**Table of Contents**

[Introduction 3](#_Toc445889473)

[Project Planning and Organisation 4](#_Toc445889474)

[Original Project Plan 4](#_Toc445889475)

[Final Project Plan 4](#_Toc445889476)

[Reasons for Change 4](#_Toc445889477)

[Organisational Tools 4](#_Toc445889478)

[Research 5](#_Toc445889479)

[Application Type 5](#_Toc445889480)

[Web Application 5](#_Toc445889481)

[Native Application 5](#_Toc445889482)

[Comparison of Native and Web Applications 5](#_Toc445889483)

[Android 7](#_Toc445889484)

[iOS 7](#_Toc445889485)

[Comparison of Android and iOS Applications 7](#_Toc445889486)

[Possible Competition 8](#_Toc445889487)

[Standard Routing 8](#_Toc445889488)

[Disability Improvements 8](#_Toc445889489)

[Institutional Routing 8](#_Toc445889490)

[Static or Dynamic Routing 8](#_Toc445889491)

[Static Routing 8](#_Toc445889492)

[Dynamic Routing 8](#_Toc445889493)

[Location Services 10](#_Toc445889494)

[Global Navigation Satellite System (GNSS) 10](#_Toc445889495)

[Cell Tower Based Location 11](#_Toc445889496)

[Wi-Fi Based Location 11](#_Toc445889497)

[Current Android Solution 12](#_Toc445889498)

[Improved Suggestion 12](#_Toc445889499)

[Routing 13](#_Toc445889500)

[Representing a Graph 13](#_Toc445889501)

[Edge List 13](#_Toc445889502)

[Adjacency Matrix 13](#_Toc445889503)

[Adjacency List 13](#_Toc445889504)

[Path Finding 14](#_Toc445889505)

[Breadth-First Search 14](#_Toc445889506)

[Dijkstra’s Algorithm 14](#_Toc445889507)

[A\* Algorithm 14](#_Toc445889508)

[Software Engineering 16](#_Toc445889509)

[Architectural Style 16](#_Toc445889510)

[Object-Oriented 16](#_Toc445889511)

[Event-Based Implicit Invocation 16](#_Toc445889512)

[Programming Style 16](#_Toc445889513)

[Design Patterns 17](#_Toc445889514)

[Observer Pattern 17](#_Toc445889515)

[Adapter Pattern 18](#_Toc445889516)

[Template Pattern 18](#_Toc445889517)

[Strategy Pattern 19](#_Toc445889518)

[Composite Pattern 19](#_Toc445889519)

[Design 20](#_Toc445889520)

[Initial High- Level Design 20](#_Toc445889521)

[Responsibilities 20](#_Toc445889522)

[Communications 20](#_Toc445889523)

[Initial View Design 21](#_Toc445889524)

[Initial Model Design 23](#_Toc445889525)

[Initial Controller Design 23](#_Toc445889526)

[Data Requirements 23](#_Toc445889527)

[End-Point Builder 23](#_Toc445889528)

[Mid-Point Builder 23](#_Toc445889529)

[Link Builder 23](#_Toc445889530)

[Design Changes and Improvements 23](#_Toc445889531)

[Testing Strategies 23](#_Toc445889532)

[Performance Analysis & Future Proofing 23](#_Toc445889533)

[Routing Performance 23](#_Toc445889534)

[Drawing Performance 23](#_Toc445889535)

[Future Work 23](#_Toc445889536)

[Improving the Route Modifiers Implementation 23](#_Toc445889537)

[Adding Location Services 23](#_Toc445889538)

[Reaching More Platforms 23](#_Toc445889539)

[Release and Updates 23](#_Toc445889540)

[Appendices 24](#_Toc445889541)

[Appendix 1 Strathclyde Disability Adviser Interview 24](#_Toc445889542)

[Appendix 2 Initial High Level UML Diagram 25](#_Toc445889543)

[Appendix 3 View UML Diagram 26](#_Toc445889544)

[Appendix 4 Initial Model UML Diagram 27](#_Toc445889545)

[Appendix 5 Initial Controller UML Diagram 27](#_Toc445889546)

[Appendix 6 Final UML Diagram 27](#_Toc445889547)

[Appendix 7 User Guide 27](#_Toc445889548)

[Appendix 8 Installation Guide 27](#_Toc445889549)

[References 28](#_Toc445889550)

# Introduction

The main aim of this project is to create an application that can:

* Implement a routing algorithm capable of dealing with an institute the size of the John Anderson campus.
* Visualise this route in a way that a user can easily understand.

In addition to this over the course of the project an investigation should be carried out into the feasibility of including route modifiers and location services into this application. For the purposes of this report Route Modifiers are defined as settings which depending on their values will give different routes for different users. An example of this would be to avoid stairs. Location services would be some way of automatically determining where the user is within the mapped section. This could then be used to either provide Dynamic Routing as they follow the path or to simply allow them to input their start location more easily.

# Project Planning and Organisation

This project took place over several months which meant that if time was not managed carefully important tasks may have been left incomplete. From this need a project plan was developed.

## Original Project Plan

## Final Project Plan

## Reasons for Change

## Organisational Tools

# Research

Before any development work could begin several key questions had to be answered about how the project should proceed: Would the application provide static or dynamic routing to the user? How would the Application be developed? How would the shortest path be found?

## Application Type

It was decided that the application should be able to be used on mobile devices as this offers the greatest levels of flexibility to the user. This means the user could use it throughout the day as they traverse the mapped area as opposed to having to plan their routes for the day in advance. With this requirement in mind there are two main types of application that would be possible a Web Application or a Native Application [1].

### Web Application

These applications are developed to be ran within a web browser. This means that they can be used across virtually any device including both mobile devices and more traditional computing platforms. They are designed and built as a website with special care taken to appear correctly on mobile devices. This means they can be accessed by navigating to a particular webpage on the phone.

### Native Application

Native applications are built specifically for the mobile operating system being targeted. This means if it is built for Android it cannot be used on iOS or vice versa. To use a Native application, they are first installed from an OS specific store (either the App Store for iOS or the Google Play Store for Android). They can then be launched from a home menu.

### Comparison of Native and Web Applications

To decide between Native and Web Applications they must be compared on a number of key features to the project. This reasoning can be seen in Table 1.

