Edinburgh Bus Tracker System for Android
BSc (Hons) Computer Science

Niall George Scott

April 1, 2010

Declaration

I, Niall Scott confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

Date:

Abstract

Many people use public transport to reach their destinations. Public transport usually works on a timetable basis, however many real-world factors can adversely affect the punctuality of public transport. Many of those people travelling on public transport would like real-time tracking of their choice of public transport so that they can plan their journeys more efficiently. In the city of Edinburgh, Scotland, the local authority (The City of Edinburgh Council) has put a system in place where members of the public can check how long it will be until their bus will reach their bus stop by placing on-street “Bus Trackers” at selected stops around the city. There is also a publicly accessible website which also gives the same information, but for all stops around the capital which are served by Lothian Buses, the main bus operator in the city. The aim of this project is to investigate bringing this real-time bus information to Android mobile devices and implementing an application which will accomplish this task.

Acknowledgements

I would like to thank the following people for helping me undertake this project:

- My dissertation supervisor, Dr Peter King at the School of Mathematical and Computer Science, Heriot-Watt University, Edinburgh, for agreeing to supervise this project and providing helpful hints and pointers, particularly in how to take this project further.
- The City of Edinburgh Council, particularly Stuart Lowrie and Gary Wilson from the Department of Transport Planning & Policy, for giving me permission to proceed with my application, being extremely helpful in correspondence and giving up their time to provide me with information which helps me develop the project.
- Gordon Christie, creator of the Bus Tracker application for the iPhone mobile device (EdinBus). His correspondence has been extremely helpful and has helped me cross many barriers in the project.
- Thomas Clulow, for providing many ideas on how to tackle project hurdles and providing feedback about the project.
- Everyone who has tested the resulting application and provided feedback. Their input has been extremely valuable.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The Bus Tracker system in the city of Edinburgh, Scotland, is a useful resource to its residents and visitors, letting them know when their intended bus will arrive at their bus stop [1]. The Bus Trackers placed at many bus stops around the city of Edinburgh is a system run by the City of Edinburgh Council which informs passengers in real-time an estimated time of when their bus is due at their bus stop. The information is delivered by screens located directly next to the bus stop. The screen displays all service numbers which stop at that particular bus stop, and for each service displays the next two departures. As well as the service, it displays the final destination for that service and an estimation of how long, in minutes, the next bus for that particular service is due at the bus stop. Other information may be displayed, such as if the bus is capable of taking wheelchair users, or on very rare occasions, important travel information can be displayed, such as information on diversions. The bus relays its location information to a central computer system which then works out how long it will take the bus to reach each stop on its route and the computer system outputs this information to all Bus Tracker enabled stops on the route. A website is also available at http://www.mybustracker.co.uk where the public can check live bus departure information for all bus stops within Edinburgh [2].

Live departure information for public transport is by no means a new invention. It is something which is common inside airports, train stations, tube stations, tram systems and on bus systems in other parts of the world. A system very similar to Edinburgh's system exists in Brighton & Hove, England, which has on street bus tracking boards as well as a website where the same information can be looked up by a user [3]. For both systems, third party developers have created mobile phone applications, primarily for the iPhone platform. For rail, National Rail host a website and have created an iPhone application which lets users view live departure information for trains [4][5].

Android is a free and open software platform developed by the Open Handset Alliance (OHA) [6]. The OHA is a group of sixty-five technology and mobile companies who work together to develop the Android system. It is perhaps fair to say that Google is the single biggest influence of the OHA. Android is designed for current and future 'smartphones' and is a derivation of the GNU/Linux open source operating system [7]. A smartphone is a mobile device capabale of the traditional features of a mobile phone, such as phone calls and Short Message Service (SMS) messaging, but has further functionality such as the ability to send and receive e-mail, browse the internet in a feature-rich web browser, have inbuilt mapping software which works with a Global Positioning System (GPS) signal for position and location features and extendible with applications developed by third-parties around the world. Many mobile manufacturers, such as Motorola, HTC and Samsung have signed up to the OHA and have released mobile devices running the Android platform. Although Android is only installed on a handful of handsets at the moment, there is strong evidence to suggest that there is a good future ahead for Android if current trends continue [8]. Many handset manufacturers are due to release handsets with the Android platform, further pointing to the platform's popularity.

1.2 Aims

The aim of this project is to combine the Bus Tracker system and the Android platform in to a system where a user can view live bus stop information on their Android device. This will consist of an easy to use application for the Android device and any supporting software which may be required. The system will also aid the user to get the bus stop information as fast as they can by making the system as easy to use as possible, making the system efficient and by providing supporting features to help accomplish this. The Android platform was chosen because as of yet, no other system exists which brings the Edinburgh Bus Tracker system to the Android platform and it is felt a system like this will benefit many people within Edinburgh and its surrounding areas with Android devices.

1.3 Objectives

The project has several main objectives.

- To investigate the Android development environment.
- To investigate the Android user environment.
- To investigate the Bus Tracker system.
- To investigate ways of assisting the user to find their bus stop within the system, which may include a maps and location service.
- To develop the final system.

1.4 Problems to be solved

A variety of problems need to be solved for this project to be successful.

1.4.1 Guaranteeing the application will always work with the data supplied to it

Currently there is no Application Programming Interface (API) for the Bus Tracker service supplied by the City of Edinburgh Council for the bus tracker system. This means that the only current way of retrieving bus tracking data is via the bus tracker Hyper Text Markup Language (HTML) website or the Wireless Application Protocol (WAP) website. The format of both of these websites is never guaranteed to be the same from one day to the next as the parties responsible for the running of these websites may decide to make design and code changes resulting in the mobile application no longer being able to parse the data provided to it. The system will need to be designed around this uncertainty to ensure the application will always work with the data supplied to it.

1.4.2 Assisting the user to find their bus stop using maps and location aware services

The user may be a stranger to the city or simply may want to find their stop in a prompt fashion. By having a map of the city built in to the application which shows all the stops in the city overlaid on top of the map and can be be scrolled and zoomed upon, the user can quickly find their bus stop in a matter of seconds rather than having to find the bus stop code for their particular stop. Further to this, many Android devices can find the device's current location using GPS signals or mobile phone mast triangulation. Using this technology, the user can find their nearest bus stop(s). To display the real-time departures for this bus stop, the user will only need to simply click on the stop within the map and the real-time departures for that stop will be displayed.

1.4.3 Assisting users by making the application quick to navigate around and time saving features

When the user is wanting to find out the bus tracker information, they are often pushed for time. Ease of use in the application is essential to the application's success as this will greatly help the user. This is particularly challenging in a mobile environment where limited screen space is a problem. The user may not want hassled by overcoming multiple hurdles to get the information they desire, therefore finding information must be done in the least number of steps possible and aids should be included to help the user achieve this.

1.4.4 The application must be of acceptable performance

The application will be running on mobile device hardware. In a mobile environment resources are limited. The application must be able to complete tasks efficiently as to not use too much processing and memory resources. Persistent storage is not of concern as the application should not have to store much more than a few kilobytes (KB) persistently. More often than not, the end user will use this application while connected to a mobile network data connection rather than a wireless Local Area Network (LAN) connection. Many mobile networks limit their users to a monthly bandwidth usage allowance and often mobile data links are slow therefore it is in the user's interest that the application consumes as little bandwidth as possible. It is also in the interest of the City of Edinburgh Council, who run the web server where the data will be retrieved from, that the application uses as little bandwidth as possible.

1.5 Dissertation outline

Chapter 1: Introduction. The aim of this chapter is to give a brief introduction to the Bus Tracker system, the Android platform, state the aims and objectives of the project and a discussion of the problems to be solved within the project.

Chapter 2: Background. This chapter gives the background information and research which makes this project possible and what strategies were considered to tackle the problems to be solved in this project.

Chapter 3: Design and pre-implementation. This chapter states the software requirements of the system associated with this project as well as finding inspiration from similar systems for the system design and a discussion on a variety of risks which may affect this project. This chapter ends with some thoughts to the design of the final system.

Chapter 4: Implementation. This chapter discusses in depth the server component and client application in the final system.

Chapter 5: Testing. This chapter discusses how the system was tested to make sure it is robust and produces correct results. Also mentioned is how the application was tested by users.

Chapter 6: Discussion. This chapter wraps up the dissertation by evaluating the project, mentioning related work and possible further work that could extend the system. The dissertation is also concluded here.

Chapter 2

Background

2.1 Understanding the Bus Tracker system

As mentioned in the introduction section, there is no API available for the Edinburgh Bus Tracker system for developers to integrate their applications with. Further to this, there is no technical documentation available to assist developers, therefore understanding the system will need to be conducted through the process of 'reverse engineering'. The relevant department at the City of Edinburgh Council has been contacted to seek permission to proceed with this project as the final system will make use of their resources and they have ownership of the data, and they have given their full backing as well as offering to assist where they can. The City of Edinburgh Council were asked if they had any documentation which would be relevant and helpful to the development of this project, and they responded by saying that they had a document created by their system provider which is a specification document of the Bus Tracker website. As this document is considered commercially sensitive, it is not in the public domain and the document could only be viewed on an invite only basis. Upon viewing the document, it provided little more information than what was already known about the Bus Tracker system, as the main focus of the document was to outline how the Bus Tracker website[2] would be laid out and displayed to the user. The document did not go in to the technical details required for this project [9].

During the meeting with the City of Edinburgh council, greater insight in to how the system works was achieved. At the present time, the Bus Tracker system only works with Lothian Buses, the largest public transport operator within the city of Edinburgh. Other transport companies were approached by the City of Edinburgh Council at the time of the implementation of the system, however Lothian Buses were the only operator prepared to make the investment required to make their vehicles and systems compatible with the Bus Tracker system. The system works by almost all vehicles in Lothian Buses' fleet having GPS receivers and radio equipment on board. The bus is able to determine where it is using the GPS signal, and is able to tell when its doors are open (when the bus is at a bus stop) for further precision, and relays this information to a radio receiver. There are several radio receivers around the city of Edinburgh, most notably one placed on Edinburgh's most famous landmark, Edinburgh Castle. The radio receivers relay the bus data to Lothian Buses headquarters where the controllers can view information for all buses in service. The location data is then fed in to the Bus Tracker system, which calculates the distance between the bus and every stop on its route and gives a time estimation for every stop based on the distance from the stop. This is done for every bus in service. The time estimations are then fed by radio signals to their respective bus stops, and to the Bus Tracker website. If a bus is not fitted with the required equipment, or this equipment is faulty, the Bus Tracker system will report an estimated time based on the route timetable rather than live information.

As part of the technical research, the inner workings of the Bus Tracker website were dissected to understand how the website worked. The website uses HTML and Asynchronous JavaScript and XML (AJAX) technologies to bring the information to the user. Two separate ways of getting the live bus stop data was found;

- Get the live bus stop data from the Wireless Markup Language (WML) pages from the WAP website. WML is essentially a slightly modified derivative of Extensible Markup Language (XML) [10] and there are many freely available libraries available to parse XML documents. Regular expressions are also an option as they may prove to be faster than parsing a whole XML document when only a small section of a XML document is required. Choosing to parse the live bus stop data from the WAP website brings about the limitation of only being able to view the next two departures for the specified bus stop.

- Get the live bus stop data from the Hypertext Transfer Protocol (HTTP) website. The website will return an XML document with the live bus stop data for the specified bus stop. This document has the same information contained within it that the WML pages do, however it can contain more than two departures for each service.

