Abstract

**Table of Contents**

[1 Introduction 1](#_Toc447122880)

[2 Project Planning and Organisation 2](#_Toc447122881)

[2.1 Original Project Plan 2](#_Toc447122882)

[2.1.1 Research 2](#_Toc447122883)

[2.1.2 Implementation 2](#_Toc447122884)

[2.1.3 Testing 3](#_Toc447122885)

[2.1.4 Possible Extensions 3](#_Toc447122886)

[2.1.5 Write Up 3](#_Toc447122887)

[2.2 Final Project Plan 4](#_Toc447122888)

[2.2.1 Research 4](#_Toc447122889)

[2.2.2 Design 4](#_Toc447122890)

[2.2.3 Development 5](#_Toc447122891)

[2.2.4 Testing 5](#_Toc447122892)

[2.2.5 Write-Up 6](#_Toc447122893)

[2.3 Organisational Tools 6](#_Toc447122894)

[2.3.1 Project Planning Tools 6](#_Toc447122895)

[Figure 1 Trello Board for Original Project Plan [1] 7](#_Toc447122896)

[2.3.2 Version Control 7](#_Toc447122897)

[2.3.3 Documentation 8](#_Toc447122898)

[3 Research 9](#_Toc447122899)

[3.1 Application Type 9](#_Toc447122900)

[3.1.1 Web Application 9](#_Toc447122901)

[3.1.2 Native Application 9](#_Toc447122902)

[3.1.3 Comparison of Native and Web Applications 9](#_Toc447122903)

[3.1.4 Android 10](#_Toc447122904)

[3.1.5 iOS 10](#_Toc447122905)

[3.1.6 Comparison of Android and iOS Applications 10](#_Toc447122906)

[Table 1 Comparison of Android and iOS Applications on Key Issues 11](#_Toc447122907)

[3.2 Possible Competition 12](#_Toc447122908)

[3.2.1 Standard Routing 12](#_Toc447122909)

[Figure 2 Examples of Google Maps Functionality 12](#_Toc447122910)

[Figure 3 Examples of Google Maps Indoor Routing Attempts 13](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447122911)

[3.2.2 Institutional Routing 13](#_Toc447122912)

[Figure 4 Smithsonian Institution Map 14](#_Toc447122913)

[Figure 5 Smithsonian Institution Floor Plan 14](#_Toc447122914)

[Figure 6 Glasgow Uni Map Screenshot 15](#_Toc447122915)

[3.2.3 Disability Improvements 15](#_Toc447122916)

[Figure 7 Disabled Go Strathclyde University Colville Building Survey Section 16](#_Toc447122917)

[3.3 Static or Dynamic Routing 17](#_Toc447122918)

[3.3.1 Static Routing 17](#_Toc447122919)

[Figure 8 Static Routing Example [13] 17](#_Toc447122920)

[3.3.2 Dynamic Routing 17](#_Toc447122921)

[Figure 9 Dynamic Routing Example 18](#_Toc447122922)

[3.4 Location Services 19](#_Toc447122923)

[3.4.1 Global Navigation Satellite System (GNSS) 19](#_Toc447122924)

[3.4.2 Cell Tower Based Location 20](#_Toc447122925)

[Figure 10 Cell Tower Triangulation 20](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447122926)

[3.4.3 Wi-Fi Based Location 21](#_Toc447122927)

[3.4.4 Current Android Solution 21](#_Toc447122928)

[Figure 11 Outdoor Location Example 22](#_Toc447122929)

[Figure 12 Indoor Location Example 22](#_Toc447122930)

[3.4.5 Improved Suggestion 23](#_Toc447122931)

[3.5 Routing 24](#_Toc447122932)

[3.5.1 Representing a Graph 24](#_Toc447122933)

[3.5.1.1 Edge List 24](#_Toc447122934)

[3.5.1.2 Adjacency Matrix 24](#_Toc447122935)

[Figure 13 Example Graph and Corresponding Adjacency Matrix 24](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447122936)

[3.5.1.3 Adjacency List 25](#_Toc447122937)

[3.5.2 Path Finding 25](#_Toc447122938)

[3.5.2.1 Breadth-First Search 25](#_Toc447122939)

[3.5.2.2 Dijkstra’s Algorithm 26](#_Toc447122940)

[3.5.2.3 A\* Algorithm 26](#_Toc447122941)

[3.6 Software Engineering 28](#_Toc447122942)

[3.6.1 Architectural Style 28](#_Toc447122943)

[3.6.1.1 Object-Oriented 28](#_Toc447122944)

[3.6.1.2 Event-Based Implicit Invocation 29](#_Toc447122945)

[3.6.2 Programming Style 29](#_Toc447122946)

[3.6.3 Design Patterns 30](#_Toc447122947)

[3.6.3.1 Observer Pattern 30](#_Toc447122948)

[Figure 14 Observer Design Pattern UML 31](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447122949)

[3.6.3.2 Adapter Pattern-----------------------Not Redrafted-------------------------- 31](#_Toc447122950)

[Figure 15 UML Representation of Adapter Pattern 32](#_Toc447122951)

[3.6.3.3 Template Pattern-----------------------Not Redrafted-------------------------- 32](#_Toc447122952)

[Figure 16 UML Representation of Template Pattern 33](#_Toc447122953)

[3.6.3.4 Strategy Pattern-----------------------Not Redrafted-------------------------- 33](#_Toc447122954)

[Figure 17 UML Representation of Strategy Pattern 34](#_Toc447122955)

[3.6.3.4 Composite Pattern-----------------------Not Redrafted-------------------------- 34](#_Toc447122956)

[Figure 18 UML Representation of Composite Pattern 35](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447122957)

[4. Design 36](#_Toc447122958)

[4.1 Initial High-Level Design 36](#_Toc447122959)

[4.1.1 Responsibilities 36](#_Toc447122960)

[4.1.2 Communications 37](#_Toc447122961)

[Table 2 High Level Communications 37](#_Toc447122962)

[4.2 Initial View Design 38](#_Toc447122963)

[4.2.1 Displaying Necessary Information to The User 38](#_Toc447122964)

[4.2.2 User Interactions ------------You’re Here ------------------------------------------- 39](#_Toc447122965)

[4.3 Initial Model Design 41](#_Toc447122966)

[4.4 Initial Controller Design 42](#_Toc447122967)

[4.6 Data Requirements 43](#_Toc447122968)

[5 Testing Strategies 45](#_Toc447122969)

[5.1 Full System Testing 46](#_Toc447122970)

[5.2 Unit Testing 47](#_Toc447122971)

[5.3 Sub-System Testing 47](#_Toc447122972)

[6 Performance Analysis & Future Proofing 48](#_Toc447122973)

[6.1 Routing Performance 48](#_Toc447122974)

[6.2 Drawing Performance 48](#_Toc447122975)

[7 Future Work 49](#_Toc447122976)

[7.1 Improving the Route Modifiers Implementation 49](#_Toc447122977)

[7.2 Disabled Go Integration 50](#_Toc447122978)

[7.3 Multiple Route Options 51](#_Toc447122979)

[7.4 Reaching More Platforms 52](#_Toc447122980)

[7.5 Release and Updates 53](#_Toc447122981)

[7.6 Adding Location Services 53](#_Toc447122982)

[8 Future Applications 54](#_Toc447122983)

[8.1 Without Extensions 55](#_Toc447122984)

[8.1.1 Recruitment 55](#_Toc447122985)

[8.1.2 Conferences 55](#_Toc447122986)

[8.1.3 Deliveries 55](#_Toc447122987)

[8.2 With Extensions 56](#_Toc447122988)

[8.2.1 Disability Services 56](#_Toc447122989)

[8.2.2 Fire Safety 56](#_Toc447122990)

[Appendices 57](#_Toc447122991)

[Appendix 1 Original Project Plan 57](#_Toc447122992)

[Appendix 2 Final Project Plan 58](#_Toc447122993)

[Appendix 3 Comparison of Native and Web Applications 59](#_Toc447122994)

[Appendix 4 Initial High Level UML Diagram 60](#_Toc447122995)

[Appendix 5 View UML Diagram 61](#_Toc447122996)

[Appendix 6 Model UML Diagram 62](#_Toc447122997)

[Appendix 7 Controller UML Diagram 63](#_Toc447122998)

[Appendix 8 Full System Test Results 64](#_Toc447122999)

[Test One 64](#_Toc447123000)

[Figure 19 Test One Result 64](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123001)

[Test Two 64](#_Toc447123002)

[Test Three 65](#_Toc447123003)

[Figure 20 Test Two Result 65](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123004)

[Figure 21 Test Three Result 65](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123005)

[Test Four 66](#_Toc447123006)

[Figure 22 Test Four Result 66](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123007)

[Test Five 67](#_Toc447123008)

[Figure 23 Test Five Result 67](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123009)

[Test Six 67](#_Toc447123010)

[Figure 25 Test Six Result 67](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123011)

[Test Seven 68](#_Toc447123012)

[Figure 26 Test Seven Result 68](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123013)

[Test Eight 68](#_Toc447123014)

[Figure 27 Test Eight Result 69](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123015)

[Test Nine 70](#_Toc447123016)

[Figure 28 Test Nine Result 70](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123017)

[Test Ten 70](#_Toc447123018)

[Figure 29 Test Ten Result One 71](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123019)

[Figure 30 Test Ten Result Two 71](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123020)

[Figure 31 Test Ten Result Three 72](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123021)

[Figure 32 Test Ten Result Four 73](file:///C:\Users\kbb12\Documents\Uni\Project\UniversityRoutePlanner\Report.docx#_Toc447123022)

[Appendix 9 User Guide 73](#_Toc447123023)

[Appendix 10 Installation Guide 73](#_Toc447123024)

[References 74](#_Toc447123025)

# 1 Introduction

The main aim of the 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 at the University of Strathclyde.
* Visualise the provided route in a way that a user can easily understand.

An additional aim of the project is to carry out an investigation into the feasibility of including route modifiers and location services. Route Modifiers are defined as settings which by changing their values would allow the user to change the route provided. An example of using a route modifier would be to avoid stairs. Location services would be a method of automatically determining where the user is within the mapped area. The user’s location 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.