Table Comparison of Native and Web Applications on Key Issues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Issue to compare | Native Application | Web Application | Which is better for this project | Reason |
| Portability | Can only be used on the OS it was designed for. | Can be used on virtually any Operating System that includes support for a web browser. | Web | If this was to be released it would be important for as many students to have access to it as possible. Although multiple Native Applications could be created it would be far easier to have one Web Application. |
| Performance | Developed specifically for each platform which means greater performance. | Lower performance. | Native | Performance may be an issue as the size of the map increases. Although this is also an issue for the choice of routing algorithm it makes sense to make the later choice easier. |
| Access to phone sensors | Virtually full access to all of the phone sensors. | More restricted access to phone sensors. | Native | This could be important for location services. Although Web applications have access to geolocation information [2] they do not appear to have access to information on Wi-Fi access points they are not currently connected to [3]. Native applications by comparison have access to this information [4]. |
| Programming Language | Uses Object Oriented languages (Java or Objective C) which makes Encapsulation, Abstraction, etc far easier than they would have been otherwise. | Uses JavaScript for the main bulk of work. This can be programmed in an object oriented fashion but it is much more difficult to do so. Does not enforce strict type-checking meaning errors can be made far more easily during development. | Native | It provides a better framework for producing an error free application more quickly. |

As can be seen from Table 1 Native applications have a number of advantages that are important to the project particularly for the study into Location Services feasibility. There are two major choices to be considered when developing a Native Application: Android and iOS.

### Android

These applications are developed specifically for the Android OS and are widely available for download in the Google Play Store. Android phones are produced by a range of manufacturers including Sony, Samsung and HTC.

### iOS

iOS applications are developed specifically for Apple’s mobile devices i.e. the iPhone and iPad. They are available for download on these devices through the App Store.

### Comparison of Android and iOS Applications

A number of issues should be considered before deciding on a platform to build the application for. Several of these are examined in

Table Comparison of Android and iOS Applications on Key Issues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Issue | iOS | Android | Better for this project | Reason |
| Percentage of users worldwide | 13.9% [5] | 82.8% [5] | Android | Users are far more likely to own an Android phone and therefore be able to use the application. |
| Integration with mapping services | Wide support for many major mapping applications including Google [6] and Bing [7]. | Similar levels of support from the major mapping applications [8] [9]. | Tie | Both options offer similar levels of integration with popular mapping applications. |
| Availability | None. | Several devices of different sizes and features. | Android | Circumstances surrounding the project have dictated these are more easily available. |

For the reasons detailed above the application will be developed as an Android application.

## Possible Competition

### Standard Routing

### Disability Improvements

### Institutional Routing

## Static or Dynamic Routing

To determine whether the application should provide static or dynamic routing it is first important to understand what both involve.

### Static Routing

For the purposes of this report Static Routing will be defined as routing between two fixed points with the whole route available for the user to view. A popular example of this is google maps which can be seen in Figure 1.

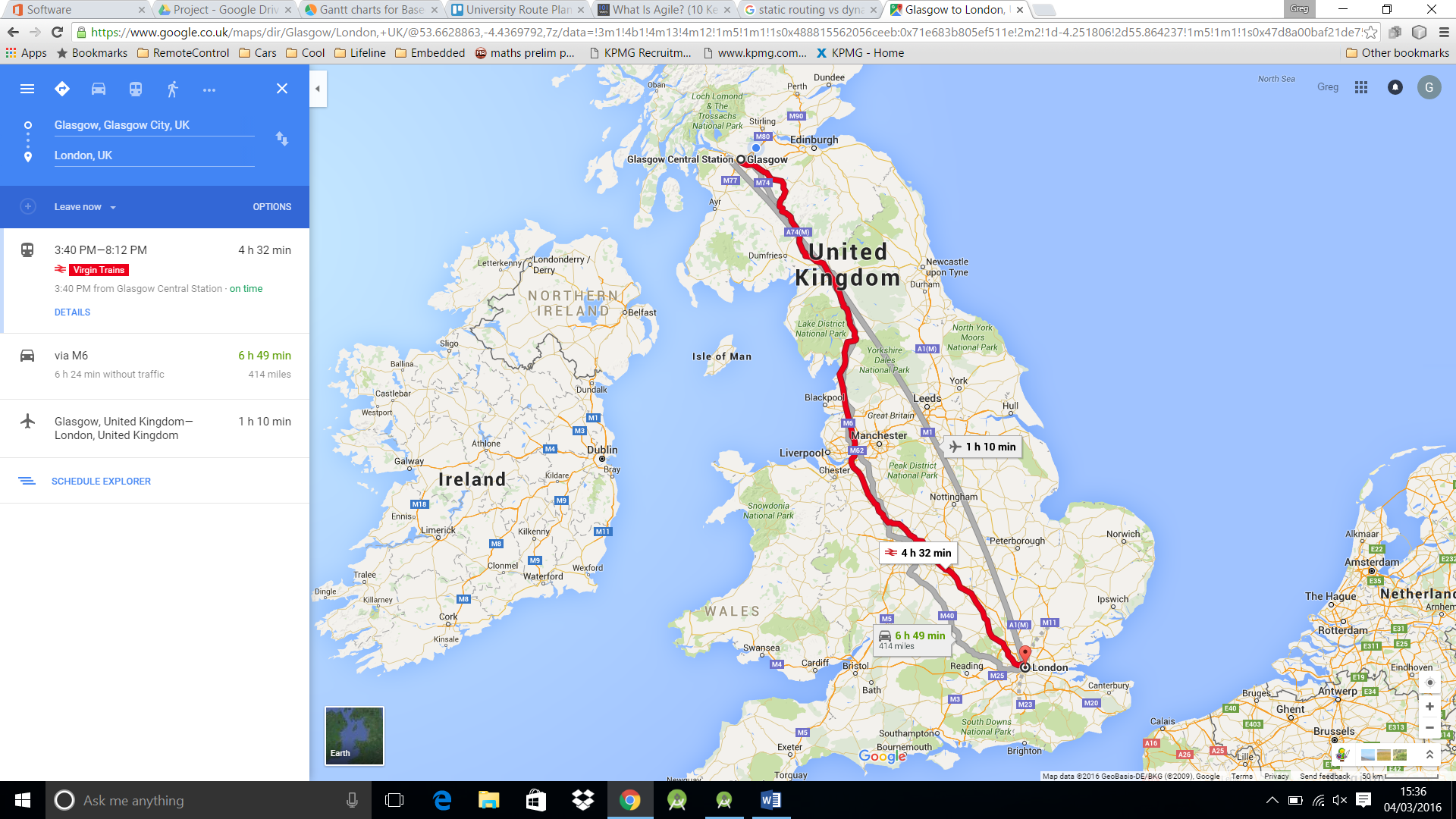


Figure Static Routing Example [10]

### Dynamic Routing

By comparison Dynamic Routing will be defined as routing from the user’s changing location to a fixed endpoint. This means as the user moves the route will automatically adjust to their new location. This is the type of routing used in Satellite Navigation systems or Google Navigation which can be viewed below in Figure 2.

