2 Extensibility
The mobile application and the server software are both extremely extensible. The projects rich utility classes are used for logging, I/O, managing shared memory, converting data formats, parsing messages and more. The utility classes have the potential to provide these valuable resources and functions to any other Java program. Having access to these functions immediately shortens the development time of future software. The code written for both the Android application and the server has been comprehensively documented. A JavaDoc has been generated and it is a good companion to anyone using the code.

The development of the Database Reader Tool elegantly demonstrates this property of extensibility. The networking components of the Database Reader Tool were lifted directly from the Rothaím application. The NetworkController immediately exposed networking functions to the tool and only a minimal amount of configuration needed to be done to get it to work.

When the server was first designed, unused hooks were embedded in the software to leave it accessible to plug-ins that would come later. When it came to integrating the Database Reader Tool with the server it was found that the prerequisite work had already been completed. Indeed some of the functionality of the server may never be used. These capabilities are not frills; they lend the software this ability to adapt to changing needs and prolong its useful lifetime.

4.3.2.3 Scalability
The scalability issues with recording routes with thousands of location observations (namely buffering to write to the database all at once and slow drawing of routes) were solved. It can be confidently said that the application itself now scales well to trips of all durations.

The server is another matter. As mentioned in the last chapter, the InnoDB engine is used to power the server database tables. This technology automatically provides row locking. This is important because the requests to the server are assigned concurrent threads by Tomcat. Since the locks are taken and released at a database level there is no need for complex inter-servlet communication. If the system is scaled to allow other servers to share the same database then no further work needs to be done to ensure data integrity.
However, if multiple servers are added to provide the service, each with their down backend database, then the problem becomes significantly more complex. Such a system is a distributed system and an architecture would have to be designed to allow clients to locate the data and servers to synchronise database locking. Discussing such an architecture, however, is outside the scope of this thesis. In short, the current system is suitable for the purpose of collecting data from a few hundred cyclists in Dublin. However, if the application were to be released internationally, further work would almost certainly have to be done on the backend to ensure the availability of the service.

4.3.2.4 Usability
The best way of quantifying usability is to refer back to UI feedback from the user trial. The fact that some users were replacing their favourite route tracking apps with Rothaím speaks for itself. One suggestion that was taken on board was to tailor the menu of each screen to display only the options relevant to the current context. This implemented and the results can be seen in Figure 8 and Figure 9 below.

Doubtless continued iterations of user-developer interaction could improve the usability of the interface even further. To make the feedback, and indeed the bug reporting, process simpler a feedback form could be added as an additional feature of the application.
Chapter 5: Conclusion

This thesis not only proposes, but implements, a solution to a real-world transport problem. An Android mobile app was created to capture the behaviour, motivation and perceptions of cyclists in the Irish road network. Infrastructure was developed to provide further application functionality and to enable the expedient sharing of cycle data with researchers. An intuitive tool was developed to grant researchers transparent access to a growing repository of data. The thesis considered the responsibilities of a developer whose software is designed for public release.

The ecosystem of software applications and services that have come about from this project mark a significant step towards the goal of making Ireland a more cycling friendly country. The project motivation outlined the need for more accurate data to assist with infrastructural planning. This project achieved its goal in this regards and supplies transport researchers with a growing store of available cycling data.

The second project objective was to create an application which would enrich the experience of cycling in Dublin. Rothaím has received an overwhelmingly positive reception from the cyclists who took part in the user trial and from the Lord Mayor of Dublin who was shown the application in January 2012. This response when combined with the user centric design decisions, ethical considerations and a completed user trial give Rothaím a very strong mandate for public release.