The information returned by both methods includes the stop code, the stop name, the service numbers and their respective live bus times, their destination and whether the bus can carry wheelchairs or not.

An individual has already developed a mobile application for the Bus Tracker system which uses the Apple iPhone platform [11][12]. The developer was contacted to seek information on how to overcome technical hurdles, such as how to retrieve the live bus data from the website [13]. The individual responded with information on how to retrieve the live bus stop data from the Bus Tracker website by describing how to access the relevant XML files on the Bus Tracker web server and also how to retrieve the GPS coordinate for a bus stop via other XML files.

If the application is to make use of location aware services then it will be necessary to at least use parts of the HTTP website as the WAP website does not have support for bus stop locations. The stop locations are not found in a single XML document, instead they are listed in route XML files, therefore to find the location of every stop serviced by all of the bus services, it will be necessary to retrieve the relevant XML documents for every single service and process them. The stop location data documents includes the stop code, the stop name and the stop GPS coordinates (given as separate X and Y coordinates).

2.2 Understanding the Android platform

The Android platform allows developers to create application which extend the functionality of the device. Applications come in many shapes and sizes. Examples include unit converters, social networking clients and games. The Android platform provides an extensive fully documented API [14] which allows developers to rapidly develop applications for the platform as much of the code is already written for them. The API also ensures commonality between applications and future compatibility. A Software Development Kit (SDK) is also available for the Android platform which includes tools such as debuggers and a platform emulator which emulates a developer's application running on a handset [15]. The emulator can be configured for a variety of different phone features and different screen sizes and types. The SDK tools can also install a developer's application on his handset and the ability to debug the application from the handset. Other SDK tools include project packaging utilities and project creation utilities.

The programming language of choice on the Android platform is Java, therefore any applications developed for it must be written in Java. The Android API includes most of the Java Standard Edition API, as well as the Android libraries, HTTP API implementations provided by the Apache foundation, JavaScript Object Notation (JSON) API and XML API. No part of the Android API is hidden from developers, unlike other platforms such as iPhone, therefore developers have access to the same platform APIs that the system applications do. This can easily be verified as the Android platform is open source. As an extension to the Android API, Google have created a Maps API which allows application developers to include Google Maps in their projects and manipulate the maps to suit the needs of their application. The Android developers have created a detailed developer guide to get developers started with the basics of the platform, and to work with more advanced features of the API, such as the Maps API [16]. The developer guide and API documentation are separate, however they do link to each other. The API documentation is generated by Javadoc from the library source code.

Android application types and life cycles need to be understood as these are fundamental concepts of the Android platform. An application can have different states throughout its life cycle such as "active”, “paused”, “resumed”, “stopped” [17]. It is important to understand when these states may come about as the application is likely to be undertaking different tasks for different states. An example would be when the application is in a “paused” state, the application performs a clean-up operation to free resources to other applications, when in an “active” state the application is in the foreground therefore interacting with the user. Android has an activity stack, with the activity in focus at the top of stack. When an activity ends, it is pushed from the top of the stack, similarly, new activities are pushed to the top of the stack. When the device requires system resources, old activities are ended to make resources for the activity in focus if the old activity is still running in the background. Also of importance is the Android Intent system of inter-process communication, or in an Android context, inter-activity communication. These are 'bundles' of data which pass information between activities and are used to start new activities. The Android platform implements its own Graphical User Interface (GUI) components based on a 'widget' system and is loosely based on the Swing/AWT model.

The Android file system follows a UNIX like structure, with default permissions for an application that only it can access its data and no other application can as each application is a “user” on the device [18]. The data for an application is also contained within its own data folder, meaning that when the application is uninstalled its data is deleted too. As well as having the ability to store flat text files on the device's file system, the application can also create structured XML files via the preferences API to store application preferences, and SQLite databases can be stored on the device and accessed via the Android API. The Android system is fully compatible with Java sockets and can connect via a wireless LAN connection or via a mobile network data connection, however the appropriate permissions need to be sought from the Android system before an application can make network connections.

Much of an application can be defined within XML documents. The application must have an AndroidManifest.xml file in its root directory which defines the application parameters, such as name, icon and a list of activities contained within it. It is recommended within the Android developer guide that the GUI is defined within XML documents. The XML document will define what GUI elements are to be placed within an activity, parameters for the supplied GUI elements and their hierarchical structure to each other and relative positions. It is also recommended in the developer guide that strings are defined in an XML document for locality reasons, as the platform can choose the correct strings definition file depending on the locale set on the device.

It is critical that the Android platform is understood for the project to be successful. The application must conform to Android must-dos and shouldn't-dos to ensure efficiency and so that the user does not become frustrated with the application. The application must be as efficient as possible with resources as to not drain the battery, as extended processor or networking use on a mobile device will drain the battery more quickly.

2.3 Understanding Google Maps and Google Maps for Android

The Android platform has support for Google Maps [19]. It would be beneficial for this project if it used Google Maps to show bus stop locations to the user to help them quickly find their bus stop.

The original usage of Google Maps was on desktop and laptop computers and accessible from a web browser at http://maps.google.com and allows the user to search for locations, find directions between two points and to find nearby businesses to a location, all shown on a map which can be scrolled and zoomed [20]. More recently, Google Maps can now show user content of a location such as pictures and Wikipedia articles and show Google 'Street View” images, where street level 360 degrees imagery has been taken of streets in selected locations.

Since its introduction, Google Maps can be found on many mobile platforms such as Nokia smartphones, Blackberry, iPhone, and most importantly for this project, Android. Google play a key role in the Android platform therefore it is in their best interest to include Google Maps for the platform. Google Maps for Android provides an API to allow developers to include maps within their applications [21]. Like the full web version, it allows developers to place 'layers' on top of the map to include location information and to include 'markers', points placed on the map in specific locations which show points of interest, such as restaurants, banks, petrol stations etc. In the context of this project, the points of interest will be the bus stops and they will be placed as markers on the map [23]. For the user to find out more information about a bus stop, namely the live departures, they simply need to press on the marker. This adds to the ease of use of the application as the alternative of displaying live departure information is to manually enter the stop code in to the application. The stop codes, or known within the City of Edinburgh Council as “SMS Codes”, can be found at every bus stop in Edinburgh and the surrounding areas.

To use the Google Maps API, application developers are required to sign up for a Google Maps API Key. This key is linked to the “signature” of the application, as all Android applications need to be digitally signed before they can be executed on an Android device. Google require this to know who is using their Google Maps service. After the Maps API Key is obtained, the application developer can include a Google Maps object inside their application and manipulate it how they please within the limitations of the provided API. The developer can enable map, as well as satellite views of the maps. Although the Google Maps application on Android includes a Street View feature, the Maps API for Android does not expose this therefore applications are not able to take advantage of the Street View features. This would be useful as it would give another perspective to users trying to find their bus stops, particularly in a crowded and/or unfamiliar area. Google have full developer documentation of the Maps API for Android via generated Javadoc.

Google Maps was chosen as the technology to look further in to for mapping capabilities on the device as currently there is no competing API available for the Android platform. Google Maps, for the mean time, is the only choice for adding mapping capabilities to an application.

2.4 Understanding location aware services within the Android platform

Location aware services in terms of mobile devices is the ability for a mobile application to determine where in the world the handset is and provide a service to the user based upon this [22]. In terms of this project, its location aware service would be to find the handset's current location and show the nearest bus stops to this location on a Google Map. The handset can find its location in one or the combination of three ways;

- Mobile network triangulation. This is when the handset has a signal to a mobile phone network and attempts to best guess its location based on its connection to mobile phone masts in the area. The results are often imprecise as it is a best guess and can often show a handset being a few streets away from its actual location.
- GPS. This is when the handset receives signals from a network of satellites orbiting the Earth and is able to work out its location based on how long it takes to receive messages from satellites. This method is much more precise than mobile network triangulation and often shows a location correct to a tolerance of a few inches. GPS does carry a few disadvantages however; the device may not be GPS capable, the user may not know how to enable GPS functionality on their device, the user may not want to use GPS functionality on their device as it drains battery power far quicker and GPS can only be received when the handset is in clear view of the sky above.
- Wireless LANs. When the device is connected to a wireless LAN the device is able to query an online database which gives location data at address precision for where the device is located. This is used with a combination of the wireless network details and the public IP address of device through the network's gateway.

In the Android platform, the API allows for the application to not worry how a location is determined as the API abstracts this away from the developer. The developer only needs to find the handset's co-ordinates from the Java classes contained within the android.location package in the Android API. In the Google Maps for Android API, this is made even easier by the MyLocationOverlay class. This class can be overlaid on the Map to show the current location of the device by placing a marker on the map. The MyLocationOverlay class handles the task of determining the device location from the location API therefore making it easier for the developer. It is considered good practice to turn off the location services when they are not being used as they consume more battery power due to the use of the processor and radio chips in the device.

2.5 Understanding methods of the distribution of structured data

Data received from the Bus Tracker website is of such complexity that it should be structured. Most of the data coming from the Bus Tracker website is either in HTML or XML [10]. Both of these formats are highly human readable however they are difficult to parse, especially in an efficient fashion. Many freely available libraries exist for many different programming languages which shorten development time and are optimised. Another disadvantage of these formats is that they are very bandwidth consuming due to carrying extra 'baggage' to make them highly human readable. In the context of the Android application it is not necessary for the incoming data to be human readable as it is still to be processed and will be output in a human readable format regardless. Such a format that solves this problem is JSON [24]. The data still arrives in a structured way but because it does not have a tag system such as HTML and XML do, then bandwidth which was otherwise used up by these formats is conserved [25]. This is important when the data is coming over a mobile network data connection. XML is strictly a document exchange structure and JSON a data exchange format. JSON should be readable to anyone who has an understanding of programming language syntax.

To compare the bandwidth saving between the two formats a simple experiment was created. A sample XML document was retrieved from a website and its equivalent in JSON was also retreieved [26]. The number of characters used by each format to represent the data (which is equivalent to number of bytes) was compared, ignoring any white space characters to format the data such as new line, tab and spaces.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber type="home">212 555-1234</phoneNumber>
  <phoneNumber type="fax">646 555-4567</phoneNumber>
</Person>
```

Figure 2.1 An example of a XML document.