# 2 Project Planning and Organisation

To ensure the project could be completed by the final deadline, knowledge of the status of the project was important. In order to confirm that at all points throughout the project the current status was known, a project plan was created. The project plan meant that if the project was not going to be complete by the final deadline, the time deficit could be identified early. Identifying the time deficit early would allow changes to be made to rectify the problem.

## 2.1 Original Project Plan

The original plan was created by splitting the project into five main sections: Research, Implementation, Testing, Possible Extensions and Write up. Each of the main sections were then sub-divided into smaller tasks, that could be tackled individually. There were two subtasks which had firm deadlines: Poster Creation (1st December 2015) and Report Write-up (1st April 2016). Both of the tasks with firm deadlines dictated when other sections must be completed by. The poster was created to show the research that had been carried out, thus strongly implying that the research section should have been completed before then. The report is to cover everything completed within the project meaning ideally everything should be completed before the report write up. The full original project plan can be viewed in Appendix 1 Original Project Plan.

### 2.1.1 Research

The Research section of the project was to be responsible for discovering how the system could operate. Research was to be performed into several areas including: the choice of platform for the application and the way in which routes through the graph would be generated. The decision was made that ten days was a suitable amount of time to create the poster. From the length of time taken to create the poster, a deadline of the 18th of November for the research section was calculated. The time available was then split amongst the sub tasks within research.

### 2.1.2 Implementation

The Implementation of the system was to be completed by the 7th of February. The deadline for Implementation was chosen carefully to allow a suitable amount of time for the final feature to be tested, an overall system test to be performed and the report to be written in time for the final deadline. During the Implementation phase of the project the decision was made that the main phases of work would be: to create the graph, implement the chosen routing algorithm, find a way to display the result to the user and allow the user to add “route modifiers”. The time available between creating the poster and the deadline set for Implementation gave the overall time available. The time available was divided amongst these main tasks based on the perceived difficulty of each task.

### 2.1.3 Testing

Four main aspects of testing were anticipated to be employed throughout the project: Unit Testing, Sub-System Testing, Full System Testing and User Trials. The Testing portion of the project was anticipated to mainly be performed in parallel with other sections of the project. The Unit Testing was to be carried out as each small section of the application was created, with Sub-Section testing being carried out at the end of each new feature addition. In order to allow enough time to complete the Implementation to a suitable standard, the decision was made to carry out User Trials in parallel with writing the final report. As the majority of planned testing would be carried out during the timescale allotted for other sections, only a small amount of testing would need to be carried out independent of other sections. The testing which would require time independent of other sections was the end of Sub-System Testing and all of Full System Testing. Two weeks was allotted for the small amount of independent testing required.

### 2.1.4 Possible Extensions

Extensions that had been considered for the project were not given an allotted amount of time. The decision was made that the key aims of the project were far more important and were hence given all of the available time. The extensions were included in the project plan in order to ensure if other tasks were completed in advance, that the available time would be used to attempt to complete the additional objectives.

### 2.1.5 Write Up

The Write-Up section consists of the two deliverables for the project. The timescales allotted to both of these deliverables have purposefully been made larger than was expected to be required. Allowing longer for the write-up section, and planning to finish the project over two weeks before the deadline, gives a significant buffer for the project. Creating a buffer gives other sections the ability to expand if necessary, while still ensuring the deliverables are completed before their respective deadlines.

## 2.2 Final Project Plan

As the project progressed the initial project plan became impossible to follow precisely. As problems occurred, and the aims of the project evolved, the project plan was continually updated to reflect the current state of the project. The final project plan is available in Appendix 2 Final Project Plan.

### 2.2.1 Research

The research section was largely unchanged from the original plan. However, as the project progressed and new aims were uncovered, the need for a second phase of research became apparent. Once the potential for the working system became clearer, different possible applications of the project were studied. In addition, the potential improvement of adding location services to the application became more obviously important. Due to the newly discovered importance of location services, a study into the feasibility of finding the user’s location was added to the new second phase of research.

As the decision was made that a second phase of research would be required, the project plan had to be adapted to allow time for the additional research without affecting the final deliverables. As discussed in Section 2.1.5 Write Up, extra time was allocated to writing the report. A portion of the surplus time available within the report write-up was re-allocated to the second research phase.

### 2.2.2 Design

Originally design and development were both encompassed within the implementation task. As the project progressed, these two tasks became clearly very separate.

The design was approached from a more Agile perspective instead of using the traditional Waterfall methodology. Using an Agile approach to project management, meant that as the development work was carried out the design was constantly being iterated upon. By gradually improving the design as the project progressed, as opposed to maintaining the original design, a stronger overall design was created.

The design stage of the project was strongly informed by the research stage and vice versa, meaning beginning the design work during the time allotted to the research phase made sense. The iterative approach also meant that design work would be taking place over the Development time. Carrying out the later design stages in parallel with development allows both to take place during the time originally allocated to Implementation.

### 2.2.3 Development

The development section in the final plan is essentially the implementation section from the original plan with some modifications. One major modification to the original plan was the inclusion of a prototype application. The prototype drew a selection of shapes on google maps. When a shape was tapped by the user the outline and body of the shape would change colour. The planned development work took significantly less time than was expected, with some tasks coming in over a week ahead of schedule.

The combination of adding the prototype and completing the previously planned development work, would have left the project ahead of schedule. However, during development a major problem with the project was discovered: sourcing the data necessary to build the graph and display the results. The problem resulted in the need to create three additional applications to compile the data necessary for the main application. Creating these additional applications and using them to build maps for several buildings and floors within the John Anderson Campus, left the project three weeks behind schedule.

The three weeks that were needed to ensure the project was completed on time were taken from other areas of the project. Two extra weeks of development time were taken from the two-week buffer that had been allocated at the end of the project. The final required week was taken from the testing section.

### 2.2.4 Testing

As development was beginning the original testing strategy was found to be inadequate. A more appropriate solution was discovered to be performing full system tests as the development work was being carried out. The unit and sub-system tests were performed once the development work was largely completed. The reasons for changing the order of the tests will be covered in more detail during Chapter 5 Testing Strategies.

The testing section fits within the timescale outlined in the original plan despite the re-arrangement of the tasks. However, as was mentioned previously, the testing section needed to be shortened by a week in order to allow the overall project to be completed on time. To lessen the time spent on the testing section, the week of testing that did not run parallel to any other work from the original plan was delayed to be completed during the report write up.

### 2.2.5 Write-Up

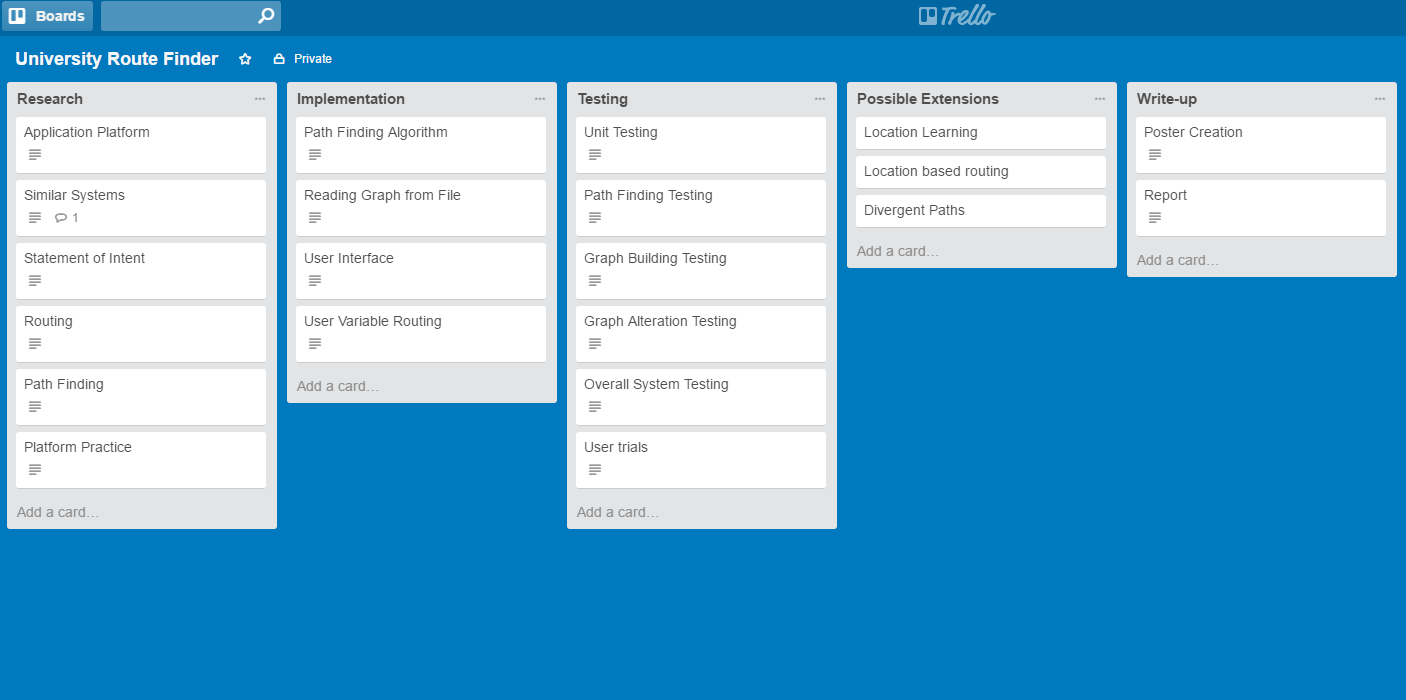
The time taken to complete both the report and poster were largely as anticipated during the original project plan. The only changes to this section was to allow other work, the second research phase and last stage of testing, to be completed in parallel to writing the report. This was anticipated to be the case when the original plan was developed and was the main reason so much time was originally allocated to the report write-up.

## 2.3 Organisational Tools

In order to complete the project within the timescale outlined in Chapter 2.2 Final Project Plan several organisational tools were employed.