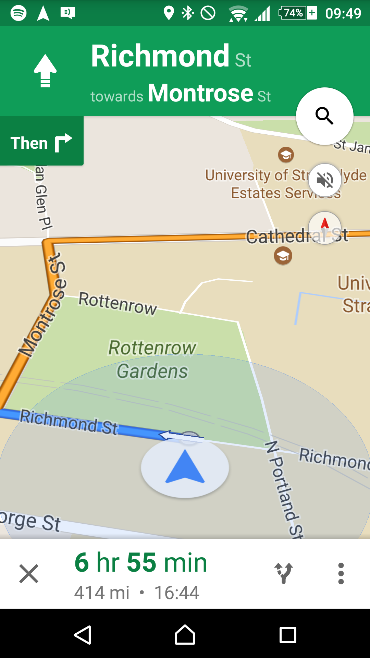


Figure Dynamic Routing Example

It is also common for the display to be more restricted showing only a short section of the route at a time i.e. where the user is to go next.

## Location Services

The main difference between the two types of routing is whether the user’s location is utilised to improve the user interface or not. In order to decide between them then it is important to determine whether this is possible.

There are three main technologies which could be used to determine the user location: Global Navigation Satellite System, Cellular Positioning and Wi-Fi.

### Global Navigation Satellite System (GNSS)

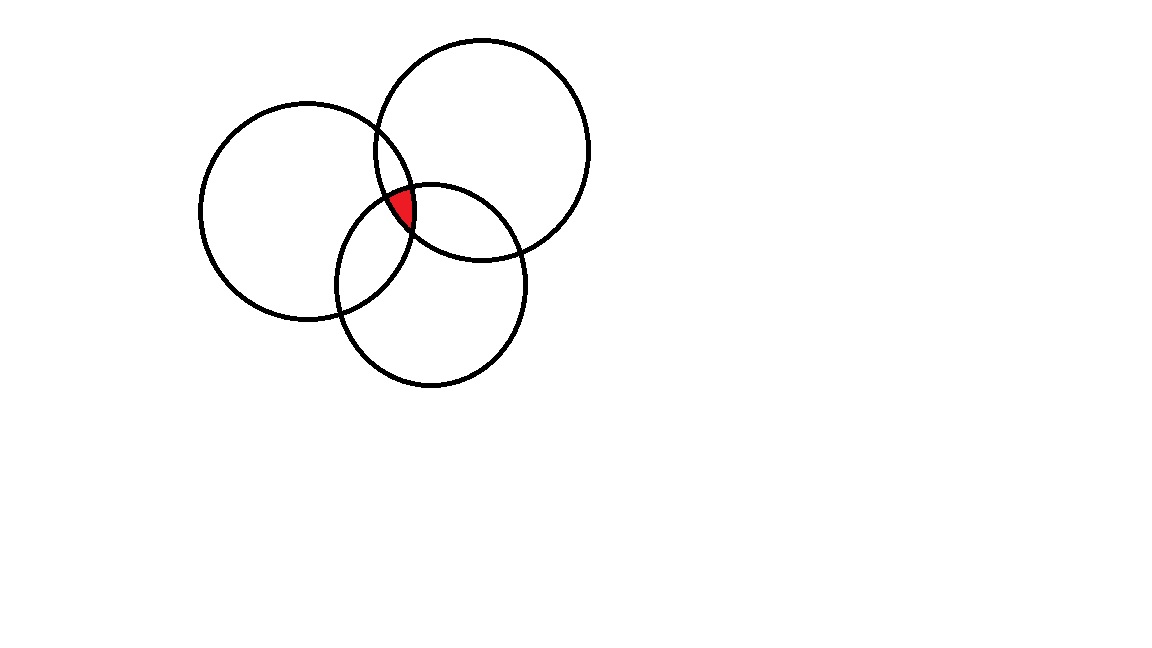
There are several of these systems currently in operation. The most famous of these being the U.S. funded Global Positioning Service (GPS) however there are others such as the Russian Federation’s GLONASS and soon the European Galileo [11]. These were devised as systems to enable users worldwide to calculate their current location. The GPS system for example comprises of twenty-four satellites continuously producing signals that tell a user where a satellite is and highly accurate timing information about when this signal was produced. All twenty-four satellites communicate on the same two frequencies using Code Division Multiple Access in order to avoid collisions. By listening to the signals generated by at least four of the satellites to discover their locations and calculating the time it has taken to reach the user and hence how far away they are from each the users own location can be calculated. [12]

The accuracy of GPS is roughly 3.5m on the horizontal plane without using additional information. There are other systems which can be used in conjunction with GPS to provide up to millimetre accuracy however 3.5m should be accurate enough for this project [13]. The accuracy of Galileo is expected to improve to the centimetre range [14] once it is completed in 2020 [15].

There are two main limiting factors however: the accuracy of the GPS receiver in the user’s phone and that much of the time the user will be indoors. The accuracy of the receiver in a typical mobile phone is roughly 2-3m under good multipath conditions or up to 10m under bad multipath conditions [16]. Multipath is when the signal may reach the receiver via many paths for example direct from the satellite compared to being reflected by a building in between the satellite and receiver. This means that the signal will degrade when near buildings due to the weak multipath suppression available on mobile phone antennae. The app will always be expected to be running in an urban landscape therefore meaning the GNSS location may have quite a low accuracy. Receiving GNSS signals indoors can become a challenge as the signal when received outdoors is already very weak when compared with the background noise and the signal can be 10-100000 times weaker when indoors [17]. This can make it difficult to receive the necessary signals from four satellites at once making it near impossible to reliably and quickly determine location indoors via this method.

### Cell Tower Based Location

Location can be calculated using cell towers in a similar way to that of GNSS. Essentially the phone sends a ping to the nearby cell towers and times how long it takes to receive a response. If the locations of the cell towers are known then an estimate of the phone’s location can be calculated using the respective distances from each cell tower [18]. An example of how this works can be seen below in Figure 2 Cell Tower Triangulation.



1

3

2

Figure Cell Tower Triangulation

In the figure above the arrows represent the calculated distance from the cell tower. With only one cell tower response this means the user could be anywhere in the surrounding circle. This is as the furthest the user could possibly be from the tower would be is a straight line from it (the radius of the circle) however there is no guarantee that is the path being taken. Instead it could be reflecting off of buildings or other materials meaning the user could be anywhere within the surrounding circle. With responses from more cell towers the location becomes more accurate. As can be seen in Figure 2 Cell Tower Triangulation with these responses the user could only be in the overlapping area from each circle.

Cell Tower Triangulation could potentially be a more reliable method of geolocation as it even tends to work indoors, presuming there is a mobile phone signal, and even works better in more urban areas where this application would be expected to be used, due to the higher number of cell towers [18]. It does however have some drawbacks in that it requires an accurate knowledge of cell tower locations which are not typically available publically making this more difficult [19]. On average for example in the United States of America (USA) the average accuracy of a three tower triangulation is 0.75 square miles [20]. This is far too inaccurate for the purposes of this application.