```json
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber": [
         { "type": "home", "number": "212 555-1234" },
         { "type": "fax", "number": "646 555-4567" }
     ]
 }
```

Figure 2.2 The equivalent of the example XML document in JSON format.

The size of the XML document stands at 354 bytes where as the equivalent JSON document stands at 249 bytes. This is a difference of 105 bytes for a very short document. This difference will be greater for larger documents and can make a difference if this saving is made thousands of times. If this JSON text was to be transported over a network one thousand times times, a bandwidth saving of 105,000 bytes (approximately equal to 102.54KB). This is important on a mobile device with constrained bandwidth.

The structured data coming from the Bus Tracker website is in XML format therefore to convert the data in to JSON format for the mobile devices some sort of intermediate processing is required. This could be done by an intermediate server, providing the data to the client via a minimalistic protocol with the ability to transfer JSON documents. This method also brings about the advantage that if the XML format provided by the Bus Tracker web server changes for any reason then only a single server needs to be updated rather than possibly hundreds of Android clients, simplifying the application distribution process and preventing user frustration. The downside to this method is that it presents another point of failure in the already long chain of information passing. If the server fails then the Android clients will not be able to use the Bus Tracker service. The advantages and disadvantages of this method need to be weighed up in order to determine how to best proceed. In an Android API context, it does not matter if the device reads JSON or XML, as the Android API has support for both.

Chapter 3

Design and pre-implementation

3.1 Software requirements

The software has a variety of requirements which must be met for the system to be successful;

- The target system must be able to retrieve live bus stop departure data and present it to the user when it is supplied a unique bus stop code in the clearest possible fashion.
- The target system must allow the user to save the bus stop as a favourite bus stop so they can revisit this bus stop for live departure data at another point in the future. This allows the user to use the system in a prompt fashion and not overcome too many hurdles to get the data they want. This counts towards the aim of allowing the user to complete tasks in the fewest number of actions possible.
- The target system must be as easy as possible to use. First impressions count and this is particularly true for software. The application must be welcoming to the user and items must be laid out in sensible ways and non ambiguous captions should be used.
- The target system should provide a Google Map combined with location aware services which allows the user to select nearby bus stops to their determined location, aiding ease of use.
- The application must be processor, memory and bandwidth efficient.
- The application must be made available on the Android Market, the Android application store, free of charge and the source code produced must be open and freely available.

- The application must conform to technical standards, for example, the HTTP protocol and the guidance provided in the Android developer guide.

3.2 Project risks

There are a variety of risks associated with undertaking a software project, the specific risks for this project are outlined below.

3.2.1 Risks within the Edinburgh Bus Tracker system

The Edinburgh Bus Tracker system appears to be quite volatile due to its current lack of developer support and the ways of retrieving data from the website. Possible risks could include;

- The City of Edinburgh Council could withdraw their permission for the application to use the Bus Tracker service. This would be a devastating blow to the project as access to their system is a critical requirement for the system to work. This is unlikely as after the initial correspondence with a representative from the City of Edinburgh Council, they explicitly stated that they are keen for developers to develop for their service and are also considering introducing a developer API to make it easier for developers to use the system. The City of Edinburgh Council is also seeing great success and interest in the system.
- Until an API is made available, the system will need to parse data from the Bus Tracker website. The data is not guaranteed to always be in the same format and any changes made to the website could make the parser unable to parse the website data correctly. This is likely to happen at some point in the future, as in correspondence with the City of Edinburgh Council they stated they were planning on making some changes to the website in the future. As discussed in section 2.5, a possible solution to solve another problem, that of reducing bandwidth usage of the mobile device, could be used to solve this issue too. If the City of Edinburgh Council were to provide an API for the system which output JSON formatted documents, then the need for an intermediate server would be eliminated altogether.

- The city of Edinburgh Council could terminate the Bus Tracker service, making the system useless. The only possible contingency is that the system could parse bus timetables and guess when a bus service is to reach a bus stop. Although the short term future of the Bus Tracker services has been assured by representatives from the City of Edinburgh Council, the long term future of the Bus Tracker service has not been guaranteed.

3.2.2 Risks within the Android platform

So far it has been determined that this project is entirely possible with the API given on the Android platform. Possible risks as a result of the Android platform could be;

- The Android platform does not take off and therefore has a small target audience and/or the project is scrapped. Given the success so far for the Android platform this is unlikely.
- The Android developers drastically change the platform API, rendering all previous applications useless. This is unlikely as it would not be in the best interest of the platform to render all previous applications useless. Also, applications which are compiled for a specific API version are guaranteed to work for that API version and newer API versions. Applications compiled for a newer API version than the system it is intended to be installed on will not work.
- At the moment Google charge a small fee for developers to place their applications on the Android Market. This is a once-off fee and exists to promote creativity rather than developers filling up the Android Market with “junk”. In the future, it is possible Google could charge a yearly fee so that developers can distribute their applications on the Android Market. Also, Google may introduce a more stringent application approval process. This would be following Apple's model of an “Application Store”, where many people have expressed displeasure in the application approval process.

All in all, the Android platform appears to be a very stable platform to develop for.

3.2.3 Risks within the project planning, design, implementation, testing and report writing

The risks in this category are the sort of to be expected in a software project;

- The project could be a lot more complex than first anticipated. A best-effort will need to be produced in the event of this occurring.
- The project could be late due to bad time planning or unanticipated hurdles as the project progresses. Again, a best-effort will need to be produced within the allotted time.

3.3 System design

3.3.1 Server side and back end

It was decided the final system design would incorporate the intermediate server. The intermediate server would have the job of converting the live stop XML [10] data from the HTTP Bus Tracker Website [2] in to JSON [24] format which is then transported to the Android clients. This brings about two main advantages; bandwidth would be conserved as the JSON data is less bandwidth intensive than the equivalent XML data and this would also make deployment easier as the clients would not need to be updated every time time the XML data format changed, only the server would. It is felt that the intermediate server would be the best option for live bus data until The City of Edinburgh Council expose an API for developers. The intermediate server would also provide the database of bus stops to the clients required for the mapping functionality within the Android application. It will accomplish this by weekly downloading the list of bus stops and their locations from the Bus Tracker web server and creating a SQLite database of these stops. SQLite was the chosen database technology as SQLite exists in the Android API [14] and it also produces a single file database which can be transported over a network and imported on to another system. The database will be made available as a single file in a web accessible location for the clients to download via the HTTP protocol. The HTTP protocol was chosen as it is a protocol designed for file transferring and existing libraries can be reused on both the server and client side. On a regular basis, while the Android client is running on the user's device, it will query the server to see if a newer version of the database exists and if so, download the new version of the database and replace the version it has on the device. The server will also be able to inform the client of the latest version of the client available for updating purposes. If a new client version is available, the client will inform the user and guide them to the relevant part of the Android Market to update. Figure 3.1 shows a model of the desired data fetching method:

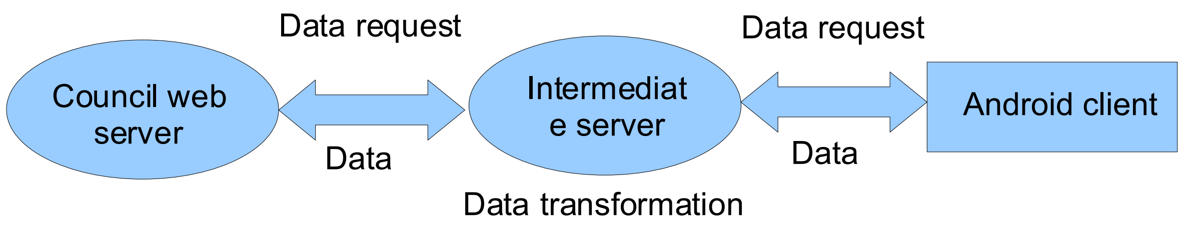


Figure 3.1 shows the desired data fetching method.

A suitable format for the JSON data was created. JSON provides a way of transporting structured data which is less bandwidth intensive than XML.

```
{
		"stopCode": "<stop code>",
		"stopName": "<stop name>",
		"services": [
				{ "serviceName": "<service name>", "route": "<service route>", "buses": [
									{ "destination": "<destination>", "arrivalTime": "<arrival time>",
"accessible": <true/false> }
						]
				}
		]
 }
```

A real world example is shown below:

```
{
		"stopCode": "36232658",
		"stopName": "Haymarket Satio",
		"services": [
				{ "serviceName": "3", "route": "Mayfield -- Clovenstone", "buses": [
									{ "destination": "CLOVENSTONE", "arrivalTime": "5",
"accessible": true },
									{ "destination": "CLOVENSTONE", "arrivalTime": "*30",
"accessible": false }
						]
```

```
                                    },
                                    { "serviceName": "25", "route": "Riccarton -- Restalrig", "buses": [
                                                                       { "destination": "RICCARTON", "arrivalTime": "*12",
"accessible": false },
                                                                       { "destination": "RICCARTON", "arrivalTime": "42", "accessible":
true }
                                                     ]
                                    },
                                    { "serviceName": "33", "route": "Westburn -- Ferniehill", "buses": [
                                                                       { "destination": "BABERTON", "arrivalTime": "22", "accessible":
true },
                                                                       { "destination": "LONGSTONE", "arrivalTime": "45",
"accessible": true }
                                                     ]
                                    }
                  ]
}
```

This example shows three bus services; 3, 25 and 33 as stopping at this bus stop and each service has two next departures. There is an external library available for the Java platform as well as a library included within the Android platform for creating and parsing JSON documents. The "JSON text" transported across the networking is transported as a single line, the document does not contain any line breaks.

The protocol between the client and server has been created to use as little bandwidth as possible. It follows the format of *command:paramter1,parameter2*. An example command for fetching live bus stop data would be *getBusTimesByStopCode:<stopCode>* such as *getBusTimesByStopCode:36232658*. An example session is shown below:

```
Client: getBusTimesByStopCode:36232658
Server: +
Server: <JSON text here>
Server: -
Client: exit
```

If the server encounters an error, such as if it cannot connect to the Bus Tracker web server, or if an incorrect stop code is supplied, it will return a string such as *Error: <error string>*.

The server also supports extra commands as previously discussed. An example is shown below:

```
Client: getLatestAndroidClientVersion
Server: 0.0.1
Client: getDBLastModTime
Server: 1269480086498
Client: getDBURL
Server: http://www.rivernile.org.uk/busstops.db
Client: exit
```

The server will have the capability of serving multiple clients simultaneously (the maximum amount can be configured in a server configuration file) and the server will be multi-threaded.

3.3.2 Android client

Similar clients and systems were investigated to get ideas for the look of the Android client. Firstly, EdinBus [12] for the iPhone platform was looked at for ideas. The EdinBus application follows a typical design that an iPhone application would follow. The Android look and feel is different, however the EdinBus application was looked at to see how a similar system to that of this project interacts with the user. Figures 3.2, 3.3 and 3.4 show the user interface of the EdinBus application.

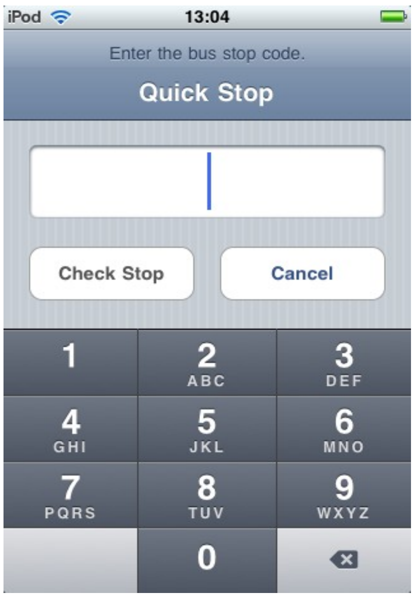


Figure 3.2

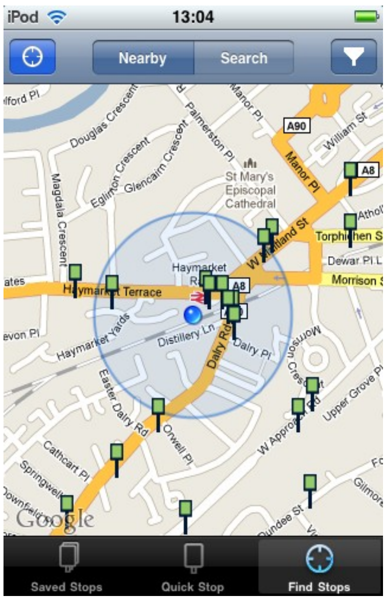


Figure 3.3

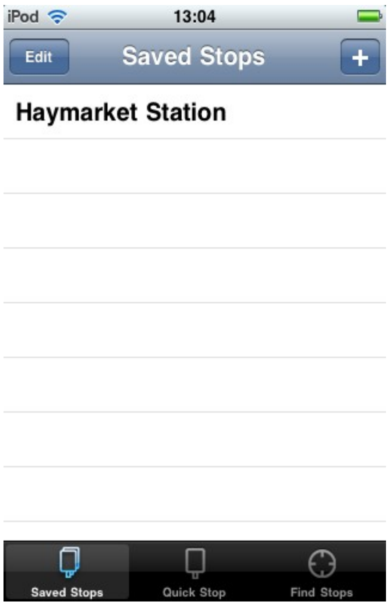


Figure 3.4

Figure 3.2 shows where the user can enter a bus stop code using the supplied number pad on the touch screen. All bus stops have a unique code attributed to them which is printed on the timetables at the bus stop. The user can enter this bus stop code instead of searching for it by other means and they will be taken straight to the live departures for this bus stop.

Figure 3.3 shows a Google Map interface on the iPhone platform. The iPhone shows the handset's current location on the map at the centre. The markers shown on the map are the bus stops and the user presses on a marker to reveal more data for that bus stop. The data shown shows the user the name of the bus stop and also a list of what services stop there. From there, the user can decide to choose another bus stop or can proceed to find out the live departures for the selected bus stop.

Figure 3.4 shows the application home screen which also hosts the list of “Saved Stops.” The bar shown at the bottom of the screen is used to navigate to other parts of the application and the user can add further favourite bus stops to this list.

Investigation was carried out in to finding out what a typical user interface for the Android platform looked like. Unlike the iPhone which only has a single navigation button (the “home” button) and the rest is done by the touch screen interface, Android devices typically have other buttons on the device for navigation. These buttons include a “home” button, a “menu” button and a “back” button. The rest is usually done via the touch screen interface. It is typical to find on the Android platform that devices have a main menu where the user can select which action to perform and by pressing on the menu item it will perform this action for the user. To get back to the main menu, the user presses the “back” button on the device until they reach the main menu. Some dialogues, or “activities” within Android, may have a menu attributed with them. This may be a menu which can be accessed via the “menu” key, or a contextual menu which can be accessed by “long pressing” on an item on the display. A long press is where the user presses and holds on an item for a few seconds and a contextual menu is displayed or another action is performed. The “home” key will exit the current application and take the user back to the device's home screen. Where text is to be input, the device displays an on-screen keyboard. The following figures show typical Android user interfaces.

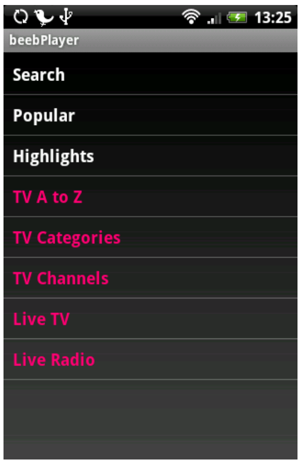


Figure 3.5

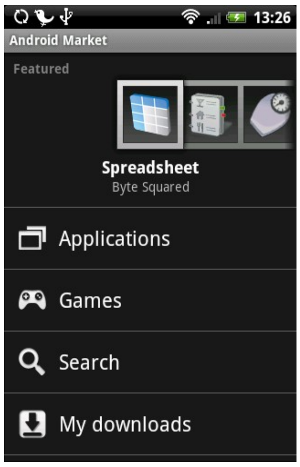


Figure 3.6

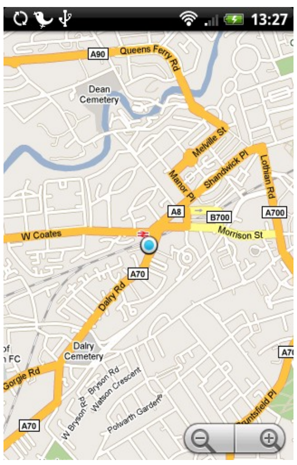


Figure 3.7

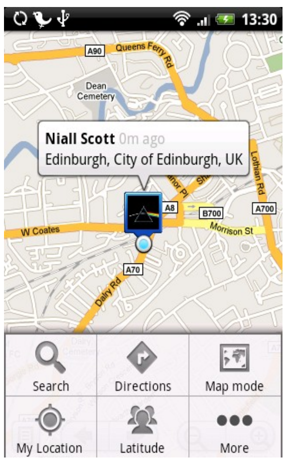


Figure 3.8

Figure 3.5 shows the main application menu being shown where users can select items to perform actions. Figure 3.6 is an extension of this, where a gallery is shown above the menu. Figure 3.7 shows a Google Map [19] interface. Figure 3.8 shows a Google Map interface with the device's current location shown and the "menu" key has been pressed to expose a menu for the current "activity". These figures show the short of direction this project will be heading in terms of user interface.

The user interface will consist of the following components;

- **MainActivity:** this is the first activity the user will interface with in the application. It will contain a menu of possible actions the user can undertake in the application. This will extend from ListActivity.
- **DisplayStopDataActivity:** this is where the live bus information will be displayed to the user. This will extend from ExpandableListActivity.
- **EnterStopCodeActivity:** this is where the user will be able to manually enter a stop code. This will extend from Activity.
- **FavouriteStopsActivity:** this is where the user will be able to access their list of saved favourite bus stops. This will extend from ListActivity.

- **AddEditFavouriteStopActivity:** this is where the user will be able to add a new saved favourite bus stop or edit a stop already saved. This will extend from Activity.
- **BusStopMapActivity:** this is where the Google Map will be displayed to the user and they are able to find their bus stop using the mapping functionality. This will extend from MapActivity in the Google Maps API.
- **PreferencesActivity:** this is where the user will be able to change a handful of settings related to the application. This will extend from PreferenceActivity.

Activity, ExpandableListActivity, ListActivity and PreferenceActivity can be found in the Android platform API and they are default activities which can be extended from.

The interfaces will interact with each other and follow on from each other in the following way:

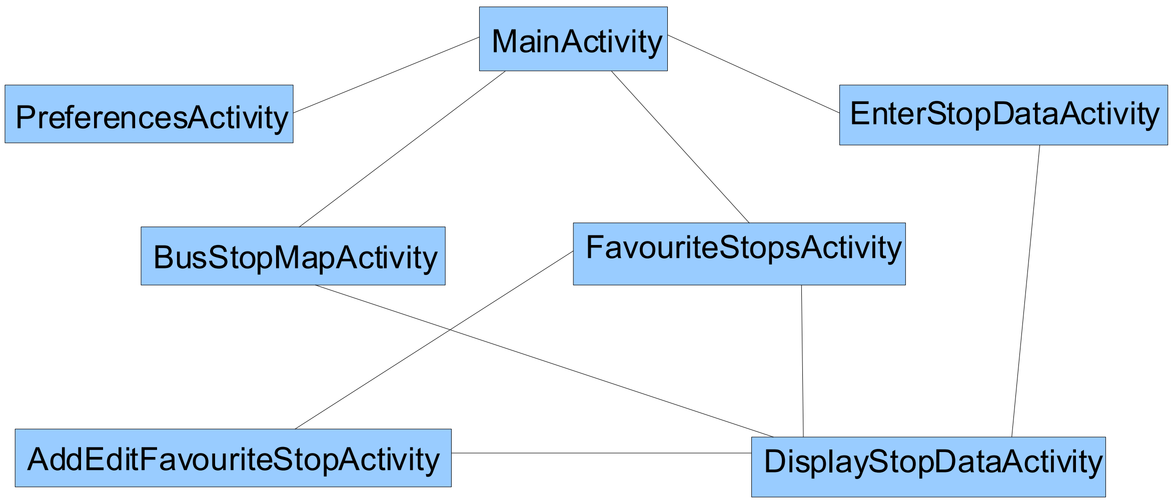


Figure 3.9 shows how the application activities will link together.

Chapter 4

Implementation

4.1 Hardware and software used

The following hardware and software were used in the creation of this project;

Android device: HTC Hero

Android API: Android API version 1.5 with Google APIs version 4.

Programming language: Java.

Integrated Development Environment: NetBeans with nbandroid plugin.

Extra libraries used: JSON, SQLite JDBC.

The HTC Hero device was chosen to be the development device as that was what was available at the time. It was decided to develop for Android API version 1.5 as that is the API version that most Android devices are running. Java was the programming language used to develop the system as Android applications have to be created in Java. NetBeans was chosen as the IDE as it provides good project management with inbuilt Subversion support.

4.2 Server component

The server component was the first section to be implemented as the client would not be able to function and thus be tested without it. The fundamentals of the server were implemented first, such as reading of the configuration file and the handling of the connection sockets. The reading and parsing of the XML [10] documents which contain the live bus stop data from the City of Edinburgh Council web servers was implemented next. Once this had been implemented then client commands could be issued to the server to retrieve the JSON [24] formatted data. As the protocol between the client and server is plain-text, then it is possible to do this through the Telnet utility. At this stage the server component was tested extensively. Development on the server component was halted whilst development on the Android client application begun.

When development on the server component resumed, the main task was to compile a database of all bus stops in Edinburgh with their name, stop code and location. The server would do this if it could not find a previously created bus stop database upon starting, if it has been 7 days since the stop database was last created or if the time is 06:00:00 on a Sunday. This is because if there are any changes to bus stops on the Bus Tracker system, it occurs at some point on a Sunday morning.

The Bus Tracker system does not provide the bus stop details in a single, easy to parse XML file, it is split in to many XML files depending on the services it is for. As many services share bus stops, then it is inevitable there will be duplicates. To accomplish this, the server downloads the stop data for the bus service '1'. It puts in to the database all of the stop data for this bus stop and creates a list of the bus stops it has found. It also creates a list of all the bus services it has found. Once it has finished with this bus service, it moves on to the next bus service in the list and repeats the process until the list of bus services has been exhausted. If a bus stop already exists in the list, it is simply ignored and the next bus stop is then looked at. A mapping between stop codes and bus services is made to determine what bus services stop at each bus stop. When the server compiled the database, the server found in excess of 2,500 bus stops and made around 8,000 stops to services mappings. On a fairly modern desktop computer, it takes around 3 minutes to create the bus stops database. Once the database has been created, it is put in to a web accessible location and the database can be downloaded by the client via the HTTP protocol.