### 2.3.1 Project Planning Tools

In order to create the original project plan as discussed in Chapter 2.1 Original Project Plan the task was broken down into tasks and then again into subtasks. To break down the tasks in an organised manner the project management tool Trello was used. Trello is a webpage or mobile application which allows the user to create lists then add cards to these lists. Cards can be used to represent a task in a project and can have a number of attributes associated with them such as due-dates or pictures. A section of the Trello board used to create the original project plan can be viewed in Figure 1.



###### Figure Trello Board for Original Project Plan [1]

The original project plan was kept for documentation purposes. However, the lists were copied to a new board which was updated as the project progressed. By regularly updating this new board the progress and aims of the project could be easily tracked.

Another important organisational tool was Ganttify [2] which can be used to take project plans developed in Google Calendar or Trello and create a Gantt chart to represent the plan. Gantt charts, created using Ganttify, for both the original and final plans can be found in Appendix 1 Original Project Plan and Appendix 2 Final Project Plan respectively.

### 2.3.2 Version Control

On any project with a sizeable software component using some form of Version Control is important. For this project Git was used as the Version Control System. Using Git gives a variety of benefits to the project, including being able to work on features in parallel on different branches. By using different branches for different features, the changes in one cannot negatively affect another. Importantly using Git also gives the ability to revert to a previous version of the software, if some change breaks key functionality. The revert ability means once a working version has been created that version can always be easily returned to at a later date.

Git requires a central server, to which work can be uploaded to or downloaded from, in order to provide the full level of functionality. The Git server for this project was provided by Github [3]. By using Github a copy of all the work is stored on their servers. The copy not only allows the project to be worked on from multiple machines; but also means that, provided the work is regularly pushed to the server, no large amount of work could be lost if anything was to happen to a work station.

### 2.3.3 Documentation

On a project of this size maintaining a record of the work that has been completed throughout is important. There are two ways in which the record of work has been maintained: writing a Logbook and using the Git Log. During the project whenever any design or research work has been carried out, notes and the purpose of the work has been documented in the logbook. In addition to this whenever anything is added to or changed in the project, Git requires that a message detailing the changes be written. The messages that have been added, known as the Git log, also form a record of work, as they outline what has been changed at each stage and frequently why the changes have been made.

# 3 Research

Before any development work could begin, several key questions had to be answered about how the project should proceed. To answer these questions investigations into several different aspects of the project were carried out.

## 3.1 Application Type

It was decided that the application should be able to be used on mobile devices, as mobile applications offer the greatest level of flexibility to the user. Creating the application for mobile means that a user could use the application throughout the day, and not just when they have access to a computer. Using the application throughout the day would be more convenient, as doing so allows the user to follow routes while they traverse the mapped area. Following routes while navigating the area, would require far less advanced planning for the user when compared to only planning routes while they have access to a computer. With the requirement that the application run on mobile devices, there are two main types of application that would be possible: a Web Application or a Native Application [4].

### 3.1.1 Web Application

Web applications are developed to be ran within a web browser, and are built as a website with special care taken to appear correctly on mobile devices. Running within a browser means that they can be used across virtually any device, including both mobile devices and more traditional computing platforms. Designing the webpage for mobile devices means they can be accessed and easily operated by navigating to the webpage on the phone.

### 3.1.2 Native Application

Native applications are built specifically for the mobile operating system being targeted. As native applications are platform specific, if the application is built for Android the application 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). The application can then be launched from the home menu.

### 3.1.3 Comparison of Native and Web Applications

To decide between Native and Web Applications they must be compared on a number of attributes important to the project. The main issues that were considered along with discussion on which matches each requirement more closely for the project can be found in Appendix 3 Comparison of Native and Web Applications.

As can be seen from Appendix 3 Comparison of Native and Web Applications, 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.

### 3.1.4 Android

Android 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.

### 3.1.5 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.

### 3.1.6 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 the most important issues are examined in Table 1.

###### 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. |

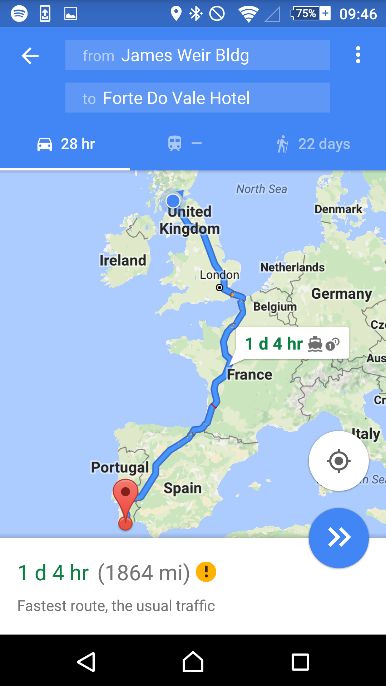
For the reasons detailed in Table 1 the application will be developed as an Android application.

## 3.2 Possible Competition

Before creating the application, a study into what other options are currently available to the user should be carried out. By understanding what is currently available, differentiating the new application from the existing competition will become easier.

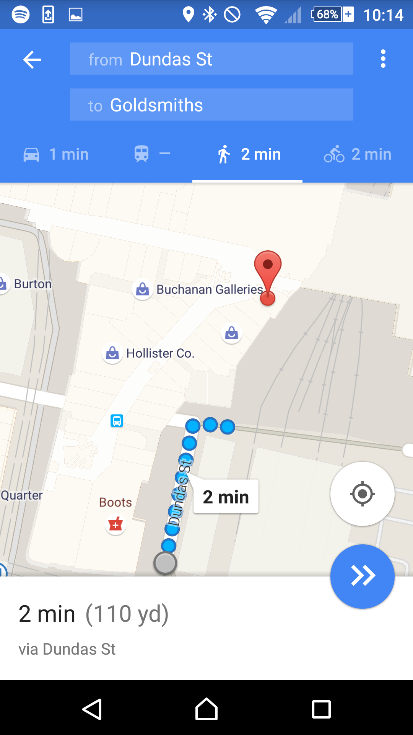
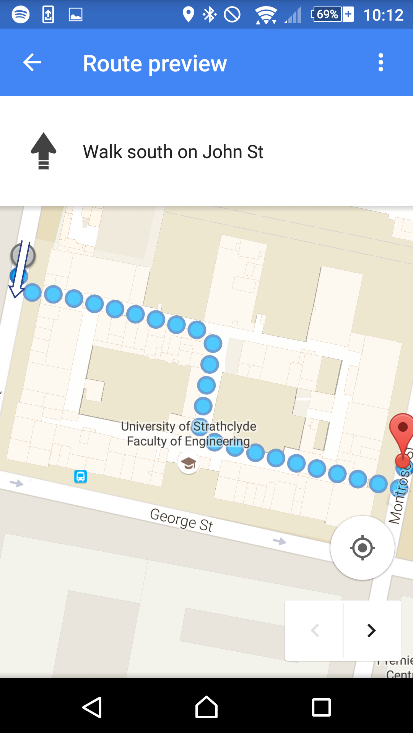
### 3.2.1 Standard Routing

There are many mobile applications which offer the user routes between a start and destination. With between one and five billion downloads, one of the most popular routing applications is Google Maps [10]. Google Maps currently offers routes between almost anywhere in the world, incorporating a variety of transport modes including: walking, driving or public transport. Despite Google Maps’ massive level of coverage, the level of detail within the maps is incredibly high. Many buildings even have internal floors that can be viewed.



###### Figure Examples of Google Maps Functionality

One limitation of Google Maps, is that the routing provided does not extend to the indoor portions of the maps. Despite occasionally giving very specific routes through buildings as options, Google does not allow routing to locations within a building. A sample of the indoor routing capabilities of Google Maps is shown in Figure 3.



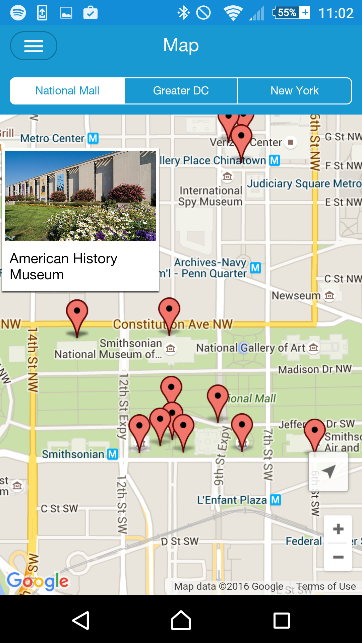
###### Figure Examples of Google Maps Indoor Routing Attempts

As can be seen from Figure 3, Google Maps can recognise where certain indoor destinations are. However, instead of routing the user to their destination, the application will direct the user to their nearest entrance to the overall structure. Despite only routing to the nearest entrance, Google must have some knowledge of how to navigate within buildings. The routes provided will occasionally use corridors within a building in order to reach the other side.

### 3.2.2 Institutional Routing

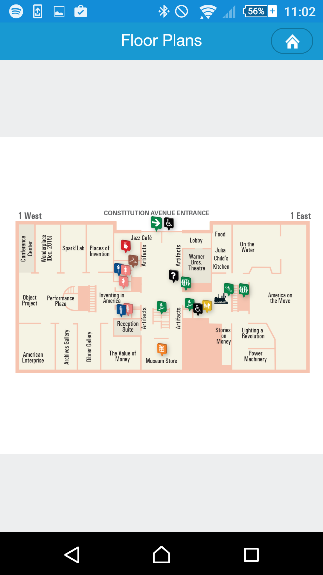
There appears to be no major competitors within Institutional Routing. The Google Play store was searched for mapping applications that could cover routing within different institutions including: Oxford University, Cambridge University, St. Andrews University, Dundee University, The Louvre or the Queen Elizabeth University Hospital. Many of the institutions that were searched are notoriously difficult to navigate within for visitors. However, none of the institutions searched for have widely publicised applications available to help visitors navigate the institution.

The closest possible competitors that could be found are the Smithsonian Institution official application [11] and a non-official University of Glasgow application [12]. The Smithsonian Institution’s application splits the provided maps into two areas within the app which they refer to as: Map and Floor Plan. The Map section shows the locations of the buildings on a map using markers, and allows the user to tap the markers to discover the selected building’s name. The outdoor mapping functionality provided can be seen in Figure 4.