### Wi-Fi Based Location

In much the same way cell towers can be used to calculate the user’s location so can Wi-Fi base stations. By using easily available data such as signal strength a rough estimate of distance from the base station can be calculated. If several base stations can be identified, then the user’s location can be calculated using triangulation.

### Current Android Solution

Android as an OS has several built in features for detecting the user’s current position and provides app developers with the opportunity to use this data. It currently uses a combination of all 3 of the location services previously mentioned [4]. The accuracy while outside is particularly high as can be seen in

TODO

This is likely due to the availability of a strong GPS signal in combination with cell tower and Wi-Fi location services. When inside however the provided location services are far weaker as can be seen in Figure 4.

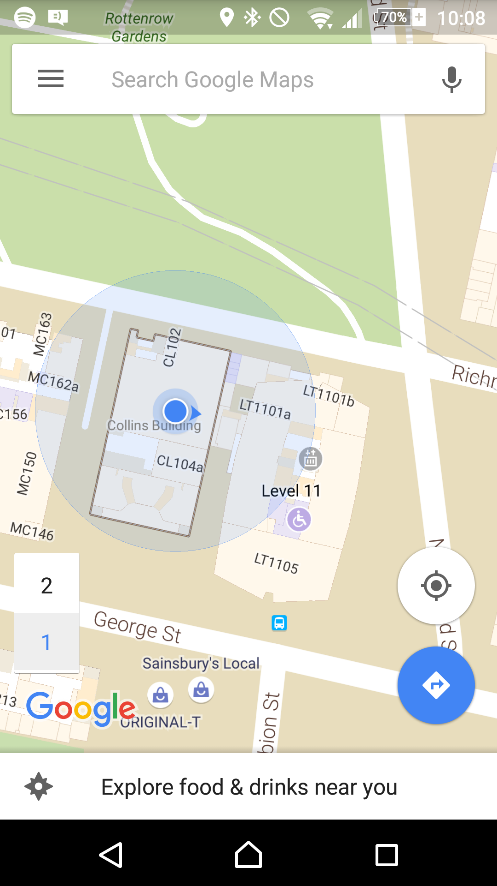


Figure Indoor Location Example

In Figure 4 again the reported location is shown with a small blue circle this can be compared to the actual location represented by the red circle to show that when indoors there can be a large margin of error. The larger blue circle represents the area in which google admits the location may fall within. Although technically the actual location does fall within this area the data is still almost unusable from the perspective of this app. The area of uncertainty shown above covers an entire building as well as significant amounts of two neighbouring buildings meaning this is unlikely to be useful when trying to navigate within one building. This large degree of uncertainty and inaccuracy is most likely due to a lack of a reliable GPS signal combined with a low knowledge of the exact Wi-Fi and Cell Tower locations. This should be improving however as google is trying to improve its knowledge of Wi-Fi locations using their massive user base. This is being done by taking the times when a user has a good knowledge of their own location using GPS to send google information on the surrounding Wi-Fi signals and strengths [21]. This could eventually theoretically provide strong indoor locations worldwide however is currently falling short of this. Even if it does improve using GPS as the starting point means height will always likely be a problem meaning differentiating between floors of a building could be difficult.

### Improved Suggestion

As previously stated it is believed that the main problem currently facing indoor location services is a knowledge of the locations of different Wi-Fi sources. Although steps are being made to try to improve this globally within a smaller area that is to be mapped such as a University campus this problem becomes far easier to deal with. Within this area the

locations could be found manually preferably by using documentation written during the installation of a Wi-Fi network however if not a Wi-Fi map could potentially be built up when the app is being built.

## Routing

In order to build an application that will provide a route for the user between their chosen start and destination a graph representing the area to be mapped must be built. Once the graph has been built a method for finding the shortest path between two nodes in that graph must be found to indicate the route that the user should follow.

### Representing a Graph

There are three main ways of representing a graph within a computer system Edge List, Adjacency Matrix and Adjacency List [22].

#### Edge List

Consists of a list of all of the edges in the system. This is a fairly easy to understand representation of a graph however can be difficult to traverse during path finding algorithms. By indexing the information by edges rather than nodes it can make moving from one node to another more difficult. Each time it is needed to check what edges are connected to the current node it is necessary to move through all of the edges checking whether or not that node is attached to that particular edge.

#### Adjacency Matrix

Consists of a Matrix with the nodes along both axes and the weight of the edge connecting them at the point where they meet. An example of this can be seen in Figure 5 and Figure 6.

Figure Example Graph

4

C

6

5

D

1

A

B

2

2

E

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | Null | 2 | 4 | Null | 1 |
| B | 2 | Null | Null | Null | 2 |
| C | 4 | Null | Null | 6 | Null |
| D | Null | Null | 6 | Null | 5 |
| E | 1 | 2 | Null | 5 | Null |

Figure Example Adjacency Matrix

Adjacency matrices are typically used in very densely populated graphs i.e. those with a lot of edges per nodes. This is as the number of edges does not affect the storage space required for this implementation however if the graph is sparser with each node having far fewer edges than there are other nodes this method becomes increasingly inefficient with almost the entire matrix being filled with null values to represent no edge.

#### Adjacency List

This is a method in which each node has a list of the nodes it is connected to. By storing the data in this way it makes routing algorithms far simpler as it is very easy to quickly determine which nodes are connected to the one that is currently being examined. Naturally this does not support edge weights however this functionality can be added via a slight modification. By storing a list of edges for each node this provides all of the benefits of an Adjacency List however also allows for weights to be added to the connections allowing for a representation of a weighted graph.

### Path Finding

Once the graph representation has been constructed an algorithm for finding the shortest path must be implemented. There are many algorithms that have been developed for finding the shortest path between two nodes in a graph and each have their own areas of expertise depending on the type of graph.

#### Breadth-First Search

This algorithm starts at one of the two nodes it is to find the route between. From there it then moves out to all of the surrounding nodes and assigns them a value of one. It then takes each of the nodes with a value of one and assigns all of the surrounding nodes with a value of two. This continues until it reaches the desired node. It is guaranteed to find the path between the two in the fewest number of hops however does not take into account edge weights.

By not taking into account differing edge weights it would make constructing a graph representing the campus to be mapped far more difficult. This is as each of the nodes would have to take roughly the same amount of time to walk between to produce an acceptable route. This becomes particularly difficult when adding lifts to the routes as waiting on lifts could potentially take several minutes. Waiting times like this mean either all of the nodes would have to be particularly far apart, which is likely not possible in a corridor with many rooms coming off of it, or there would have to be conceptually many nodes the user would move between while waiting on a lift which is essentially just adding weights in a very inefficient way.