```
CREATE TABLE bus_stops (_id TEXT PRIMARY KEY, stopName TEXT, x INTEGER, y INTEGER)
CREATE TABLE service_stops (_id INTEGER PRIMARY KEY AUTOINCREMENT, stopCode TEXT, serviceName
TEXT)
```

Figure 4.1 shows the SQL schema used in the bus stop database.

To aid in responsiveness, the server is multi-threaded. The main thread of the application deals with the starting up of the server then sits and waits for incoming connections. When a new client connection has been created, a new thread is created which deals with that particular connection. When the client disconnects, that thread exits. Whenever the bus stop database is being created, that is done in its own thread as it is a resource intensive task.

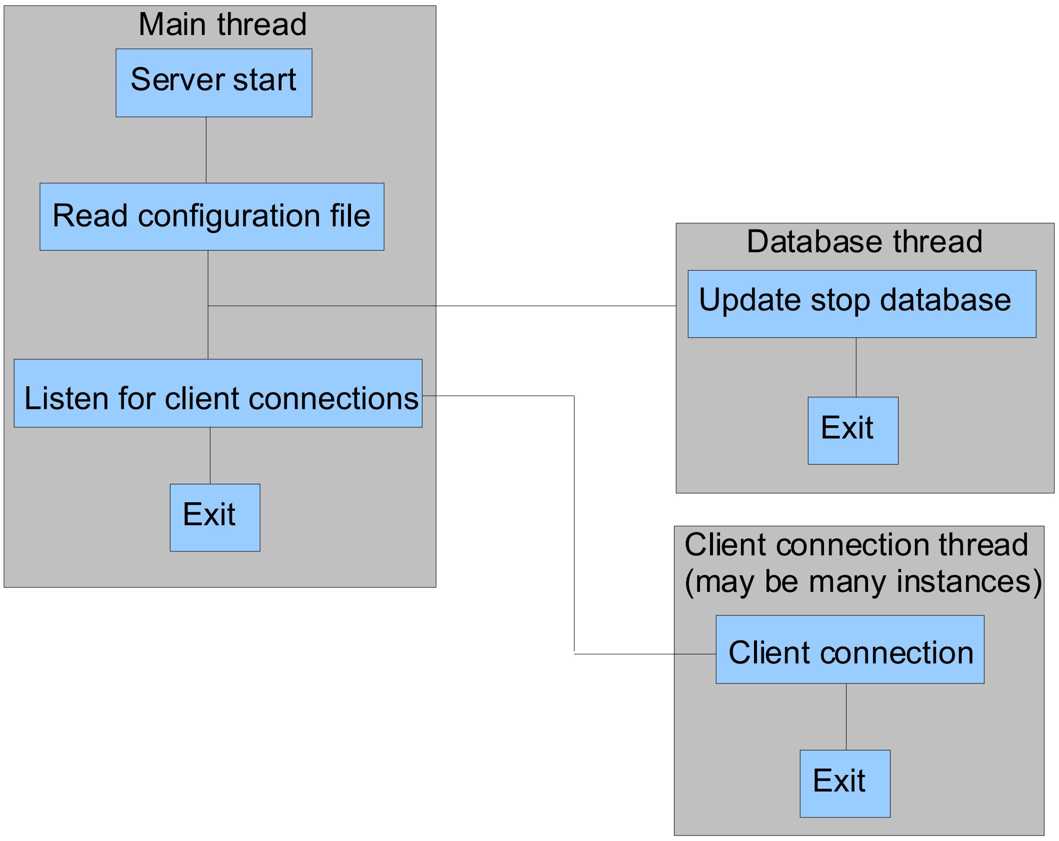


Figure 4.2 shows a simplification of the multi-threaded nature of the server component.

4.3 Android client

The implementation of the Android client began after the server was able to parse and output the live bus stop data. The strategy in working on the Android client was to work on and try to complete as much as possible each “activity” before proceeding to the next activity.

4.3.1 MainActivity

The MainActivity is the entry point in to the application. It displays a list of actions the user can take to navigate to another section of the application. This activity is extended from ListActivity, which is a class included in the Java API which handles a ListView component. When the user selects an item in the ListView, it takes the user to the relevant activity in the application, such as pressing on “Preferences” will take the user to the PreferencesActivity. The new activities are started by creating an “intent”, which is an object which describes which activity is to be undertaken, and then firing this intent. This activity also creates threads. The first thread will check to see if the bus stop database exists – if it doesn't then it will copy a version of the database included in the application package. The second thread will check to see if the bus stop database is up to date with the server, if it is not then it will download a new version over HTTP and replace the old version then inform the user that the database has been updated. This thread will also check to see if the client is up to date with the server. Currently no action is taken if the client is not up to date as the application is not yet in the Android Market. This functionality will be enabled once the application is ready to be submitted to the Android Market.

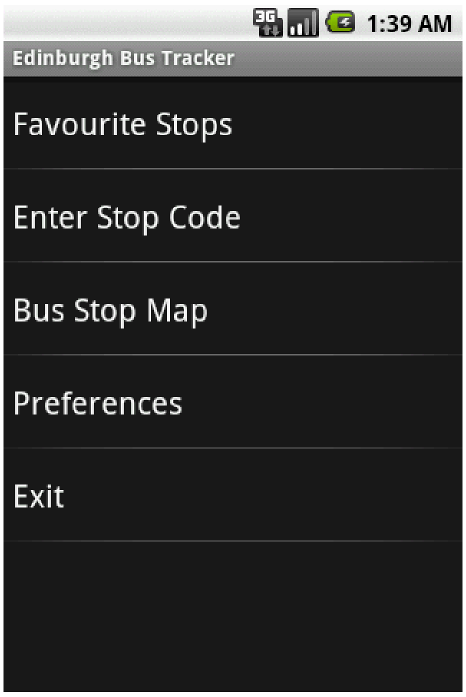
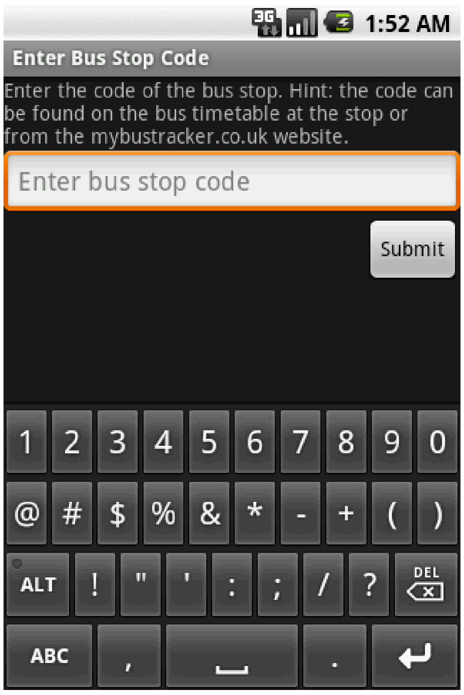


Figure 4.3 Screenshot of MainActivity.

4.3.2 EnterStopCodeActivity

The EnterStopCodeActivity allows the user to manually enter a bus stop code in to the application. This was the second part of the application to be implemented. This activity consists of a label which informs the user what to do, a text entry box which allows the user to enter the stop code and a submit button to let the user proceed. The device keyboard (whether that be a physical keyboard or an on-screen keyboard) will only let the user enter numbers in to the text entry box as bus stop codes can only be a number. An error is displayed to the user if they try to proceed without entering a code. To navigate back to the main menu of the application, the user presses the "back" button on their device.

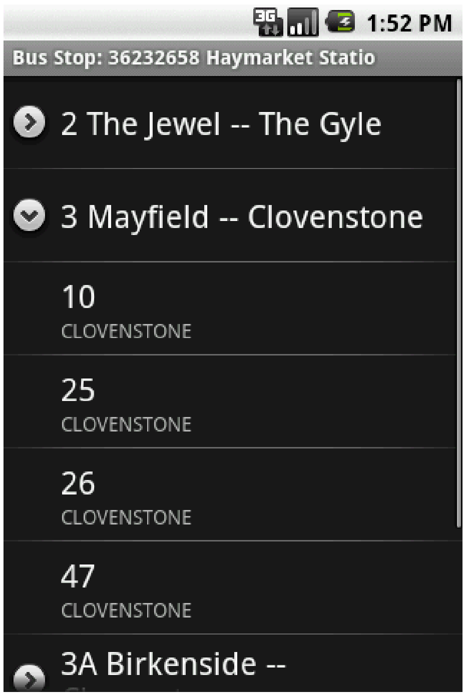


Enter the code of the bus stop. Hint: the code can be found on the bus timetable at the stop or from the mybustracker.co.uk website.

Figure 4.4 Screenshot of EnterStopCodeActivity.

4.3.3 DisplayStopDataActivity

DisplayStopDataActivity displays the live bus stop data to the user. This activity accepts “intent” data which contains the bus stop code. This activity extends ExpandableListActivity which hosts a ExpandableListView component. An expandable list is a list where the parent nodes can be expanded to reveal child nodes. In this case, the parent nodes are the bus services and the child nodes are the times for that bus service. This activity handles the connection to the server to retrieve the live bus stop data. Whilst this activity is busy retrieving the data, a progress dialog is displayed to the user. If any errors occur whilst retrieving the data, error dialogs are displayed to the user and give them the option to retry the connection or to cancel – cancelling takes the user back to the previous activity they have come from. If the data has been successfully retrieved, then the JSON text will be put through the JSON parser and then the ExpandableListView will be populated. If the user has turned the auto-refresh preference on in the application preferences or via the menu in this activity, the data for the bus stop will be refreshed every sixty seconds. This activity has three menu items; the ability to add the current bus stop as a favourite bus stop (takes the user to AddEditFavouriteStopActivity), turning auto-refresh on or off for this bus stop and to force a manual refresh for this bus stop.


1:52 PM

Bus Stop: 36232658 Haymarket Statio

2 The Jewel -- The Gyle

3 Mayfield -- Clovenstone

10
CLOVENSTONE

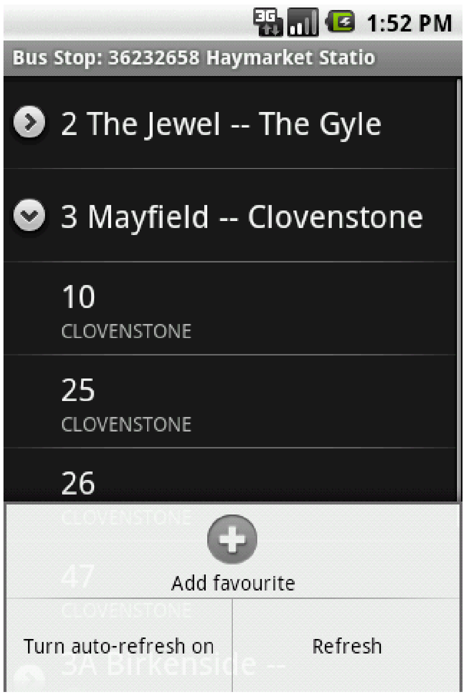
25
CLOVENSTONE

26
CLOVENSTONE

47
CLOVENSTONE

3A Birkenside --

Figure 4.5 shows the live bus data being displayed.


1:52 PM

Bus Stop: 36232658 Haymarket Statio

2 The Jewel -- The Gyle

3 Mayfield -- Clovenstone

10
CLOVENSTONE

25
CLOVENSTONE

26

Figure 4.6 shows the live bus data being displayed, with the activity menu.

4.3.4 FavouriteStopsActivity

FavouriteStopsActivity displays a list of saved favourite bus stops to the user and allows the user to select the stop to show the live data for that bus stop. This activity extends ListActivity which contains a ListView component. The list is populated when the activity is started by querying a SQLite database which contains the list of favourite bus stops. A contextual menu is also associated with this activity. It allows the user to view the live stop data for that stop (just like tapping the stop will achieve), edit the stop name they have stored and delete the stop from the list of favourite stops.

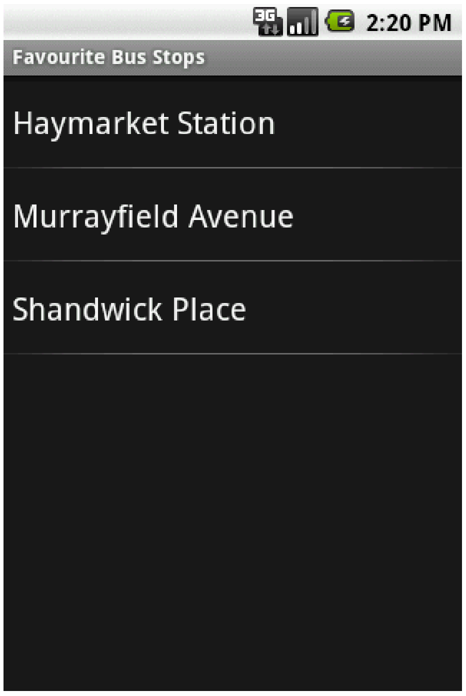


Figure 4.7 shows an example list of saved favourite stops.

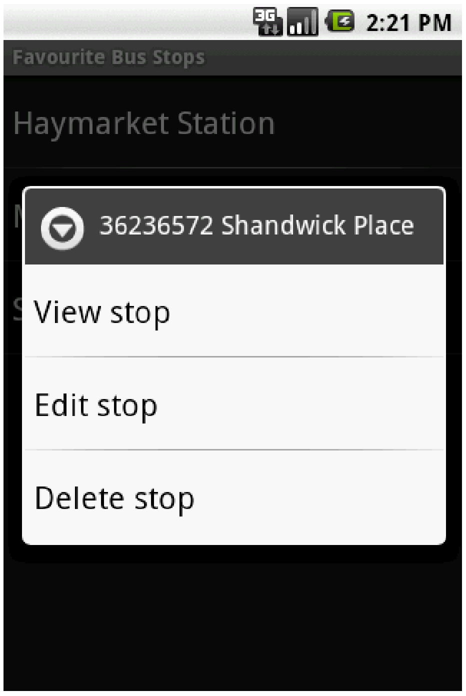
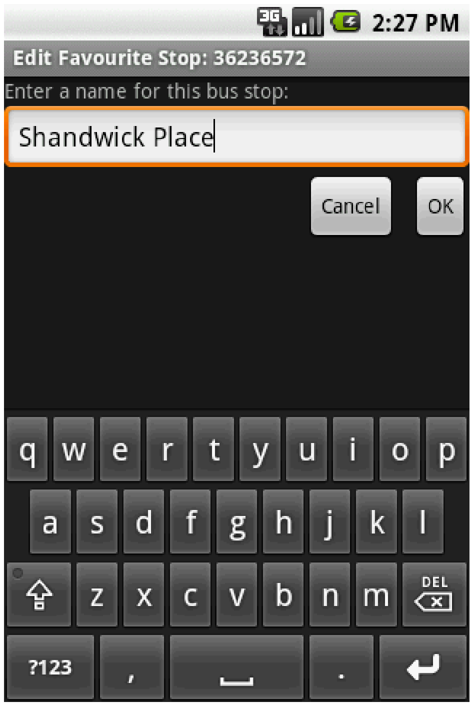


Figure 4.8 shows the same list, with the contextual menu shown.

4.3.5 AddEditFavouriteStopActivity

The AddEditFavouriteStopActivity is shown when the user wants to either add a favourite bus stop or edit an existing favourite bus stop. The user can add a favourite bus stop from DisplayStopDataActivity and can edit an existing favourite bus stop from FavouriteStopsActivity. This activity contains a label informing the user what to do, a text entry box which will accept any character the Android keyboard support, a “cancel” button and an “OK” button. Only the stop name is editable, as the user may want to name the bus stop in their favourites something other than what the Bus Tracker system provides. The “cancel” button will take the user back to the activity they have just came from, and the “OK” button will confirm the changes the user wishes to make, with that being either adding the favourite bus stop to the database, or editing the name for an existing favourite bus stop.


Edit Favourite Stop: 36236572

Enter a name for this bus stop:

Shandwick Place

Figure 4.9 shows the user editing a bus stop name.

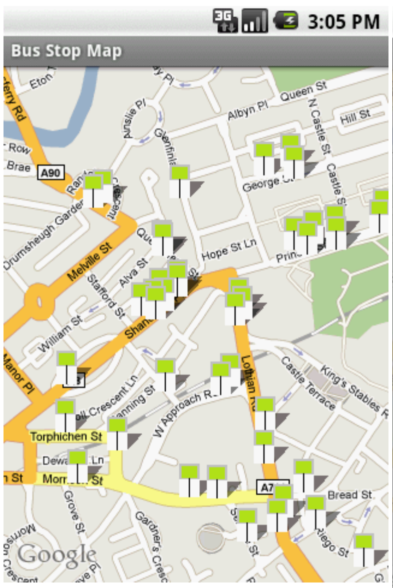
4.3.6 BusStopMapActivity

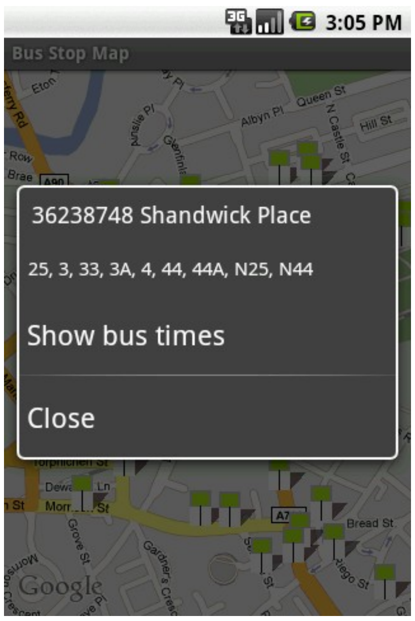
BusStopMapActivity is perhaps one of the most complicated activities in the application. It extends from MapActivity contained within the Google Maps API and hosts a MapView object. Before a project can use a MapView object, the developer must sign up for a Google Maps API key and include this key in their project to be able to download the map "tiles". This activity uses the zoom controls built in to the Google Maps API. By default, this activity will centre on Edinburgh Castle and zoom in to zoom level 12 which gives a good view of most of the city.

Google Maps can accommodate overlays on the map to allow the develop to place markers on the map. Firstly, a "My Location" overlay is created. This functionality is within the Google Maps API and it contains a class which will manage the retrieving of the device's location from the device's sensors (GPS, network triangulation). This class will also draw the location found on to the map as part of an overlay. The "My Location" overlay will continuously listen for changes to the device's location and update the overlay accordingly. When this class finds the device's location for the first time since being initialised, known in the API as the “first fix”, the map will pan so the device's location is at the centre of the map and the map will zoom in to zoom level 17.

The top overlay is that of all the bus stops. The bus stops are shown as markers on the map at their respective GPS coordinate locations. When the view of the map is changed, which can be either the map zoom level has been changed or the map has been panned, the class responsible for managing the bus stop overlay, “BusStopMapOverlay”, will work out what area on the map is visible to the user and query the SQLite database of bus stops to return only those bus stops that the user will be able to see. These stops are then converted in to map markers and then placed in the overlay with a familiar icon in their respective positions. The map markers can be tapped on to reveal a menu. The menu displays the stop code, stop name, the list of services which stop at that bus stop and a list of possible actions (show bus times or close the menu).

A menu is also associated with the map activity and contains three items. The “My Location” item allows the user to pan back to the device's current location if the map is aware of the current location. The second menu item allows the user to switch between a map view – the traditional way of viewing a Google Map, or a satellite view which shows satellite imagery instead and has street names and roads overlaid on the top. The third menu item allows the user to toggle between traffic mode being on or off – traffic mode overlays on the map areas that Google are aware of bad traffic congestion in an area.





36238748 Shandwick Place

25, 3, 33, 3A, 4, 44, 44A, N25, N44

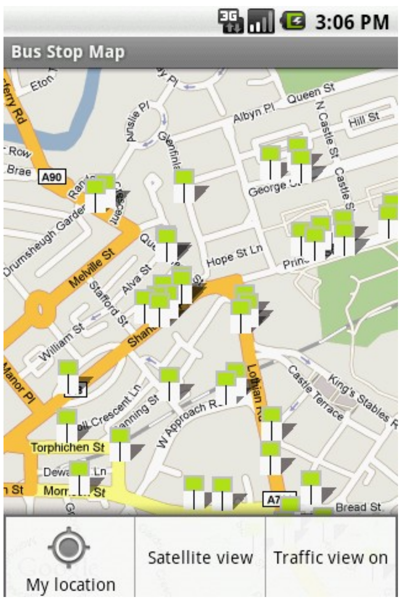


Figure 4.10 shows the map activity with the bus stop markers.

Figure 4.11 shows the menu that is displayed when a bus stop marker is tapped on.

Figure 4.12 shows the menu associated with the map activity.

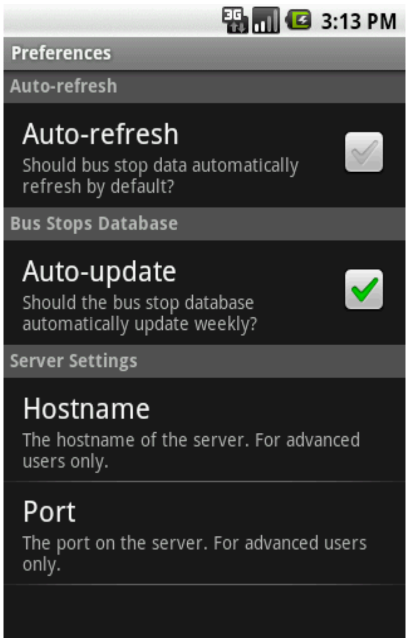
4.3.7 PreferencesActivity

The PreferencesActivty contains a list of preferences that the user can change within the application. This activity extends from PreferenceActivity, a framework built in to the Android API which allows developers to create preference activities. The structure of the preferences can be defined in an XML file and included as a project resource. This activity can then "inflate" this resource and create the relevant preference layout. This application allows the user to change a variety of preferences to affect the application behaviour;