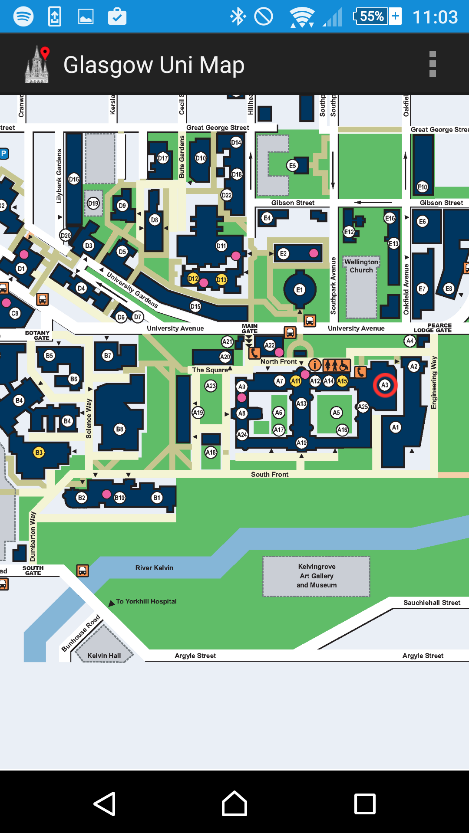
###### Figure Smithsonian Institution Map

The Floor Plan section of the Smithsonian Institution’s application provides a series of menus which the user can navigate to plan their own route through the institution. However, there is no routing or interactivity available on these maps. A sample of how the Floor Plan section appears is shown in Figure 5.



###### Figure Smithsonian Institution Floor Plan

The only other possible competitor is named Glasgow Uni Map. Glasgow Uni Map provides the ability for the user to type in the name of a building within the campus, and for this building to be highlighted on the campus map. There are no additional features other than simply circling the building that the user has typed in. The application has been downloaded between one thousand and five thousand times, such is the need for institutional routing. The highlighted map produced by the application can be viewed in Figure 6 with building A3 highlighted as the Thomson Building was selected on the previous screen.



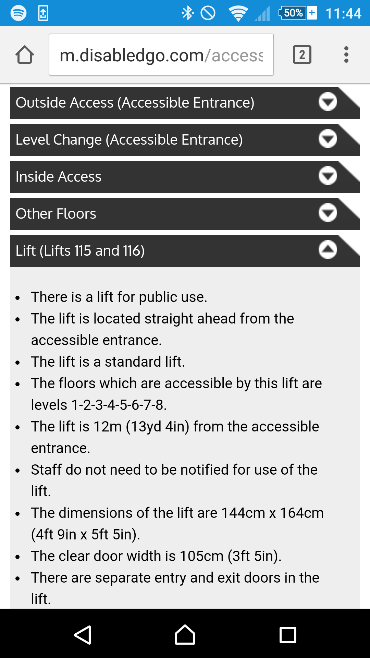
###### Figure Glasgow Uni Map Screenshot

### 3.2.3 Disability Improvements

By adding the ability to avoid stairs within the application, the first step towards making the application a major asset to any Disability Service has already been made. As will be discussed in Chapter 8.2.1 Disability Services the disability related features would be improved given more time. This means that the application developed during the project could potentially compete with the services currently used by Disability Services within institutions.

According to Colin Flynn, a Disability Advisor for the Disability Service at the University of Strathclyde, the main technological service available to mobility impaired students to help them plan a route around the University is DisabledGo. DisabledGo provides information on many institutions throughout the country. By navigating a number of menus information can be discovered about the access to buildings, floors and some rooms. The information provided details whether there is ramp available or not, the steepness of either the stairs or ramp and lighting levels for the area.

Although this survey may be very useful for students when deciding upon whether or not to join a university; The survey is clearly impractical for students to use the service to plan routes around the university. All the survey could feasibly be used for is allowing the student to determine whether or not a short section of a route is accessible or not. Possible alternatives are not made easily available within the survey. An example of how the survey information is provided can be viewed in Figure 7.



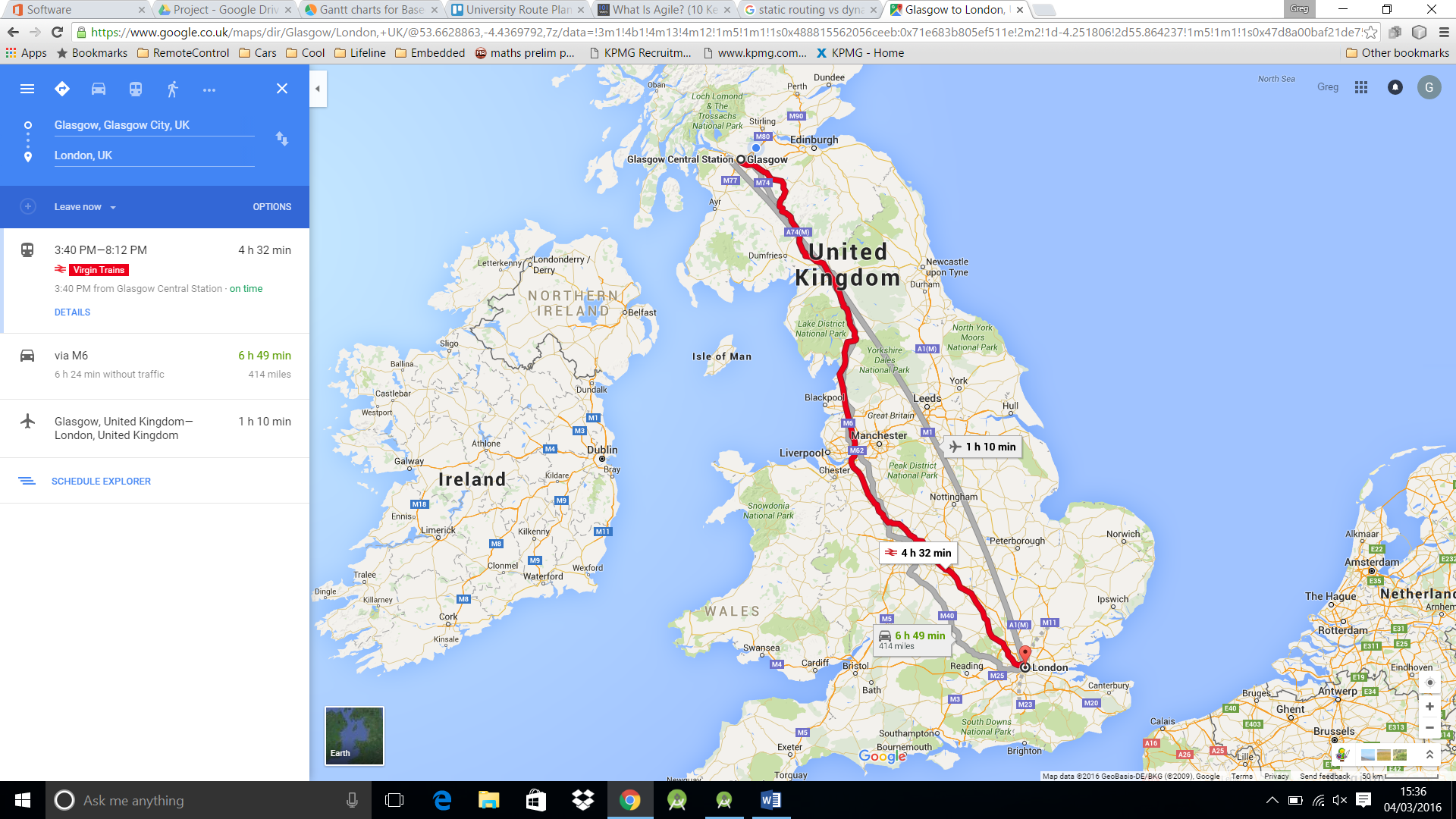
###### Figure Disabled Go Strathclyde University Colville Building Survey Section

## 3.3 Static or Dynamic Routing

To determine whether the application should provide static or dynamic routing what both involve should be understood.

### 3.3.1 Static Routing

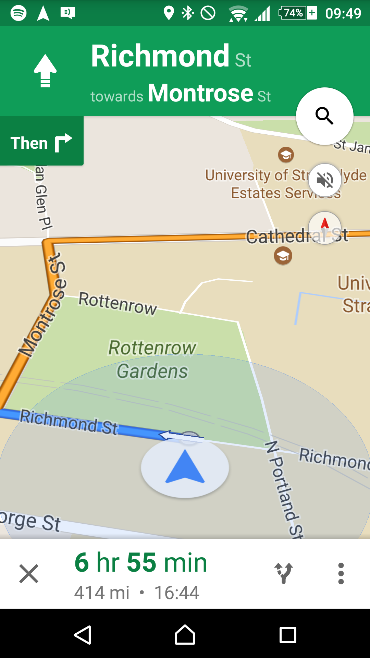
For the purposes of this report Static Routing will be defined as routing between two fixed points with the whole route available for the user to view. A popular example of this is Google Maps which can be found in Figure 8.



###### Figure Static Routing Example [13]

### 3.3.2 Dynamic Routing

By comparison Dynamic Routing will be defined as routing from the user’s changing location to a fixed endpoint. Using the user’s location as the start, means as the user moves the route will automatically adjust to their new location. Dynamic Routing is used in Satellite Navigation systems or Google Navigation which can be viewed in Figure 9.



###### Figure Dynamic Routing Example

With Dynamic Routing the view is also commonly more restricted, showing only a short section of the route at a time i.e. where the user is to go next.

## 3.4 Location Services

The main difference between the two types of routing discussed, is whether the user’s location is utilised to improve the user interface or not. In order to decide between them then, deciding whether using location services is feasible becomes important.

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

### 3.4.1 Global Navigation Satellite System (GNSS)

There are several GNSSs 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 [14]. The GNSSs were devised as a way to enable users worldwide to calculate their current location. The GPS system for example, comprises of twenty-four satellites continuously producing signals. The signals produced tell a user where the satellite is and provide highly accurate timing information about when the signal was produced. By listening to the signals generated by at least four of the satellites, and hence discovering their locations, the user’s location can be calculated. The user’s location is found by calculating the distance from each satellite, using the time taken to receive the signal, and combining that with the knowledge of the exact location of the satellites [15].