#### Dijkstra’s Algorithm

The main problem with breadth first is that it doesn’t allow for different weights on the edges. Dijkstra’s Algorithm by comparison will always find the shortest path in a weighted graph. It works by essentially starting at one of the two nodes that the path is between and marking it as examined. Each of the nodes that it is connected to then have a number representing the weight of the edge taken to reach them assigned to them. From there the node with the lowest number associated with it is marked as examined and all the nodes it is connected to have their numbers updated to be the sum of the weight to the examined node plus the new edge unless they already have an associated number which is less. This continues until the destination node is marked as examined at which point the shortest path between them has been discovered.

#### A\* Algorithm

The A\* Algorithm is essentially a modification of Dijkstra’s algorithm that allows the shortest path to be found more quickly. It does this by adding a heuristic value to the comparison when deciding which node to examine next. An example of a possible heuristic value for this case would be the straight line distance between each node and the current destination. This means that when choosing which node to examine next it would add both the difficulty in reaching that node and how physically far away from the destination the node is. This means that instead of spreading out equally in all directions like in breadth-first or towards the easiest direction like in Dijkstra’s it would instead spread towards the destination quicker than in other directions. The algorithm relies on the fact that the heuristic function always returns a value less than or equal to the length of the shortest path from that node to the destination. Calculating a heuristic value like this can be fairly simple for some applications for example a single level mapping application could use straight line distance fairly easily. However, a multi-level mapping application such as this would face difficulties in implementing such a system. Calculating the straight line distance for example would be impossible without detailed knowledge of the heights of each building. Even if the straight line distance could be calculated it would likely be a very poor heuristic value as it could lead the search algorithm to move towards the destination node even if the quickest route would be to move in the opposite direction towards a stair case. It would still find the quickest route but in this case it would likely be less efficient than Dijkstra’s despite the additional effort required to include the heuristics.

## Software Engineering

To develop a system of this scale and ensure it is easy to maintain and extend it is important to try to use good software engineering practices throughout. This should aid development and testing easier by making the software more flexible and forgiving of change. During this report the various styles and patterns will be referred to as taught in the class CS409 Software Architecture and Design at Strathclyde University.

### Architectural Style

An Architectural Style is the highest level programming choice to be made. It is concerned with how data and functionality should be separated. For example, the “Main Program/Subroutine shared data style” in which all of the data is available throughout the program and the functionality is broken up into subroutines which can be called when needed. For this project the architectural style is for a large part dictated by the decisions that have already been made. By choosing to develop for android this means the programming language that will be used is Java. As java is best suited to the Object-Oriented architectural style this is the main style that will be used throughout the project. However due to the way the Android OS and support libraries have been built up there will also be aspects of Event-Based Implicit Invocation included throughout.

#### Object-Oriented

This particular architectural style relies on splitting the program into objects, instances of classes, which each have access to their own particular data and can perform their own actions. Objects can then communicate via method calls between them. This leads to greater levels of abstraction meaning a change to one section of code will not dramatically alter other sections within the program. An instance of this could be if one object requests the quickest route between two nodes from a second object. The second object could be changed internally, to represent the graph differently or use a different path finding algorithm, but it wouldn’t affect the first class in any way provided the nodes to be passed in and the route returned still obeyed some kind of standard definition. The best way to create a standard definition is by creating an interface. An interface is a list of the publically available method calls that other classes can use to communicate with any classes implementing that interface. Many languages that support interfaces and classes, including java, also support abstract classes which are to some extent an amalgamation of the two. They give implementations to some of their methods however leave it to subclasses to implement others.

#### Event-Based Implicit Invocation

Event Based Implicit Invocation is similar to the Object Oriented style in many ways. However, instead of communicating between components via method calls an object can register for updates based on a particular event. This is best exemplified during use of the User Interface (UI) objects provided in the Android Software Development Kit(SDK). When a UI object is created a class which implements the provided interface can register for updates about events such as it being touched or swiped. Then when the user next touches or swipes on that UI object the registered object will be notified with relevant information such as the location of the touch.

### Programming Style

A good Programming Style will provide a high-level guide to how the program should be laid out and how both functionality and data will be separated. There are many different programming styles however since re-usability was a major factor in choosing which to adopt it was decided Model-View-Controller would be used.

Model-View-Controller divides data and functionality into the three main sections mentioned in its’ name. The Model should deal with everything programmatically for example in this project that would be the graph and path finding algorithms amongst other matters. The View should determine how that information is provided to the user for example this could be a map or simply a list of instructions. It is important that out-with the defined interface for communicating that these components do not rely on each other. This means the View could be changed entirely without any changes at all being necessary in the model. The Controller deals with user input and determines how that should affect the model. An instance of this in the project could be if the user selects a new room in some way it is the controller’s responsibility to alert the model to this change in state. Again outside the well-defined interfaces the controller should be able to be changed entirely without affecting either the model or the view.

By using this programming style it means components even at a particularly high level are particularly interchangeable. This is important in case at some point during development something changes dramatically. A possible reason for this would be if instead of using Google Maps it was decided university blueprints would be more accurate. Even in this case it would mean only the view would need to be changed leaving two thirds of the code with no alterations needed despite the large change in functionality.

### Design Patterns

Design Patterns are essentially proven solutions to commonly recurring problems. Throughout programming certain problems such as how to change algorithms during runtime appear again and again. Through use of these design patterns we have a way to deal with them effectively. Below is an overview of each of the design patterns which have been used throughout the project.

#### Observer Pattern

The Observer Design Pattern is essentially a lower level version of the same principles the Event-Based Implicit Invocation Architectural Style was based on. Essentially with it one component allows others implementing a specific interface to register with it for updates. Whenever this first object determines itself to have changed in some way it notifies all of those listening for updates. Using the Observer Pattern allows for two components which hold similar data to remain consistent while also staying loosely coupled. This design pattern can be seen most clearly in the sample UML diagram shown in Figure 7.

Observer

Update()

Update(State)

Concrete Subject

getState()

Subject

addObserver(Observer)

notify()

Concrete Observer

Figure Observer Design Pattern UML

The main advantage to this design pattern is that it very strongly decouples the concrete subject from the concrete observer allowing any changes to the concrete observer to be made easily. The main disadvantage however is that it can make debugging potential problems more difficult as the flow of code becomes less obvious. There are two versions of the Observer pattern: the push model and the pull model. They both follow the same basic structure however with one major difference. The push model gives the Observer the information it requires when it is notified. By comparison the pull model simply alerts the Observer that something has happened. This means the Concrete Observer requires a reference to the Concrete Subject in order to determine what has changed and react accordingly. These differences can be seen clearly in Figure 7 with the aspects only in the push model shown in blue and those only in the pull model shown in red. The push model has the added advantage of also cleanly decoupling the Concrete Observer from the Concrete Subject.