- **Auto-refresh:** the user can select whether auto-refresh is on by default or not when they view live bus stop data. This is off by default.
- **Auto-update:** the user can select whether to automatically update the bus stop database every week or not. This is on by default.
- **Hostname:** the user can change the hostname of the server to connect to from the default. This should not normally be changed by the user and the preference description states that this setting should only be changed by advanced users (developers). If the user does change it,

they can reset it to default by changing the setting to a blank hostname which resets the hostname back to the default. The default is “bustracker.selfip.org”.

- **Port:** the user can change the port on the server to connect to. Again, this should not normally be changed by the user and it is a setting for advanced users. Like the hostname, it can be reset back to the default value by changing this setting to a blank value. The default is “4876”.



3:13 PM

Preferences

Auto-refresh

Auto-refresh
Should bus stop data automatically refresh by default? ✓

Bus Stops Database

Auto-update
Should the bus stop database automatically update weekly? ✓

Server Settings

Hostname
The hostname of the server. For advanced users only.

Port
The port on the server. For advanced users only.

Figure 4.13 shows the PreferencesActivity.

Chapter 5

Testing

Testing is an important piece of software engineering to ensure the system behaves as desired and that the system is robust to deal with any data that may be input.

5.1 Functional tests – server

The server can be tested without the use of the client using a variety of utilities. Commands can be issued to the server via a plain-text Telnet connection and the results are returned in plain-text. Other results can be obtained by observing the host's file system and using a utility called SQLiteBrowser. Testing is often done on a case of feeding the system valid, extreme valid and invalid data and observing the output to see if it is the expected output. It is important that the server is extremely robust so that users do not experience down time on the system.

5.1.1 Commands

The server is provided with valid and invalid commands. The server replies as expected with invalid commands - “Error: unknown server command.” The server replies without an error for valid commands, further tests were undertaken to determine if the data returned was correct. The only server command which accepts a parameter, “getBusTimesByStopCode”, was used without its parameter to see how the server behaved, and the expected output “Error: the number of parameters for getBusTimesByStopCode is 1.” was returned. When a command which doesn't accept a parameter is fed a parameter, it ignores the parameter and carries out its function as if the parameter does not exist.

Test	**Pass**
The server does not crash or behave unexpectedly when provided with incorrect or	Yes

wrongly formatted commands.	

5.1.2 Getting live bus stop data

The main function of the intermediate server is to retrieve the live bus stop data from the Bus Tracker web server. The server can be tested by providing the “getBusTimesByStopCode” command with a valid bus stop code, a properly formatted (an integer) but incorrect bus stop code and a wrongly formatted and incorrect bus stop code. The server returns the JSON text as expected when provided with a valid stop code. This JSON text data is compared against the Bus Tracker website for the same bus stop and the data matches up correctly. The only time the server will not do this is when it cannot connect to the Bus Tracker web server, when this occurs the server returns an error to the client. When a properly formatted but incorrect stop code (bus stop does not exist) is supplied to the server, the server will return a blank JSON text. The client knows that if the stop code and stop name returned are blank, then the stop does not exist and reports to the user that the stop most likely does not exist. If a wrongly formatted and incorrect bus stop code is supplied, the behaviour of the server is to ignore the command and not return anything. This is undesirable and will be fixed as an error of some sort should be returned. The server is volatile in the sense that if the format of the XML document coming from the Bus Tracker web server is to change, then the server will mostly likely break and the necessary amendments will need to be done to the server before it can function again.

Test	**Pass**
The server handles the live bus stop data correctly.	Mostly

5.1.3 Bus stop database creation

The server creates the bus stop database under one of these three circumstances occurring; the server is started and the bus stop database does not exist, the server is started and the bus stop database it has found is more than 7 days old, the time is 06:00:00 on a Sunday morning. It can be determined when the bus stop database was last created by finding out from the file system (such as the “ls -lh” command in UNIX) when the database file was last modified. Further to this, the UNIX timestamp (in milliseconds) of the database creation time is stored in the database and a file is created in the server root directory which contains this same timestamp. The database can be analysed using the SQLiteBrowser utility. Under normal circumstances, the database results in being about 320KB in size with around 2,500 bus stops in the database with around 8,400 bus stop to bus service mappings. The server successfully creates the bus stop database upon determining that the stop database does not exist or is more than 7 days old when it is started. The server fails to create the database at 06:00:00 every Sunday morning, instead it creates it exactly every week since the server was started. The result was found to be an incorrect assumption made about the Java API and the source code was modified appropriately. It is assumed that all of the data in the database is consistent with the Bus Tracker system as correct results are reported within the Android application whilst using the mapping functionality.

Test	**Pass**
The server generates a correct database periodically.	Yes

5.1.4 Other server commands

The server contains three other commands; a command to find the timestamp of the latest version of the database the server holds, a command to find the web accessible URL of the database a command to find the latest version of the Android client. The timestamp of the database creation time is read from a file in the server's working directory. If the server cannot find the file, a timestamp of 0 is returned and the client understands to not proceed in updating the database if this is the case, otherwise the timestamp is returned. The web accessible URL of the database is configured in the server configuration file and cannot be reconfigured whilst the server is running, the server needs to be restarted for the change to take affect. The server does not check to see if the URL is valid, it assumes it is. If a URL is not defined in the server configuration file, the a default URL is returned ("http://localhost/busstops.db"). The latest Android client version is defined in a file in the server's working directory and can be changed at run time. No validation is done on the data in the file and if the file cannot be found or it is a blank file then "Unknown" is returned.

Test	**Pass**
Other server commands return expected output.	Yes

5.2 Functional tests – client

The client can be tested from the emulator included in the Android SDK or from an Android device. Not all activities within the application need to be tested as they do not process any data. A list of activities which do not need to be tested are listed with the reason why this is the case;

- **PreferencesActivity:** this activity uses standard Android API components which have been tried and tested by the platform developers and other application developers and as such, are considered robust.
- **FavouriteStopsActivity:** this activity simply displays the results of a query to a SQLite database. The user can only perform pre-defined actions within the activity (choose an option from a menu and proceed to another activity). This activity would only likely fail if the schema of the database was to be changed, this is extremely unlikely.

5.2.1 MainActivity

To the user, this activity only displays a menu, a list of choices to other activities in the application. In the background, this activity also starts other threads which can run even when this activity is not in the foreground. One thread checks to see if the bus stop database exists and if it does not, then make a copy of the stop database from the application package to the application data directory. This usually only occurs when the application is started for the very first time on the device. This can be verified to be successful if BusStopMapActivity is able to display bus stop markers on the map.

Another thread will check with the server to see if the bus stop database and the client are up to date. Currently checking to see if the client is up to date does not function and will be enabled in a future version of the application. When the bus stop database is updated, the user is informed via a notification. The database can be verified to be successfully updated by using the Android SDK debugging tool “ddms” to extract the database from the device's file system and using the SQLiteBrowser utility to check the database creation timestamp. If the timestamp matches that of what the server is reporting as the newest version of the database, then the update is successful. This can only be verified within the emulator as production devices do not allow the debug utility to have access to the device's file system. It is planned for a future version of the application to include an AboutActivity which will state the application version and author information and list a few application statistics, one of which will be the latest database version. The application appears to update the database successfully and no issues are known about this system after testing, except for the server-side issue mentioned in section 5.1.3.

Test	**Pass**
The application can successfully update its own database from the server	Yes

5.2.2 EnterStopCodeActivity

EnterStopCodeActivity allows the user to manually enter a stop code to view the live bus stop data for that bus stop. This activity allows the user to enter the stop code via a hardware keyboard on the device or via an on-screen keyboard. The text entry box has been restricted to only allow numbers to be input. This activity does not do any processing on the stop code, it simply passes the stop code value to DisplayStopDataActivity. To test this activity is simple – press all keys on the keyboard and if anything other than numbers appear in the text entry box then the test has failed. After testing this, no issues are known about this activity.