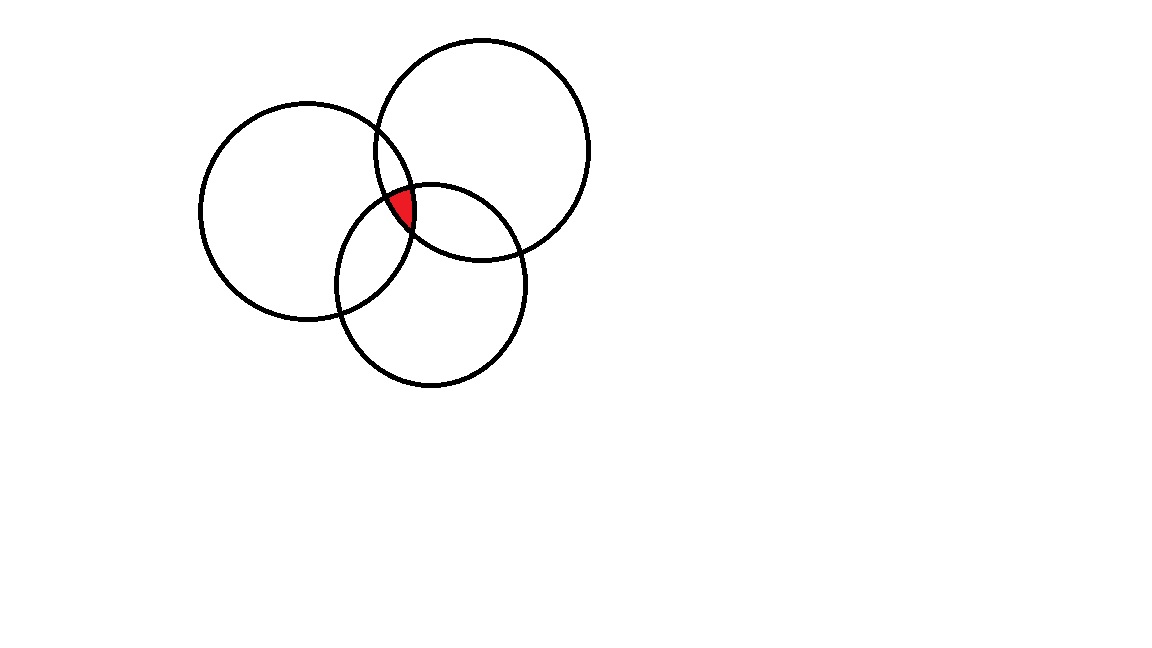
The accuracy of GPS is roughly 3.5m on the horizontal plane without using additional information. There are other systems which can be used in conjunction with GPS to provide up to millimetre accuracy. However, 3.5m should be accurate enough for this project [16]. The accuracy of Galileo is expected to improve to the centimetre range [17] once the system is completed in 2020 [18].

There are two main limiting factors: 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 [19]. Multipath is when the signal may reach the receiver via many paths. An example of multipath would be direct from the satellite compared to being reflected by a building in between the satellite and receiver. The weak multipath suppression available on mobile phone antennae means that the signal will degrade when near buildings. 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, the signal can be 10-100000 times weaker when indoors [20]. The reduction in the signal to noise ratio can make receiving the necessary information from four satellites at once particularly difficult. Without information from multiple satellites, determining location indoors via this method becomes virtually impossible.

### 3.4.2 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 the response takes to be received. 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 [21]. An example of how this works can be seen below in Figure 10.

###### Figure Cell Tower Triangulation



1

3

2

In Figure 10, the arrows represent the calculated distance from the cell tower. With only one cell tower response, the user could be anywhere in the surrounding circle. The user must be within the surrounding circle, as the furthest they could possibly be from the tower, with that response time, would be a straight line from the cell tower (the radius of the circle). However, there is no guarantee that the path being taken is a straight line. Instead the signal 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 10 with these responses the user could only be in the overlapping area from each circle.

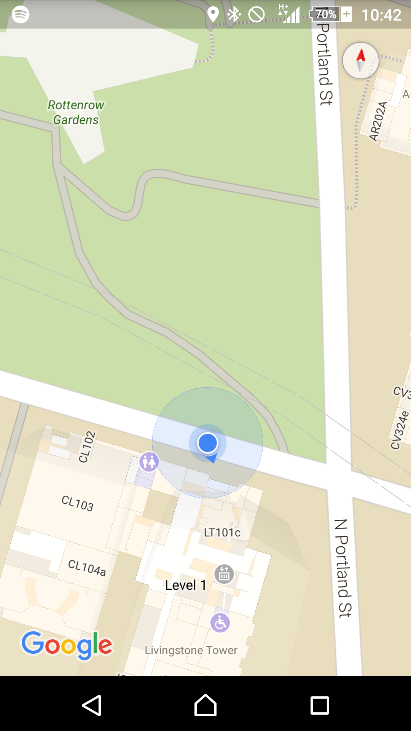
Cell Tower Triangulation could potentially be a more reliable method of geolocation, as the calculated location still tends to be as accurate while indoors, presuming there is a mobile phone signal. Cell Tower Triangulation even works better in more urban areas where this application would be expected to be used, due to the higher number of cell towers [21]. There are some drawbacks in that to perform the triangulation, an accurate knowledge of cell tower locations is required. Cell Tower locations are not typically available publically making using Cell Tower Triangulation more difficult [22]. On average for example in the United States of America (USA) the average accuracy of a three tower triangulation is 0.75 square miles [23]. The accuracy of Cell Tower Triangulation is not at an acceptable level for this project.

### 3.4.3 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 and their locations are known, then the user’s location can be calculated using triangulation.

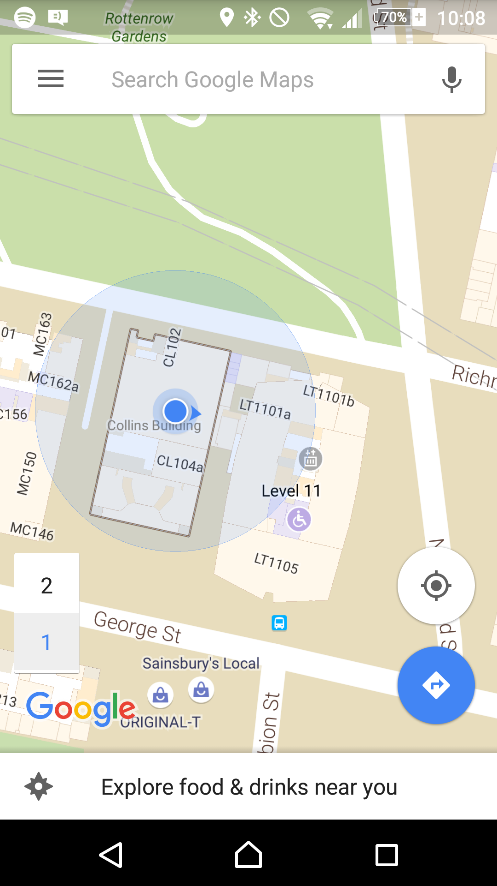
### 3.4.4 Current Android Solution

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. The location provided by the Android OS currently uses a combination of all 3 of the location services previously mentioned [24]. The accuracy while outside is reasonably high as can be seen in Figure 11.



###### Figure Outdoor Location Example

In Figure 11 the reported position is represented by a small blue circle. The small blue circle matches perfectly with the actual position of the user at that time. The accuracy of the reported location is likely due to the availability of a strong GPS signal, in combination with cell tower and Wi-Fi location services. When inside, the provided location services are far weaker as can be seen in Figure 12.



###### Figure Indoor Location Example

In Figure 12, the reported location is again shown with a small blue circle. The reported location 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 the project. The area of uncertainty shown in Figure 12, covers an entire building as well as significant amounts of two neighbouring buildings. The large area of uncertainty means the provided location services are unlikely to be useful when trying to navigate within one building. The inaccuracy and uncertainty within the location is most likely due to the lack of a reliable GPS signal, combined with a low knowledge of the exact Wi-Fi and Cell Tower locations.

The reported position should be improving as Google is trying to advance their knowledge of Wi-Fi locations using their massive user base. The improvement is being carried out 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 [25]. The improved knowledge of Wi-Fi base station locations could eventually provide strong indoor locations worldwide. However, the currently available location is far to inaccurate for use in this project.

### 3.4.5 Improved Suggestion

As previously stated, the main problem currently facing indoor location services is a lack of knowledge about the locations of different Wi-Fi sources. Although steps are being made to try to improve the knowledge of Wi-Fi locations globally, within a smaller area that is to be mapped, such as a University campus, the problem becomes far more manageable. Within the smaller area the locations could be found manually, preferably by using documentation written during the installation of a Wi-Fi network. However, if an existing Wi-Fi map is not available, a new map could potentially be built by creating another application to track Wi-Fi locations.

## 3.5 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. The shortest path in the graph can then be used to indicate the route that the user should follow.

### 3.5.1 Representing a Graph

There are three main ways of representing a graph within a computer system: Edge List, Adjacency Matrix and Adjacency List [26].

#### 3.5.1.1 Edge List

An Edge List is a list of all of the edges in the graph. The Edge List is a fairly easy to understand representation of a graph. However, the Edge List representation can be difficult to traverse during path finding algorithms. Indexing the information by edges rather than nodes, can make moving from one node to another more difficult. Each time the algorithm needs to check what edges are connected to the current node, all of the edges must be cycled through checking if they are attached to that node.

#### 3.5.1.2 Adjacency Matrix

An 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 13.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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 |

4

C

6

5

D

1

A

B

2

2

E

###### Figure Example Graph and Corresponding Adjacency Matrix

Adjacency Matrices are typically used in very densely populated graphs i.e. those with a lot of edges per node. Dense graphs are more appropriate for an Adjacency Matrix as the number of edges does not affect the storage space required. If the graph is particularly sparse, with each node having far fewer edges than there are other nodes, the Adjacency Matrix becomes increasingly inefficient. In the sparsest graphs almost the entire matrix is being filled with null values to represent no edge.