#### Adapter Pattern

This pattern was designed to solve the problem in which two different parts of a system rely on different interfaces to do essentially the same thing. It can be particularly useful when using frameworks or libraries which have already been developed with new code that for some reason needs to have its’ own independent interface. There are two main variations of the Adapter pattern, Class and Object, however as Class relies on multiple inheritance a feature which is only available to a minimal extent in java for the purposes of this project only the Object variation will be considered. The structure of this pattern can be seen in Figure 8.

Client

Target

Adapter

Adaptee

Figure UML Representation of Adapter Pattern

In the diagram shown in Figure 8 the client needs to use the specific interface “Target” however the necessary function is actually within “Adaptee” which cannot implement this interface. In order to solve this a new class, “Adapter”, was created to take the data from client and change it into the format needed by “Adaptee”.

#### Template Pattern

The template pattern is used to define the order in which steps are carried out but leave the implementation of these steps open to be changed dependent on what is needed at the time. This allows much of the outline and other functionality of a class to be defined in one class and for a certain bit of functionality to only be defined in a subclass. The Template Pattern can be seen in Figure 9 UML Representation of Template Pattern.

Abstract Class

templateMethod()

fixedMethod1()

variableMethod()

fixedMethod2()

Concrete Class

variableMethod()

Figure UML Representation of Template Pattern

In Figure 9 the abstract class would implement the template method and both fixed methods. It would then leave the variable method to be implemented by its’ subclasses. The template method would make calls to the fixed and variable methods in the order it requires them knowing that the concrete class will be able to provide the variable methods. The concrete class then implements the variable methods allowing the class to perform the duties it needs to. The main benefit of the template pattern is that it allows multiple classes to be created quickly that differ in one area of their implementation. For example, in a drawing package it would allow multiple shapes (the abstract class) such as square, circle or triangle (the concrete classes) to draw themselves differently despite needing to perform the same actions before and after such as checking there was adequate space and alerting a listener to the change. The main disadvantage is that this cannot be changed at runtime once the class is created it will always perform that action in the same way.

#### Strategy Pattern

The Strategy Pattern is used when a particular piece of functionality needs the ability to be changed at runtime. For instance, if a camera application used an algorithm to provide photographs with a filter applied to them the Strategy Pattern would allow users to change which filter they wished to apply while the application was running. A representation of this pattern can be seen in

Context

Strategy

Algorithm()

Concrete Strategy A

Algorithm()

Concrete Strategy B

Algorithm()

Figure UML Representation of Strategy Pattern

Essentially how this works is by having multiple classes

#### Composite Pattern

# Design

## Initial High- Level Design

As previously discussed the high level design should follow the Model-View-Controller programming style. To do this successfully it is important to outline exactly what each subsystem will be responsible for in this design and how they will communicate. For a full overview the high level UML diagram is shown in Appendix 1 Initial High Level UML Diagram.

### Responsibilities

The model will deal with the state of the program i.e. what the currently selected start and end nodes are, where the user is currently looking, what errors have occurred and what the current route is. It will also be the only subsystem to deal with the data as a graph and will therefore be responsible for route finding.

The view will deal with how the current state of the program will be displayed to the user for example what rooms will be displayed, how the route will be given and what inputs will be taken from the user.

The controller will deal with reacting to user input and changing the model accordingly. When it is informed of some form of user input from the view it will decide in which way this user input will affect the model.

### Communications

The communications between each of the three subsystems can be seen in Table 3.

Table High Level Communications

|  |  |  |  |
| --- | --- | --- | --- |
|  | Model | View | Controller |
| Model | n/a | Using the observer design pattern in the push variation. | n/a |
| View | n/a | n/a | View will hold a reference to Controller through the IController interface. This will allow |
| Controller | Controller will hold a reference to the Model through the IModel interface. | n/a | n/a |

By using the Observer pattern to communicate between the model and view it keeps these two sub-systems independent from one another. The controller to view and controller to model communications will pass through interfaces this will aid decoupling however not to the same extent as the Observer pattern does between the model and view. This is as instead of just passing through the data and allowing the other object to with it what it wants the components now call specific methods. This is a compromise that has to be made however because the links between the view to controller and controller to model are too complex to be broken down into an observer pattern.

## Initial View Design

The first issue that must be addressed when designing how the view should function is understanding exactly how it will display the supplied information and interact with the user. It was decided early in the project that from an aesthetic point of view it would be preferable to show the route on top of a pre-existing map.

By using a pre-existing map, it would both cut development time and ensure that the output looked pleasing. As previously discussed there are many mapping applications which could be used or extended in order to provide this functionality [8] [9]. As the application was being developed for the Android OS the users would likely be familiar with Google Maps as it comes pre-installed on most Android phones. Google provides an extensive number of library classes to aid in the production of Android applications using their maps. These classes provide ways to add the map to a section of the screen and then perform a series of actions to the map such as moving location or drawing elements on top. It also provides a number of interfaces which can be implemented and then added as listeners to the map object. These listeners are one half of the Observer pattern and provide an easy way to react to the user performing actions on the map.

To display the route to the user it was decided that showing the entire map at once would be far too overwhelming for the user so instead the concept of planes was developed. A plane is an area of the map that can be displayed for example outside or a particular floor of a building. Each plane then has a series of endpoints and midpoints associated with it. Endpoints are where the user may be trying to reach for example outside these could be a building or on a floor of a building these would be rooms. Midpoints are points that would be needed to follow the route for example entrances to rooms or points where corridors join. As Midpoints are not necessarily useful for the user unless they are on the route they are following it was decided that Endpoints would always be displayed when on that plane however midpoints would only be used to draw the route.

As discussed Google provides a variety of options for adding items to the map two in particular stood out as ways in which to provide information to the user: Polygons and Polylines. Polygons can be drawn onto the map with a variety of design choices and would be used to alert the user to the locations of the Endpoints mentioned above. Polylines are essentially lines that can be drawn between two points on the map in a variety of colours. It was decided that these would be drawn between the relevant midpoints to display the route to the user.

The main user interactions that will be required from the applications are: selecting the start and destination, starting routes and navigating around the map to discover the campus.

To select the start and destination it was decided that a variety of options should be available to the user. The choice of using Polygons to display the endpoints was also partially influenced by the ability to add a listener to the map to discover when these polygons are clicked by the user. This ability was then used to give the user the option of selecting endpoints by tapping on them. This selection would need to be displayed to the user again in some way to inform them that their selection had been successful. Two methods were employed to do this: changing the colour of the selected endpoints and adding two text boxes to the application displaying the names of the currently selected endpoints. It is also a possibility that the user won’t know the exact location of the room however and so it was also decided that they should be able to type the name of the start or destination into the provided text boxes in order to select them as well.

Starting routes is relatively simple and only requires the option to refresh the route when they wish so a button was added providing this functionality.