Test	Pass
The text entry box in EnterStopCodeActivity does not accept any characters except for numbers.	Yes

5.2.3 DisplayStopDataActivity

DisplayStopDataActivity accepts a stop code via the Android intent system as a parameter and then retrieves the list bus stop data for that intent. It will accept any value for the stop code, except for blank or null values. This value is then sent to the server. The server may return the JSON text of live bus data, may return an error, or the connection may time out. This activity successfully displays an error to the user in the form of a dialog when an error occurs, such as when the connection times out or the host cannot be resolved, the JSON text is wrongly formatted or if no stop code is supplied to the activity. Even when the application connects to an unknown protocol, it will report that the incoming data can not be parsed. If the JSON text is in a different format than the application can parse, an error will be reported that the data can not be parsed. This activity will successfully attempt a refresh of the data every sixty seconds if auto-refresh has been turned on. Only one known issue exists for this activity – when an error dialog is displayed the device is rotated, if the device has accelerometers to detect the orientation of the device then the user interface will rotate too and all data within the dialog disappears. This needs to be looked in to. In summary, this activity correctly displays the JSON text it is supplied from the server.

Test	Pass
DisplayStopDataActivity correctly displays the bus stop live data and reports errors to the user if an error is encountered.	Yes

5.2.4 AddEditFavouriteStopActivity

AddEditFavouriteStopActivity allows the user to define a name for a bus stop and add the bus stop to a database on the device. No rules are placed upon what names are allowed. The bus stop has already been verified to be correct as the user can only add the stop as a favourite stop if they are viewing the live bus stop data for that stop in DisplayStopDataActivity. Upon testing this activity, a wide variety of names were input, even characters which need to be escaped in SQL and also SQL commands to test if the activity was subject to SQL injection. The activity suffered no issues and the data was displayed as it was input in FavouriteStopsActivity.

Test	**Pass**
AddEditFavouriteStopActivity successfully accepts any stop name, as well as SQL escaped characters and SQL code and does not have undefined behavoir.	Yes

5.2.5 BusStopMapActivity

BusStopMapActivity contains the Google Maps object and has an overlay to display the bus stop markers. The activity successfully displays all the bus stop markers which are within the visible map space. These markers are retrieved from the bus stops database file. This activity will fail if the database does not exist. Most of the functionality is managed by the Google Maps API. The markers appear to be in their correct geographical locations on the map and there are no known issues . Perhaps the only thing that can be improved here is the efficiency of the code to make panning the map a smoother experience for the user.

Test	**Pass**
BusStopMapActivity displays all bus stops in their correct locations, does not report any errors throughout usage and allows the user to view more information about a bus stop.	Yes

5.3 User testing

Simply testing the functionality of the application does not go far enough to test the system. Ideally, it is best to test the system in a real world environment. Once the system contained the basic functionality of viewing live bus stop data, builds of the Android application were sent to a small select group of Android users to try the application out. Individuals were also invited to test the application out on a supplied Android device if they did not possess such a device. This process continued throughout development, where new builds were sent to individuals as milestones were reached. Generally user feedback was very positive, commenting on the usefulness of such a system and particularly the Google Maps functionality.

Once the application had reached all of the objectives for this project, a build was sent out to a select list of Android users instructing them to attempt to break the application to test its robustness and to check for all round correctness. Only a single report was returned, stating that the error message changed when the connection to the server was unsuccessful and the user retried. This issue was corrected promptly.

A Google Code project exists for this project and this service provides an issue tracker. If users experience problems with the system due to a result of a bug or error then they can report the issue at the issue tracker found at http://code.google.com/p/androidedinburghbustracker/issues/list .

A questionnaire was completed by a group of people to evaluate the application. The participants were asked to undertake a number of tasks within the application. To aid ease of use, the user must press the buttons and screen as little as possible as the user should not have to cross multiple hurdles to achieve the result that they desire. The participants were asked to rate how logical they thought the process of carrying out a task was. At various points the participants were asked if they felt the application was returning correct results. At the end of the questionnaire, the participants were asked if the application had crashed on them at all, how easy the application was to use overall, if they felt the application is useful and if they would recommend this application to other Android users in Edinburgh. The participants were allowed to ask for help using the application at any point. The questionnaire is included in appendix B.

5.3.1 Results

Ratings are on a scale of one to five, with five being the most favourable. Six people were questioned in the evaluation stage. The participants appeared to use the application confidently and seemed familiar to them. In finding out how many screen and button presses it took the user to complete a task, it will be researched how to bring this number down to as few key presses as possible. At each stage when the participants were asked how logical they thought the process was to complete a task, the lowest rating was four out of five with a few ratings being five out of five. All participants agreed the process of finding a bus stop via the inbuilt mapping capabilities was very logical (five out of five). The participants who answered four out of five found that the entering a stop code process and the adding a favourite stop and retrieving this favourite stop process was fairly logical. All participants but one who forgot to answer the question on the questionnaire found that the application returned correct results. The application did not crash for any participants. 75% of the participants rated that the application was very easy to use (five out of five) and the remaining 25% of participants rated the application fairly easy to use (four out of five). All participants feel that the application is useful and that they would recommend it to other Android users in Edinburgh.

The questionnaire asks the participants to suggest improvements that could be made to the system. Improvements suggested include:

- Integrate a Twitter feed in to the application like EdinBus for the iPhone.
- Ability to filter bus routes on the map.
- Include icons within the application.
- The bus times could be displayed more clearly when viewing the live bus times.

Chapter 6

Discussion

6.1 Objectives

In the Introduction chapter, in section 1.3, a series of objectives for this project were laid out. All objectives have been achieved during the course of the project. This chapter aims to evaluate the project as a whole and discuss what obstacles were overcome to reach the objectives and how successful the project is.

6.2 Difficulties

Whilst the development of the server component went very quickly and progressed smoothly, the Android client did not go as quickly and smoothly. Putting the theory of the Android developer guide was more difficult than anticipated and time was spent having to go back and research the Android platform again or reading up on a topic that was not anticipated beforehand. The single biggest hurdle of developing the Android client was implementing the Google Maps component. It was felt that perhaps the developer documentation for this API was inadequate and answers had to be found by experimenting where the developer documentation was ambiguous or non existent, thus taking up many hours in the development of the project. Another component which was difficult to implement was usage of the SQLite database within the Android platform. While the documentation was plentiful, it was felt that the developer documentation did not properly explain how to deal with sessions to the database and this led to many hours being consumed attempting to find the source of an exception within the application.

6.3 Evaluation

All of the objectives set out at the start of the project were accomplished. The Android platform and Bus Tracker system were successfully investigated and a sufficient understanding was gained in each to allow the development of the system to proceed. The Android user environment was understood to create a familiar user interface for the application. The Google Maps API and location aware services were researched to find out how to bring mapping and location functional to the application to assist users to find their bus stop. A final system was developed which brings all of these components together.

The final system conforms mostly to the software requirements. The system is able to retrieve and display to the user live bus stop data for their chosen bus stop. The user is able to save favourite bus stops to be able to revisit them at a later time to view the live bus stop data. The application contains a Google Maps object to allow the user to search for their intended bus stop on the map. The system conforms to technical standards as standard APIs and libraries are used where possible. The system being easy to use is a matter of opinion rather than fact, but it is felt that this has been achieved due to the user feedback and questionnaire being positive. Every effort has been made to make the system memory, processor and bandwidth efficient. The code was reviewed in both server and client components to check to see where memory usage and processing could be made more efficient and the source code was modified appropriately. The system is bandwidth efficient as it uses a simple protocol to communicate between client and server and the JSON format is a bandwidth efficient way of distributing structured data. The source code for the system is made freely available at http://code.google.-com/p/androidedinburghbustracker/ but as of yet is still to be released on the Android Market. Before this is done, the application needs to be polished up by adding icons to the application and slightly improving the general appearance.

6.4 Related work