#### 3.5.1.3 Adjacency List

The Adjacency List is a method of storing the graph, in which a list of connected nodes is maintained for each node. Storing the data as an Adjacency List makes routing algorithms far simpler, as determining which nodes are connected to the node that is currently being examined becomes easier. Naturally, an Adjacency List does not support edge weights. However, support for edge weights can be added via a slight modification. By storing a list of edges for each node, instead of just the connected nodes, all of the benefits of an Adjacency List are available. However, storing edges instead of just the nodes, allows weights to be added to the edges. Allowing edge weights means that a weighted graph can be represented.

### 3.5.2 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.

#### 3.5.2.1 Breadth-First Search

The Breadth-First algorithm starts at one of the two nodes the algorithm is to find the route between. From there the algorithm then moves out to all of the surrounding nodes and assigns them a value of one. Each of the nodes with a value of one are then examined and all of the nodes surrounding these nodes are assigned a value of two. The process continues until the algorithm reaches the desired node. The Breadth-First algorithm is guaranteed to find the path between the two selected nodes with the fewest number of hops. However, the Breadth-First algorithm, does not take into account edge weights.

By not taking into account differing edge weights, using the Breadth-First algorithm makes constructing a graph representing a campus to be mapped far more difficult. Without edge weights, each of the nodes would have to take roughly the same amount of time to walk between to produce an acceptable route. Making the nodes all take the same amount of time to walk between becomes particularly difficult when adding lifts to the routes. Waiting on lifts could potentially take several minutes, meaning each node would have to be several minutes apart. Making each node several minutes of walking apart is likely not possible within corridors with many turns. The only alternative would be to add conceptual nodes which the user could move through while waiting on lifts. However, the addition of these conceptual nodes is essentially adding weights in a very inefficient way.

#### 3.5.2.2 Dijkstra’s Algorithm

The main problem with the Breadth-First algorithm is that edge weights are not supported. Dijkstra’s Algorithm by comparison, will always find the shortest path in a weighted graph.

The algorithm works by starting at one of the two nodes that the path to be found is between, and marking the chosen node as examined. Each of the nodes that the examined node is connected to, then have a number associated with them, representing the weight of the edge taken to reach them. From there the node with the lowest associated number is marked as examined. All of the nodes connected to the newly examined node have the numbers associated with them updated. The numbers are updated to be the sum of the weight to the examined node plus the weight of the new edge. If this new associated number is higher than a previously associated number, the previous number is used. The process continues until the destination node is marked as examined at which point the shortest path between them has been discovered.

Dijkstra’s Algorithm is particularly useful as the time complexity is linear. Linear time complexity means that as the number of nodes in the graph increases, the time taken to find the path also increases linearly. Other path finding algorithms can have far higher time complexity than linear.

#### 3.5.2.3 A\* Algorithm

The A\* Algorithm is essentially a modification of Dijkstra’s algorithm that allows the shortest path to be found more quickly. The decrease in time taken to find the path, is achieved by adding a heuristic value to the comparison when deciding which node to examine next. An example of a possible heuristic value for a mapping application would be the straight line distance between each node and the current destination. Using the heuristic value means that when choosing which node to examine next, the algorithm would consider both the difficulty in reaching that node and how physically far away from the destination the node is. The result of using the heuristic value is that instead of spreading out equally in all directions like in breadth-first, or towards the easiest direction like in Dijkstra’s, the application 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 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 the one considered in the project, would face difficulties in implementing such a system. Calculating the straight line distance 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. Using a straight line distance as the heuristic value would lead the search algorithm to move towards the destination node. However, in a multi-level system the quickest route may involve moving in the opposite direction, towards a staircase or lift. The A\* Algorithm would still find the quickest route, but in many cases would likely be less efficient than Dijkstra’s Algorithm.

## 3.6 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. Using good software engineering practices, should aid development and make testing easier by ensuring the software is flexible and forgiving of change. The various styles and patterns used in the project are as defined in the class CS409 Software Architecture and Design at Strathclyde University.

### 3.6.1 Architectural Style

An Architectural Style is the highest level programming choice to be made. The choice of Architectural Style, 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 separated 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 the implication is that the programming language to be used is Java. As Java is best suited to the Object-Oriented architectural style, it is this 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.

#### 3.6.1.1 Object-Oriented

The Object-Oriented (OO) 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. Such an approach 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 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 such a change 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.

#### 3.6.1.2 Event-Based Implicit Invocation

Event Based Implicit Invocation is similar to the OO 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.

### 3.6.2 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 which can be adopted. However, since re-usability was a major factor the decision was made that the Model-View-Controller (MVC) approach would be used.

Model-View-Controller divides data and functionality into three main sections: Model, View and controller. 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, a map or simply a list of instructions. Importantly outside the defined interface for communicating, these components do not rely on each other. An implication is 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 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 highly interchangeable. An important point in case at some point during development a drastic change is required. A possible reason 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.

### 3.6.3 Design Patterns

Throughout programming, certain problems such as how to change algorithms during runtime appear again and again. Design Patterns are essentially proven solutions to commonly recurring problems. The use of design patterns provides the means to deal efficiently with such problems. Below is an overview of each of the design patterns which have been used throughout the project.

#### 3.6.3.1 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, one component allows others implementing a specific interface to register for updates. Whenever this first object recognises a change of some sort 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 14.

###### Figure Observer Design Pattern UML

Observer

*Update()*

**Update(State)**

Concrete Subject

*getState()*

Subject

addObserver(Observer)

notify()

Concrete Observer

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 debugging potential problems may become 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 14 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.

#### 3.6.3.2 Adapter Pattern

The Adapter 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. The pattern 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. 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 the Adapter Pattern can be found in Figure 15.

Client

Target

Adapter

Adaptee

###### Figure UML Representation of Adapter Pattern

In the diagram shown in Figure 15 the client needs to use the specific interface, Target. However, the necessary function is actually within Adaptee, which cannot implement the Target 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.

#### 3.6.3.3 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 change. Using the Template Pattern allows much of the outline and other functionality of an object to be defined in one class and for a certain part of the functionality to only be defined in a subclass. The Template Pattern can be found in Figure 16 UML Representation of Template Pattern.

Abstract Class

templateMethod()

fixedMethod1()

variableMethod()

fixedMethod2()

Concrete Class

variableMethod()

###### Figure UML Representation of Template Pattern

In Figure 16 the abstract class would implement the template method and both fixed methods. The abstract class would then leave the variable method to be implemented by the subclass, or potentially subclasses. The template method would make calls to the fixed and variable methods in the order required, knowing that the concrete class will provide the variable methods. The concrete class then implements the variable methods, allowing the class to perform the necessary duties. Using the Template Pattern gives the benefit of allowing multiple classes, that differ in one area of their implementation, to be created quickly. For instance, in a drawing package the pattern 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 drawing, such as checking there was adequate space and alerting a listener to the change. The main disadvantage is that the functionality cannot be changed at runtime, once the class is created it will always perform that action in the same way.

#### 3.6.3.4 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. Figure 17 shows a representation of the Strategy Pattern.

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 implementing the Strategy interface. The Context can then change what functionality to use by changing which class it calls the method on. Frequently the context will have a method allowing another class to set the strategy. Using the Strategy Pattern means that the context doesn’t need any knowledge of the underlying implementation. As the context has no knowledge of the underlying strategies, new strategies can be implemented and used easily.

#### 3.6.3.4 Composite Pattern

The Composite Pattern provides the ability to treat a group of objects the same as an individual object. Being able to treat an individual object or a group of objects the same way is often beneficial within programming. An instance would be an application which controls a user’s lights. Grouping lights in an area, say a room, and being able to use functionality available for one light bulb (i.e. turn on, turn off or set timer), with the group of lightbulbs in a room would be useful. A representation of the Composite Pattern can be found in Figure 18.

###### Figure UML Representation of Composite Pattern

 Client

 Leaf

 Composite

**Add(Component)**

**Remove(Component)**

 Component

Operation()

*Add(Component)*

*Remove (Component)*

As can be seen in Figure 18, the Composite Pattern has both the group of objects, the composite, and the individual object, the leaf, implementing the same interface, Component. The shared interface allows the client to interact with either the group or individual object in the same way. There are two variations of the Composite Pattern: safe and transparent. The methods only available in the safe variation are shown in **blue,** while those only available in the transparent variation are shown in *red*. Both variations have their disadvantages, the safe variation means that to add or remove elements within the composite, the client needs to directly communicate with the composite and not through the interface. Whereas the transparent variation, means that the “leaf” object will need to provide implementations for add and remove component, which is illogical. A light bulb can’t be added to a light bulb it has to be added to a group of light bulbs.

# 4. Design

With a software project of the scale required in a mapping application, a good software design is vital to ensure maintenance is possible. A strong design results in a program which can be easily understood by an experienced programmer. If the design can be understood easily extending the project at a later date becomes far simpler.

## 4.1 Initial High-Level Design

As previously discussed in Chapter 3.6.2 Programming Style, the high level design should follow the Model-View-Controller (MVC) programming style. To follow the MVC style successfully exactly what each subsystem will be responsible for and how they will communicate must be outlined. For a full overview the high level UML diagram is shown in Appendix 4 Initial High Level UML Diagram.

### 4.1.1 Responsibilities

The model will deal with the state of the program i.e. what the currently selected start and end nodes are, what errors have occurred and what the current route is. The model 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 the controller is informed of some form of user input from the view, deciding in which way the user input will affect the model is the responsibility of the controller.

### 4.1.2 Communications

The communications between each of the three subsystems can be found in Table 2.

###### Table High Level Communications

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

Using the Observer pattern to communicate between the model and view keeps these two sub-systems independent from one another. The controller to view and controller to model communications will pass through interfaces. Using interfaces will aid decoupling. However, using the observer pattern does mean the model and view will be the most decoupled sub-systems. The difference in the levels of decoupling comes from the difference in how data is passed between the sub-systems. The model simply provides all of the data the view may require and allows the view to react accordingly. By comparison with the other sub-system interactions one system will pass specific data through with the instruction to perform some action. Not using the Observer pattern throughout 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 easily.