Navigating around the map was more difficult than providing the other user interactions. Although at a basic level swiping to move, pinching to zoom and other basic navigation interactions are provided by Google there are some more complex interactions introduced by the concept of planes. An instance of this would be when the application starts and the user is viewing the exterior of the campus they need some way to alert the application they now wish to view the interior of one if the buildings. To do this an “inOut” button was added that would be used to either move inside the last selected endpoint if the user was currently viewing the outside plane or return outside if the user was currently viewing one of the indoor planes.

Moving inside however caused a variety of new problems. Once the user was viewing one of the internal planes the map provided by google was modified to show floor plans allowing the user to see corridors, rooms, etc. However, this meant theoretically the user could enter one building and then simply swipe along to view the inside of another building. Doing this however meant the application had no way of knowing which building the user was viewing meaning it could not draw on the necessary additions. Although google does provide a listener to alert the application when a new building is focused the information available at this time was not sufficient to distinguish one building from another as all that is provided is the number of floors in that building. To stop this from happening it was decided that the user would be stopped from viewing anything but the building they had selected while on an interior plane. This was achieved by the development of a CameraLimiter object which would question the IController to determine which building the user was currently viewing and force the view to return to the bounds of that building if the user tried to leave the area.

Once inside it was also necessary to allow the user to select which floor they would like to view. Google provides a level selector toolbar to allow the user to do this however they do not provide a direct listener for this UI element. Fortunately, when a user selects a new floor and it is displayed an event is created named IndoorLevelActivated which a listener is available for. Unfortunately, this event is also created for the default floor in every building when the floor plans are revealed i.e. whenever the user leaves the Outside plane. This means that the listener needs the ability to be turned off in certain circumstances to stop the view forcing the controller to try to move to a variety of other floors which may not even be within that building.

It was decided that an easy way for the user to follow the route between planes was needed. In order to do this another google maps component was used: Markers. A marker is an icon which can be added to the map and have a listener added to detect when the user clicks on it. This ability was then used to allow the application to move to the next relevant plane when the marker was selected by the user. It was decided these markers would be added at the first and last midpoints on the route for the plane being viewed.

As discussed earlier in order to react to user interactions a variety of Google Maps interfaces needed to be implemented. As it is the Controller’s responsibility to react to user interactions ideally the controller should implement these new interfaces. However, many of the interfaces involve passing data such as Polygons or Markers to the class which implements them. This would be an inappropriate amount of coupling between the view and controller as it would reveal the view’s inner workings to the component in the form of google maps specific components. If these were exposed it would make it far more difficult to change the view in any serious way without dramatically altering the controller. In order to combat this the adapter pattern was used. For each of the google maps specific interfaces a corresponding class was developed which would take the view specific data and convert it into a form which could be understood by the Controller. They would then convey these changes directly to the Controller using the pre-defined IController interface. This usage of the Adapter pattern can be seen in Appendix 2 View UML Diagram where IController is the Adaptee and the view (or more specifically it’s google maps components) are the client.