The original project title proposes the creation of a personalised cycling route planner for Dublin. In hindsight this objective was excessively ambitious. The barrier to creating a model which suggests an optimal route that adapts to the behaviour of a cyclist was the same one as is faced by transport planners and researchers. That is that prior to this thesis there has been a deficiency of cycling data for calibration, training and testing any such models. Some thought was given to how an adaptive cycling route planner might be implemented. Route suggestions for users with no previous cycling history would have to initially draw from a common base for their routes. The preliminary model could be created by correlating cyclist route perceptions with their routes. In this way a model of the road network can be created where each road segment would have associated variables to indicate the degree to which they are perceived as safe or tranquil and so on. A weighted algorithm would need to be developed to extract a route of a specified type for a journey between two locations. To add the personalised element the algorithm would be modified to allow a user’s documented behaviour to influence the weightings. The route returned to a user would adapt over time as the model learned more about their cycling trends.
There is huge scope for future work after this project. The application could be released on Google Play (formerly the Android Marketplace). Besides the applications benefit to the cyclist this action would increase the quantity and variety of data collected. Integrating with social media sites is another area that would benefit the application. For example, allowing cyclists to publish their journeys could encourage their friends to get on their bikes as well as simultaneously increasing Rothaím’s renown. The server handles used for the Database Reader Tool could be applied to build a website where cyclists could visualise and interact with the trips that they have uploaded.

As seen in the first chapter, similar studies have always tended to lean towards improving policy making. There is no need to stop there. Models and theories can be established from the data set to achieve everything from identifying routes where cyclists are exposed to high levels of danger to finally integrating cycling into route planners that consider multiple modes of transport.

On a personal level this was a very exciting project to work on. During the development cycle it was necessary to draw upon many of the disciplines within computer science. Identifying information of importance and designing components that would store data falls within the realm of databases. User interface design and testing were topics of usability and universal design. Transmitting information across the internet yielded greater insights to the area of networking and security. Furthermore, it was a privilege to liaise with researchers from both computer science and civil engineering. Each side provided insights from their own experience. While the civil engineers couldn’t comment on architectural or technology choices their input shaped my understanding of the project background and how it could have influence reaching beyond being just an application. Without a doubt the final product is the better of these varying perspectives.
Chapter 6: Works Cited


Chapter 7: Appendix

7.1 Appendix A

7.1.1 Participant Information Sheet

- This research is being conducted as part of an undergraduate BAI degree course in Trinity College, Dublin.

- Currently there is no personalised route planner for cyclists. This research project aims to create an Android application which learns to adapt its personalised route to reflect the user’s previous route choices.

- We will collect location data from cyclists in Dublin city so as to recognise trends and behavioural patterns in their movement. The application will improve the quality of the personalised route the more a user interacts with it.

- You will interact with this study by recording your cycling trips and uploading them to be analysed by the researchers.

- Declaration of conflict of interest: The study organisers might take advantage of their existing relationships (friends and colleagues) to recruit study participants, and thereby make progress in their research.

- You need to be aged 18 years or older and competent to supply consent to use this application.

- Participation is voluntary and you have the right to omit information without penalty. Choosing to limit certain information may reduce the accuracy of some features of the mobile app, such as estimating calories burned.

- Furthermore you may choose to withdraw entirely. In such a case your user account will be deleted along with your history and trip information.

- You may remain involved with the research project for as long as you are using the application or until the duration of the research project.

- You may use all of the apps features, including but not limited to recording, route planning, history, statistical information.

- If you are interested in learning more about the outcomes of the research project we can arrange to give you this information upon the project’s completion.

- Only non-personally identifiable information will be used in analysis, publication and presentation of resulting data and findings.
• No user feedback will be published or quoted without the prior written permission of the user.

• In the extremely unlikely event that illicit activity is reported during the study I will be obliged to report it to the appropriate authorities.

• In the interest of safety, trips should be started, paused, finished, or otherwise only when the user is stationary.

• Under no circumstances should you use a phone while cycling.

7.1.1.1 Privacy Policy
Information is collected from you when you upload a trip, register an account or choose to upload additional personal details.

When personalising your app you may choose to provide your gender, date of birth, height, weight. This will give you a better approximation of the calories burned. Note that only non-personally identifiable user information will be used in research and publications.

You may also choose to give a contact email address to receive emails about updates. If at any time you would like to unsubscribe from receiving future emails, we include detailed unsubscribe instructions at the bottom of each email.

We implement a variety of security measures to maintain the safety of your personal information when you enter, submit, or access your personal information.

Participation is voluntary and you have the right to withdraw from the study at any time. At this time your account will also be deleted along with your history and any trip information.