## 4.2 View Design

The first issue that must be addressed when designing how the view should function is understanding exactly how the application will display the necessary information and interact with the user.

### 4.2.1 Displaying Necessary Information to The User

A decision was made early in the project that from an aesthetic point of view, showing the route on top of a pre-existing map would be preferable to creating a new map. Using a pre-existing map, 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 the necessary functionality [8] [9]. As the application was being developed for Android OS the users would likely be familiar with Google Maps, which 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 over the map. Google 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.

The decision was made that showing the entire map at once would be 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 at one time. One plane would be either 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, outside these could be a building or on a floor of a building these would typically be rooms. Midpoints are points that would be needed to follow the route, examples of which include entrances to rooms and points where corridors join. Midpoints are only useful to the user when they are on the route they are following. As a result of the relative usefulness of midpoints and endpoints, the decision was made that Endpoints would always be displayed when on the correct plane. However, midpoints would only be displayed as part of a requested route.

As discussed earlier, 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. Polygons are used within the application to display the Endpoints to the user. Polylines are essentially lines that can be drawn between two points on the map in a variety of colours. The decision was made that these would be drawn between the relevant midpoints to display the route to the user.

### 4.2.2 User Interactions

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 nodes, the decision was made that a variety of options should be available to the user. One of the options provided to users is selection of the start and destination by tapping upon the desired endpoints. Using Polygons to display the endpoints meant that a listener could be added to the map to discover when endpoints are clicked by the user. Another option available to allow users to select an endpoint, is typing the name of the endpoint into one of the two provided text boxes. As the user begins to type, options will appear below which can be chosen from. Once the user has selected an endpoint, alerting the user that the selection has been successful is important. Two methods were employed to alert the user to a successful selection: changing the colour of the selected endpoints and ensuring the two textboxes display the names of the selected endpoints.

Once the endpoints have been selected starting routes is relatively simple. The only functionality required is a button which can be clicked in order to display the new route.

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 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. While outside the user requires a way to alter the view to display the interior of a building. To allow the user to move the view indoors an “In/Out” button was added. The new button is 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.

Displaying the interior of a building 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, as Google only allows enabling all floor plans, the user could enter one building and then simply swipe to the side to view the inside of another building. Allowing the user to move between buildings, without passing through the outside plane, meant the application could not guarantee which building the user was viewing. Without a guarantee as to which building was being viewed, the application could not be relied upon to draw the correct additions to the map. Although Google does provide a listener to alert the application when a new building is focused, the information available from the listener is not sufficient to determine which building is being viewed.

To ensure the application could guarantee which building was being viewed, the decision was made that the user should not be allowed to view areas outside the selected building. Stopping the user moving outside the selected area is the responsibility of the CameraLimiter class. The CameraLimiter questions the IController to determine which building the user is currently viewing. If the user tries to move the camera to another area, the CameraLimiter will force the camera to pan back to the correct area.

When viewing internal planes, the user must be allowed to select which floor they would like to view. Google provides a level selector toolbar to allow the user to move between floors. However, Google does not provide a direct listener for the level selector UI element. Fortunately, when a user selects a new floor an IndoorLevelActivated event is created, for which a listener is available. Unfortunately, the IndoorLevelActivated 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. The additional times the event is created, means that the listener needs the ability to be turned off in certain circumstances. Turning off the listener when moving between planes, stops the view forcing the controller to try to move to a variety of other floors.

Following routes using the standard method of moving into a building, then changing levels was deemed insufficient for following a route. In order to improve navigation while following a route, 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 the Marker. The decision was made that these markers would be added at the first and last midpoints on the route for the plane being viewed. The listener was then used to allow the application to move to the next relevant plane when the Marker was selected by the user.

As discussed earlier, in order to react to user interactions a variety of Google Maps interfaces needed to be implemented. As reacting to user interactions is the Controller’s responsibility, 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. Passing View specific components such as Polygons, would create an inappropriate amount of coupling between the View and Controller. If the Controller relied on Google Maps specific objects, changing the View in any serious way without dramatically changing the Controller would become impossible.

In order to reduce the level of coupling between the View and Controller, the Adapter Pattern was used. For each of the Google Maps specific interfaces, a corresponding class was developed. The corresponding class converts the View specific data into a form which can be understood by the Controller. The usage of the Adapter Pattern can be seen in Appendix 5 View UML Diagram, where IController is the Adaptee and the View (or more specifically the Google Maps components) are the Client.

## 4.3 Model Design

The model is responsible for storing the current state of the program and providing the route requested by the user. Storing the current state of the program is fairly simple and can be maintained using a series of set methods.

In order to provide the route, the graph must first be stored in a sensible manor. Recall from Chapter 3.5.1 Representing a Graph, there are three main ways to represent a graph. However, the Edge List is not appropriate as the graph will frequently be traversed by a routing algorithm, and the Adjacency Matrix representation is also inappropriate as the graph is particularly sparse. By eliminating the other two possible representations the Adjacency List variation covered in Chapter 3.5.1.3 Adjacency List becomes the obvious option.

As discussed in Chapter 3.5.2 Path Finding, there are many possible ways in which to find the shortest path between the nodes. However, during finding the shortest path there are steps which must be taken regardless of the path-finding algorithm used: checking the start and end nodes are valid, and alerting the listeners to the new route. As only a certain section of the functionality needs to be changed, and changing this functionality at runtime is unlikely to be useful, the decision was made that the Template Pattern would be used to provide the routing service. The usage of the Template Pattern can be found in Appendix 6 Model UML Diagram, where the Abstract Class is Model and the Concrete Class is DijkstrasModel.

Dijkstra’s Algorithm was chosen instead of the Breadth-First Search or A\* Algorithm as both have severe limitations. The Breadth-First Search was not chosen as the graph would require weighted edges which is not supported within the Breadth First Search. The A\* Algorithm was not used as doing so would require more development time, despite giving similar results to those produced using Dijkstra’s Algorithm.

To allow the graph to have elements that cannot be passed through - i.e. the concept of a building as opposed to actual corridors within that building. There needs to be a check placed on each node to ensure the node can be used. Adding checks upon whether or not a node can be used fits well within Dijkstra’s algorithm. The check can be added to only update the weights of surrounding nodes, if the node being examined passes the provided check.

Adding checks to the route finding algorithm, could be used to incorporate some form of disability aid into the application. To provide disability aid, further checks were added to exclude certain nodes when options were enabled. As a proof of concept for a much larger expansion, checks were developed allowing the user to choose whether or not they wished to avoid stairs, lifts or neither. As allowing the checks to be changed at runtime is important the Strategy design pattern was used. The usage of the Strategy Pattern can be found in Appendix 6 Model UML Diagram, with the Model being the context and the NodeCheck interface being the Strategy. The Concrete Strategies are hidden within the Controller for reasons which will become apparent during the next section.

## 4.4 Controller Design

The Controller’s main function is to add meaning to the user input. An instance of the Controller adding meaning to user input within the project, is when the user types in the first text box. Once the user has entered a new value, the Controller’s responsibility is to understand that the new value means the starting position has changed and to update the model accordingly. To add meaning to most of the user actions is fairly simple, and is just a matter of translating the action into an appropriate method call for the Model. However, two user interactions require more complex thought: moving planes and changing the routing modifiers.

The user can move planes in many ways including: moving level using the level selector, clicking the In/Out button or starting a route. As Google keeps track of what level was last viewed for a building, maintaining a record of which floor was last viewed for each structure is important. If a record of what floor was last viewed for each building was not maintained, when a user returned to a structure the wrong floor may be drawn. In order to maintain records for each building, the decision was made to create a collection of objects known as structures. There is a structure object created for each building which contains basic information, such as the name of that building, the last viewed level and the location of the building. The View can request the location data, via the CameraLimiter object, to ensure the app stays focused on the correct location. When an event requires a change in structure, such as moving building, the current structure variable is updated to indicate the new structure. Changing levels within a building can be handled by setting the level variable within that structure.

The Model currently deals with Route Modifiers via the NodeCheck interface. However, the user is to be allowed to choose multiple checks to be applied. There are two possible ways in which to allow the user to select multiple checks. One method, involves developing a separate concrete class for every possible combination of checks. The alternative is to treat the check as a group of checks. Developing a new class for every possible combination of checks would make extending the system far more difficult, as each new check would involve implementing exponentially more classes. The difficulty involved in extending a system in which each combination has a concrete class, implies the more elegant solution would be to treat the check as a group of checks. In order to allow the Model to continue to treat the process as one check, while allowing the Controller to treat the process as a group of checks, the Composite Pattern was used. The usage of the Composite Pattern can be found in Appendix 7 Controller UML Diagram with the Controller being the Client, the NodeCheck interface being the component, CompositeCheck being the Composite and BasicCheck, StairCheck and LiftCheck being the leaves.

The Composite pattern was used in the safe variation, which means that the Model needs no knowledge that the Composite Pattern is being used at all. The Controller handles all of the composition and creation, then simply passes the Model a check to use.

## 4.6 Data Requirements

The main problem with the design outlined so far is the massive amount of data required. To use the design as given above, requires three major groups of data: Endpoint Data, Midpoint Data and Link Data. Although none of the data is difficult to obtain, compiling the data into a useful form is time consuming.

To display endpoints in the way outlined in Chapter 4.2 Initial View Design, would require a list of latitudes and longitudes to draw the polygons. The list of co-ordinates would then have to be paired with the name of the room/building and the plane that it should be displayed on. Although for one endpoint a large amount of information is not required, the overall information required for endpoints does scale up significantly with the area being mapped. For example, the three floors and small subset of buildings used in the proof of concept data requires almost fifty endpoints.