## Initial Model Design

## Initial Controller Design

## Data Requirements

## End-Point Builder

## Mid-Point Builder

## Link Builder

## Design Changes and Improvements

# Testing Strategies

# Performance Analysis & Future Proofing

## Routing Performance

## Drawing Performance

# Future Work

## Improving the Route Modifiers Implementation

## Adding Location Services

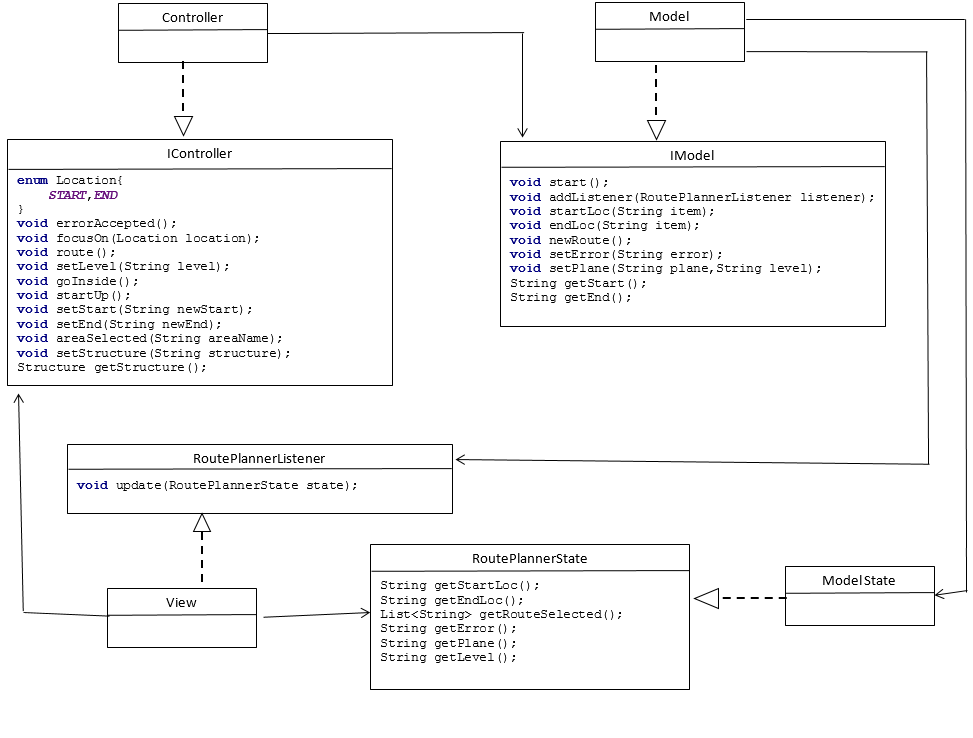
## Reaching More Platforms

## Release and Updates

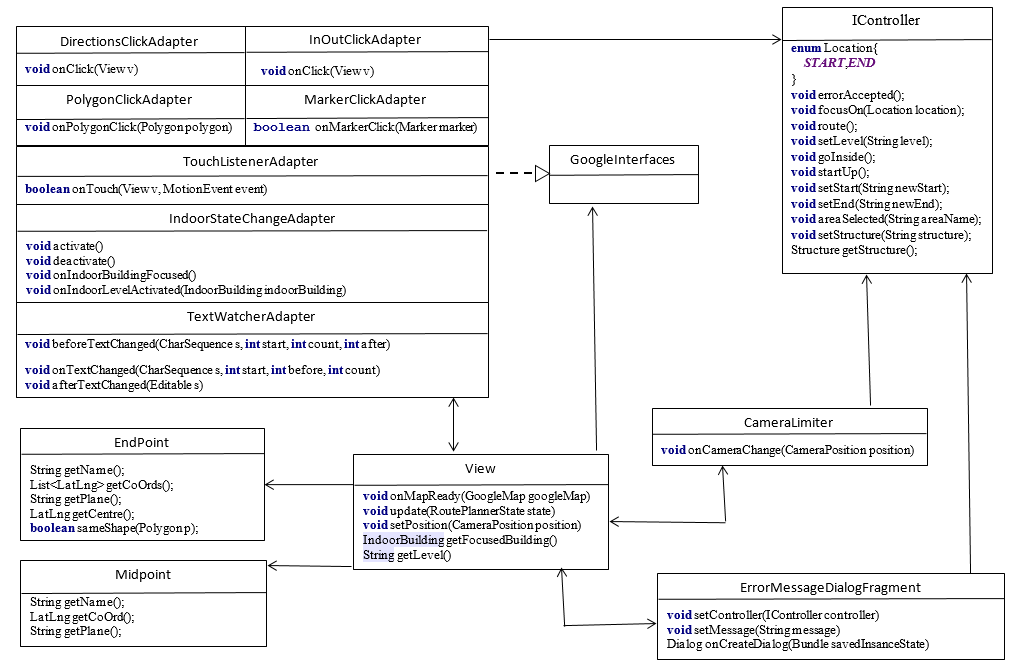
# Appendices

## Appendix 1 Strathclyde Disability Adviser Interview

## Appendix 2 Initial High Level UML Diagram



## Appendix 3 View UML Diagram



Note: Each of the adapters have their own google interfaces however they were treated as a block to simplify the diagram

## Appendix 4 Initial Model UML Diagram

## Appendix 5 Initial Controller UML Diagram

## Appendix 6 Final UML Diagram

## Appendix 7 User Guide

## Appendix 8 Installation Guide

# References

|  |  |
| --- | --- |
| [1] | P. Viswanathan, “about tech,” about, 15 December 2014. [Online]. Available: http://mobiledevices.about.com/od/additionalresources/a/Native-Apps-Vs-Web-Apps-Which-Is-The-Better-Choice.htm. [Accessed 08 March 2016]. |
| [2] | w3schools, “w3schools,” [Online]. Available: http://www.w3schools.com/html/html5\_geolocation.asp. [Accessed 8 March 2016]. |
| [3] | A. De Rosa, “envatotuts+,” 8 August 2014. [Online]. Available: http://code.tutsplus.com/tutorials/html5-network-information-api--cms-21598. [Accessed 8 March 2016]. |
| [4] | Google, “Location Strategies,” [Online]. Available: http://developer.android.com/guide/topics/location/strategies.html. [Accessed 8 March 2016]. |
| [5] | International Data Corporation, “Smartphone OS Market Share, 2015 Q2,” [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed 8 March 2016]. |
| [6] | Google, “Google Maps SDK for iOS,” [Online]. Available: https://developers.google.com/maps/documentation/ios-sdk/. [Accessed 8 March 2016]. |
| [7] | The Bing Maps Team, “New Bing Maps iOS SDK,” Microsoft, 5 May 2011. [Online]. Available: https://blogs.bing.com/maps/2011/05/05/new-bing-maps-ios-sdk/. [Accessed 8 March 2016]. |
| [8] | Google, “Google Maps Android API,” [Online]. Available: https://developers.google.com/maps/documentation/android-api/. [Accessed 8 March 2016]. |
| [9] | R. Brundritt, “Bing Maps Android SDK available on Codeplex,” 31 March 2011. [Online]. Available: https://blogs.bing.com/maps/2011/03/31/bing-maps-android-sdk-available-on-codeplex/. [Accessed 8 March 2016]. |
| [10] | 4 March 2016. [Online]. Available: https://www.google.co.uk/maps/dir/Glasgow/London,+UK/@53.6628863,-4.4369792,7z/data=!3m1!4b1!4m13!4m12!1m5!1m1!1s0x488815562056ceeb:0x71e683b805ef511e!2m2!1d-4.251806!2d55.864237!1m5!1m1!1s0x47d8a00baf21de75:0x52963a5addd52a99!2m2!1d-0.1277583!2d51.507350. |
| [11] | M. Rouse, “TechTarget,” September 2005. [Online]. Available: http://searchnetworking.techtarget.com/definition/GNSS. [Accessed 7 March 2016]. |
| [12] | E. K. a. C. Hegarty, “GPS Overview,” in *Understanding GPS: Principles and Applications*, Artech House, 2005, pp. 3-4. |
| [13] | GPS.gov, “GPS Accuracy,” U.S. Air Force, 18 September 2014. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/. [Accessed 7 March 2016]. |
| [14] | T. Bowler and M. Wall, “BBC News,” BBC, 26 August 2014. [Online]. Available: http://www.bbc.co.uk/news/business-28779524. [Accessed 7 March 2016]. |
| [15] | European Space Agency, “European Space Agency,” 18 December 2015. [Online]. Available: http://www.esa.int/Our\_Activities/Navigation/The\_future\_-\_Galileo/What\_is\_Galileo. [Accessed 7 March 2016]. |
| [16] | K. M. Pesyna, Jr, R. W. Heath, Jr and T. E. Humphreys, “GPS World,” 2 February 2015. [Online]. Available: http://gpsworld.com/accuracy-in-the-palm-of-your-hand/. [Accessed 7 March 2016]. |
| [17] | iPosi, “iposi.com,” 2014. [Online]. Available: http://www.iposi.com/tech-talk/the-technology. [Accessed 7 March 2016]. |
| [18] | D. Vaughan, “Geographic Location,” in *Windows Phone 7.5 Unleashed*, Sams Publishing, 2012, pp. 529-552. |
| [19] | J. Yang, A. Varshavsky, H. Liu, Y. Chen and M. Gruteser, “Accuracy Characterization of Cell Tower Localization,” UbiComp '10, New York, 2010. |
| [20] | Federal Communications Commission, “Accurate Location Detection 911 Help SMS App,” 2015. |
| [21] | S. J. Vaughan-Nichols, “ZDNet,” 16 November 2011. [Online]. Available: http://www.zdnet.com/article/how-google-and-everyone-else-gets-wi-fi-location-data/. [Accessed 9 March 2016]. |

[Figure 1 Static Routing Example [10] 7](#_Toc445560360)

[Figure 2 Dynamic Routing Example 7](#_Toc445560361)

[Figure 3 Cell Tower Triangulation 9](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc445560362)

[Figure 4 Indoor Location Example 10](#_Toc445560363)

[Figure 5 Example Graph 11](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc445560364)

[Figure 6 Example Adjacency Matrix 11](#_Toc445560365)

[Figure 7 Observer Design Pattern UML 15](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc445560366)

[Figure 8 UML Representation of Adapter Pattern 16](#_Toc445560367)

[Table 1 Comparison of Native and Web Applications on Key Issues 4](#_Toc445560368)

[Table 2 Comparison of Android and iOS Applications on Key Issues 6](#_Toc445560369)

[Table 3 High Level Communications 17](#_Toc445560370)