We will not sell, trade, or otherwise transfer to outside parties your personally identifiable information. We may release your information when release is appropriate to comply with the law.

By using this app, you consent to our privacy policy.
7.1.1.2 Terms and Conditions

By downloading the Application from Trinity College, Dublin (hereinafter referred to as the Research Entity), installing and using the application you agree to the following terms and conditions (the "Terms and Conditions")

The Research program contemplated by this agreement is of the type consistent with the Research Entity's status as a non-profit, educational institute and may derive benefits for the Entity and society by the advancement of science and engineering through discovery;

The Research Entity grants you the non-exclusive, non-transferable, limited right and license to install and use this Application solely and exclusively for your personal use.

You agree that you are solely responsible for (and the Research Entity has no responsibility or duty of care to you) your use of the Application (the Research Entity recommends that the application is turned on before you embark on your journey and turned off only after you have safely completed your journey) and for the consequences of any improper use.

As a condition to your downloading and using the Application, you agree not to or attempt to penetrate, modify or manipulate the Software or our service or any hardware or software thereof, held or stored on any electronic means, to invade the privacy of, obtain the identity of, or obtain any personal information or Internet protocol addresses of, any end-user that has installed the application, or to modify, erase or damage information contained on the server, or take any action whatsoever, to interfere or attempt to interfere with the proper working of the Software of our services.

You further agree not to modify the Software or separate out any of its components for use with other software, transfer the Software to another person, or to decompile, disassemble, or otherwise reverse engineer or attempt to discover any underlying proprietary information of the software or any information stored by the Research Entity by an electronic means

You may not use the Application in any manner that could damage, disable or impair the Application.
You acknowledge that by installing the Application on your mobile phone that the application may aggregate, collect retain or transmit certain data compiled by the Application, this data is compiled for research purposes only.

Neither Party to this Agreement will use the name of the other party, in any publication without the prior written approval of the other party.

The Confidential information supplied by the Volunteer to enter into this agreement will not be disclosed without the prior written consent of the Volunteer.

The Data compiled by the use of the Application will be used for Research purposes only, and the Research Entity will not disclose any confidential data uploaded by the proper use of the application save as so far in the research results of the project that may be publishable, subject to the confidentiality provisions of this agreement, may be presented in forums such as symposia or international, national or regional meetings or published in media vehicles such as books, journals, websites, theses or dissertations.

Each Party to this agreement agrees to treat the Information received from the other with the same degree of care with which it treats its own proprietary information and further agrees not to disclose such information to a third party without the prior written consent.

All rights and Title to the Intellectual Property in relation to the Application on which this research programme is based vests with the Research Entity and the Volunteer acknowledges that he/she does not in any way gain a right legal or beneficial in any Intellectual Property rights that may result from the functioning of the application.

You expressly understand and agree that the Research Entity and or its agents or affiliates or licensors are not liable to you under any theory of liability for any direct, indirect incidental, special consequential or exemplary damages that may be incurred by you through your use of the application, including any loss of data or damage to your mobile phone whether or not the Research Entity should have been aware of the possibility of such loss arising.
These terms and conditions constitute the entire agreement between you and the Research Entity relating to the Application.

7.2 Appendix B

7.2.1 Rothaím Changelog

7.2.1.1 v1.1.1
*You can now choose whether or not to use HTTPS through the Settings screen

7.2.1.2 v1.1.0
*All connections now work over a secure HTTPS connection with server authentication and data encryption

*A progress bar is shown for uploading trips

*Checking the Upload Now box after recording a trip now actually uploads it instead of causing a crash

*Fixed bug where Trips recorded on Sunday showed "Error" as the day

*Fixed a bug where the app would crash every time on startup if the app had been previously Force Closed while recording

*Number of location observations is verified at the end of each trip

*Trips that never received a GPS location will not try to save

7.2.1.3 v1.0.2
*Changed the internal storage format of timestamps so precision is not lost

*Fixed a bug where out of date session keys were not being replaced

*Fixed a bug in the History where the upload button was sometimes associated with the wrong trip

*Fixed a bug where the number of collected locations became imprecise on long trips

7.2.1.4 v1.0.1
*Inserted the correct Android Maps API key