As far as it is known, there is currently not an Edinburgh Bus Tracker application for the Android platform. Several exist for the iPhone, mostly notably EdinBus which has been previously mentioned. The City of Edinburgh Council are planning on developing an API at some point to allow developers to easily integrate their systems in to the Bus Tracker system therefore more related projects are likely to exist in the future. Most applications follow a similar format – they allow the user to manually input a stop code, allow the user to maintain a list of their favourite bus stops and may include a map to allow the user to find their bus stop.

National Rail allow users to check the status of their train on their website and also have an iPhone application to allow users to check the status of their train on the move too. The application is considered the official application which can be downloaded for a fee and National Rail have not opened up their system to allow other developers to use the data. The application, as well as the website, reports the journey and stops of a train, as well as what time it is expected at each station and reports at what time it reached stations it has already stopped at. Some train operators, such as Virgin Trains and CrossCountry have created similar applications which allow the user to train the trains for those operators and are available to download for free.

A related system is in operation in the Brighton & Hove region in England. This is very similar to the Edinburgh system as they have on-street live departure boards. Several iPhone applications exist which tie in with this system and they allow the user to find their bus stop by selecting a service number followed by a stop name. One application allows the user to find their stop from a map. No Android applications exist for the Brighton & Hove system.

A system exists in Helsinki, Finland, which allows users to see buses moving on a map of the city in real time. This allows the user to see exactly where their bus is and adds an interesting twist on bus tracking [27].

6.5 Future work

The Bus Tracker for Android system could possibly be extended further than the stated objectives to enhance the user experience and increase the usefulness of the system. Possibilities include;

- The user of the application could select a bus route and the bus route could be shown on the Google Map. This allows the user to see exactly where the bus services goes and could be useful for route planning, especially for a visitor to the city. Currently the Google Maps for Android API does not have functionality to draw a route overlay on top of the map, only markers can be drawn.
- The user could filter routes on the map to only show bus stops covered by the particular routes filtered. This makes it easier to find a bus stop on the map.
- Like EdinBus, a Twitter feed could be integrated in to the application to bring update notifications to the user from Lothian Buses advising them of special changes to bus services which may affect their journey.
- The user of the application could state either by selecting on a map, or entering place names, their journey origin and destination and the application could automatically compute the optimal bus routes and services to get between the two places. Again, this would be useful to a visitor to the city, but could also be useful to habitants of the city to possibly make their bus journeys more efficient. The application would depend on more information being available but the application would use graph theory algorithms to work out the shortest routes.
- Why should the application experience end when the user gets on to the bus? Once on a bus, the user could then use the application to determine how long it will take to reach their destination, based on the timetabled time between two points and also considering factors such as

traffic congestion and accidents, the average speed of the bus using a GPS signal and also the distance from the intended destination, again based on a GPS signal.

6.6 Conclusion

This project has filled a gap as currently there are no other known Edinburgh Bus Tracker applications for the Android platform. It is hoped that many people will find the benefit of this application. This project has shown it is possible for such a system to be implemented for the Android platform and that Google Maps can be included to improve the user experience. This project has left a lot of scope for further work to add features to and improve the system to assist the user. Many people were very welcoming of such a system when presented it. It is hoped that a partnership can be made with the EdinBus developer to improve the system overall for both the Android and iPhone platforms.

It is very pleasing to see that all of the project objectives were met and that all of the software requirements were met. It is also pleasing to see that the participants in the questionnaire found the application easy to use and found the system useful as this was the intention for the system. An unexpected outcome is that The City of Edinburgh Council have expressed interest in the system produced, pointing towards future opportunities for this project. It is hoped that with collaboration with the EdinBus developer and The City of Edinburgh Council that the whole Bus Tracker system can be improved for the greater good by providing a world leading public transport system for the city of Edinburgh which is rich with information for people who may travel on it.

Bibliography

[1] The City of Edinburgh Council. "Bus Tracker goes live this November". http://download.edinburgh.gov.uk/TransportEdinburgh/BusTrackerLiveinNovember011104.pdf . Online: November 1, 2004, Cited: February 12, 2010.

[2] The City of Edinburgh Council. http://www.mybustracker.co.uk/ . Cited: February 12, 2010.

[3] Brighton & Hove City Council. http://www.journeyon.co.uk/ . Cited: March 29, 2010.

[4] National Rail. http://www.nationalrail.co.uk/ . Cited: March 29, 2010.

[5] National Rail. http://www.nationalrail.co.uk/iphone/ . Cited: March 29, 2010.

[6] Open Handset Alliance. http://www.openhandsetalliance.com/ . Cited: February 24, 2010.

[7] Android.com. "What is Android?". http://developer.android.com/guide/basics/what-is-android.html . Cited: February 24, 2010.

[8] Geoff Duncan. "Admob: Android Showing Strong Growth". http://www.digitaltrends.com/mobile/admob-android-showing-strong-growth/ . Cited: February 24, 2010.

[9] The City of Edinburgh Council. Personal communication, 2009.

[10] World Wide Web Consortium (W3C). "Extensible Markup Language (XML) 1.1 (Second Edition)". http://www.w3.org/TR/2006/REC-xml11-20060816/ . Cited: March 3, 2010.

[11] Chris Marshall, Gemma Fraser. "Software innovator touches on new way to keep track of Capital's buses". Scotland: Edinburgh Evening News. Published: December 3, 2009.

[12] Gordon Christie. http://itunes.com/apps/edinbus/ . Cited: February 12, 2010.

[13] Gordon Christie. Personal Communication, 2009.

[14] Android.com. http://developer.android.com/reference/packages.html . Cited: February 25, 2010.

[15] Android.com. http://developer.android.com/guide/developing/tools/index.html . Cited: Febuary 25, 2010.

[16] Android.com. http://developer.android.com/guide/index.html . Cited: February 25, 2010.

[17] Android.com. http://developer.android.com/guide/topics/fundamentals.html . Cited: February 25, 2010.

[18] Android.com. http://developer.android.com/guide/topics/security/security.html . Cited: February 25, 2010.

[19] Google. http://code.google.com/android/add-ons/google-apis/index.html . Cited: February 25, 2010.

[20] Google. http://code.google.com/apis/maps/ . Cited: February 26, 2010.

[21] Google. http://code.google.com/android/add-ons/google-apis/reference/index.html . Cited: February 25, 2010.

[22] Android.com. http://developer.android.com/guide/topics/location/index.html . Cited: February 25, 2010.

[23] Wei-Meng Lee. "Using Google Maps in Android". http://mobiforge.com/developing/story/using-google-maps-android . Cited: February 25, 2010.

[24] JSON.org. http://www.json.org/ . Cited: March 3, 2010.

[25] Douglas Crockford. "JSON: The Fat-Free Alternative to XML". http://www.json.org/fatfree.html . Cited: March 3, 2010.

[26] Author unknown. "JSON". http://en.wikipedia.org/wiki/JSON . Cited: March 3, 2010.

[27] Unknown author. http://transport.wspgroup.fi/hklkartta/ . Cited: March 29, 2010.

Appendices

Appendix A – source code

The source code for this project is available at

http://code.google.com/p/androidedinburghbustracker/source/checkout

The latest revision of the source code at the time of the dissertation deliverable is revision 35. A copy of the source code is also included on the optical media in the dissertation deliverable.

Appendix B - questionnaire

Android Edinburgh Bus Tracker – Evaluation

Name...

The results used in this evaluation are completely anonymous. I agree to participate in this evaluation and agree with the ethics of this project and I have the right to withdraw my consent at any time.

Signed: …... Date.....................................

This evaluation should only take 5 minutes to complete. Throughout the evaluation you may ask for help to complete tasks are ask for points to be clarified. Follow the instructions below and fill in answers to the questions.

1. The application should be open. Please keep a count of how many times you pressed the screen or touched a button on the device. Go to “Enter Stop Code”, enter “36232658” for the bus stop code and submit.

How many times did you have to touch the screen or a button? …..................
On a scale of 1 to 5, with 5 meaning very logical, how logical was this process?
…......................

2. You should still be viewing the times for bus services. Press the Menu key on the device and press on “Add favourite”. Enter a name for the bus stop and submit. Press the back key on the device 3 times and go to “Favourite Stops”. Press on your favourite stop you have just added.

How many times did you have to touch the screen or a button? …..................
On a scale of 1 to 5, with 5 meaning very logical, how logical was this process?
…......................
Was the bus times displayed for the favourite bus stop you have just added? (Yes/No)
…..................

3. Press the back key twice. Go to “Bus Stop Map”. Search for any bus stop on the map. Press on the stop marker and then view the bus times for this stop.

How many times did you have to touch the screen or a button? …..................
On a scale of 1 to 5, with 5 meaning very logical, how logical was this process?
…......................
Did the bus stop marker appear to be in the correct location on the map? (Yes/No)
…...................

Did the application crash on you at any point? (Yes/No) …........................
On a scale of 1 to 5, with 5 being very easy, how easy was the application to use?
…..................
Do you feel this application is useful? (Yes/No) …..........................
Would you recommend this application to other Android users in Edinburgh? …......................

Please suggest any improvements below.