The precise co-ordinates of the rooms and buildings can be easily acquired using the Google Maps web interface. The latitude and longitude of an exact point on Google Maps can be obtained by clicking on the point required and then copying the co-ordinates into a text file. However, using the Google Maps web interface to find co-ordinates is particularly time consuming and can take up to an hour to outline only six endpoints. Given that to map the entirety of the John Anderson Campus at Strathclyde University would require hundreds of endpoints, the length of time taken to outline endpoints using this method is concerning. In order to efficiently create the Endpoint Data a second application was created. The new application allows the user to enter the name and plane of an endpoint using text boxes. Once the name and plane have been entered, the user can then draw a shape on top of Google Maps to create a new endpoint. The application, named Endpoint Builder, was used to construct the necessary endpoints to represent the exterior of several buildings within the John Anderson Campus, and a subset of the floors within these buildings.

As discussed earlier Midpoint Data is also vitally important to the success of the application. The Midpoint Data represents the points between which the route should be drawn. An instance of a midpoint could be where multiple corridors meet. Midpoints require similar data to endpoints, in that they need a name and plane on which they can be viewed. However, instead of requiring a list of co-ordinates, midpoints require only one set of co-ordinates. Creating the necessary Midpoint Data faces similar problems to creating the Endpoint Data. Even though each individual midpoint requires less co-ordinates to be drawn than an endpoint, there are far more midpoints to be drawn. In the subset of the John Anderson Campus created to form a proof of concept, over 150 midpoints were required.

In order to create the Midpoint Data for the proof of concept, another application was developed named Midpoint Builder. Midpoint Builder allows the user to enter a name and plane for a new midpoint, then place it on top of Google Maps using a tap. As Midpoints are easily placed incorrectly, Midpoint Builder also gives the user the ability to move and rename newly placed midpoints. Once a new set of midpoints has been finalised, they are saved to a text file which can be added to the main application.

The final set of required data is Link Data, which defines the midpoints and endpoints that are connected. Link Data also defines the weight associated with each connection, to allow the routing algorithm to find the shortest path. The Link Data could have been created manually however in the small subset of the John Anderson Campus mapped so far over well over 200 links have already been required. In order to efficiently create this data a fourth application was developed. This new application is very similar to the main application in that it allows full navigation throughout the map. However, it has three textboxes instead of two. The first two display the currently selected mid/endpoints and the third allows the user to enter a weight for the new edge between them. The new edge can then be added by pressing a button. Once a user selects a first mid/endpoint all of the edges from it are drawn onto the map. Originally the idea was to manually enter the edge weights based on how long it took to walk between them however it quickly became apparent this was going to take too long. Instead an algorithm was developed for working out roughly how long it would take to walk that distance.

To calculate the length of time it would take to walk in a straight line between two points two pieces of information were required: a distance in latitude and longitude and the length of time taken to walk that distance. A straight line distance of 443ft was measured using Google Maps. This was between the co-ordinates (55.861574, -4.245258) and (55.861330, -4.243139). Using Pythagoras’ Theorem this gives a straight line distance of 0.0021330018752922. The average walking speed of the expected user is roughly 4.95ft/sec according to the Road Engineering Journal [27]. This means the time taken to walk 443ft would be 443/4.95 = 89.49 seconds. This means that if the time taken is divided by the distance it will give a number that can be used to convert from distance to time.

When two points are selected in the Link Builder application the estimated weight is given in a text box however the user can edit this allowing links between floors such as using stairs to be put in manually.

# 5 Testing Strategies

As discussed during Chapter 2.2.4 Testing, the decision was made that contrary to traditional wisdom the full system testing would be performed during development with Unit and Sub-System testing coming later. The decision to invert the traditional order of testing was made for several reasons. However, the main reason was that unit testing and sub-system testing struggle with ensuring the view displays the necessary information in the correct manor. As the view was the first sub-system to be constructed it was decided that during development the application would be regularly loaded onto an android device to ensure it was displaying correctly. Once the view had been built the model and controller could easily be tested using full-system testing and the already functional view. Once the project had been completed, unit and sub-system testing was then used as a means of providing deeper assurances that each element of the system worked as expected.

## 5.1 Full System Testing

The tests that were performed for full system testing evolved as the project continued and features were added. In order to fully test the application text files were developed for a subset of the university. The sub-set used for testing included the second floor of the Royal College Building, the first and eleventh floors of Livingstone Tower and the outside of six university buildings. As the project neared the end, several tests would be performed each time a change was made. These tests were performed in order to ensure the change hadn’t had an adverse effect on another area of the program. The tests performed after each change were:

1. Selecting an endpoint by tapping it.
2. Moving inside a building using the In/Out button.
3. Changing level inside a building.
4. Moving outside a building using the In/Out button.
5. Ensuring the buildings maintain the last viewed floor.
6. Ensuring building views are limited.
7. Selecting an endpoint using text entry.
8. Requesting a route that must cross several planes.
9. Routes between planes can be followed by tapping the markers.
10. Each of the route modifiers change the routes as expected.

A deeper description and the results of each of the above test can be found in Appendix 8 Full System Test Results.

## 5.2 Unit Testing

Unit testing was used to ensure each individual component worked as it should. To perform the Unit testing two frameworks were used: Junit and Mockito. Junit is a testing framework used to create classes within which unit tests can be developed. The unit tests each create instances of the class that is being tested. Methods can then be called on this class and the results tested to ensure they act as is expected. Each of the tests return either pass or fail and can be ran automatically to ensure no changes to the code break any of the functionality.

When building these classes to test however, other classes need to be provided in order to create the classes. If these other classes do not work as expected, then the test could fail even if the class being tested does work as expected. Mockito, a mocking framework, provides the opportunity to create mock classes. Mock classes can be used to return the values required and any method calls on the mock classes can be verified. By verifying method calls all external interactions from the class can be monitored and the class can be monitored fully.

During creation of the unit tests for each component it was discovered, as it was expected to be, significant portions of the view couldn’t be tested fully in this way. Several classes could not be tested as certain view components, Markers and Polygons for example, were marked final and so could not be mocked. Several other methods within all areas of the program could not be tested individually an example of which could be set and get methods are typically tested together.

The unit tests can be found with the rest of the code at: <https://github.com/greglaw95/UniversityRoutePlanner>

## 5.3 Sub-System Testing

Sub-System Testing is essentially a larger version of unit testing. The main difference is that instead of testing one class a group of classes is tested. The Sub-System that was tested within this project was the combination of the model and controller.

The code can again be found at:

<https://github.com/greglaw95/UniversityRoutePlanner>

# 6 Performance Analysis & Future Proofing

If the application is to work over an entire campus or similar area, it must be shown that it is likely to continue to function at an acceptable level when the map size increases. There are two main areas which have been identified as potential problems as the map size increases: the routing algorithm and drawing the map additions.

## 6.1 Routing Performance

As the map size increases it will likely

Full Graph

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Description | No. of Endpoints | No. of Midpoints | No. of Links | No. of Planes | Average Routing Time (ms) |
| Full Graph | 45 | 164 | 240 | 4 | 7 |
| Minus RC2 | 25 | 95 | 153 | 3 | 5 |
| Minus LT | 26 | 98 | 120 | 2 | 3 |

CONSIDER SMALL TEST WHICH FINDS ALL OF THE POSSIBLE ROUTES FOR EVERY NODE THEN REPORTS THE AVERAGE TIME TAKEN TO FIND THE ROUTE. RUN THIS FOR WHOLE MAP THEN MAYBE TRY TAKING OUT A LEVEL OR TWO SEE WHAT HAPPENS TO AVERAGE ROUTING TIME

Could potentially take over a second. Probably ok much bigger campus though you might want to add some kind of waiting thing to show the user it’s working.

CHECKS??

## 6.2 Drawing Performance

At first glance it would seem as if when the number of endpoints increases there could be an impact on the length of time it takes to display the map. However, as the view depends on the concept of planes as discussed earlier only one plane will ever be shown at a time. This means regardless of how big the overall map becomes in terms of drawing times it will only ever depend on the maximum number of endpoints per plane. Due to the nature of the application this is going to be a relatively low number, presumably either the number of buildings in the campus or the number of rooms on a floor. As a precaution a test was developed to find out how long it would take to draw a varying number of endpoints. The results of this test can be seen in

|  |  |
| --- | --- |
| Number of Endpoints | Time taken to draw (ms) |
| 10 | 25 |
| 100 | 5 |
| 1000 | 3 |
| 10000 | 2 |
| 100000 | 3 |
| 1000000 |  |

# 7 Future Work

Although in its’ current form the application should work for a large scale campus there are certain features which could be added or extended to improve the finished product.

## 7.1 Improving the Route Modifiers Implementation

The Route Modifiers are currently able to identify stairs, lifts and could be easily extended to avoid other hazards such as hills. Route Modifiers could be incredibly useful as discussed later in the 8 Future Applications section. However, the method of implementing the route modifiers could be improved given more time. The current implementation relies on a naming convention in which only nodes that are on stairs or lifts have the term “Stair” or “lift” in the name. This was created mainly as a Proof of Concept and shows that using route modifiers in a final application is possible.

As discussed during the design section it is very easy to modify this functionality and change the checks without affecting either the model or the controller in any way due to the modularity of the design. To implement this feature better all that would be required would be to add extra fields and get methods to the Node object. These could then be set up with whether or not they are available under the different route modifiers on creation and the NodeCheck objects could query the fields rather than check the names.

The difficulty involved in this, and the reason it hasn’t been completed so far, is not in the main app but in the “Start-up” section. To implement this properly it would require changing the file reading section to build the graph in the new format. Although this in itself is not difficult it would mean the text files it uses would be in the wrong format and therefore new text files would be needed. The applications mentioned earlier (EndpointBuilder and MidpointBuilder) would need to be altered heavily to provide the new functionality.

All of the difficulties mentioned above would be easy to overcome. They would however, require more time than is currently available.

## 7.2 Disabled Go Integration

As discussed in Chapter 3.2.3 Disability Improvements there is a webpage named DisabledGo which performs a survey of many Institutions. The survey is made available to the public and details access to different floors or rooms.

One of the main problems with the route modifiers option is that although it is a massive improvement over what is currently available it applies sweeping statements, such as the user wants to avoid all stairs. One possible improvement that could be made to the application would be to provide the user with clickable markers at certain points on the route. When clicked these markers would provide a link to the relevant section on the disabled go website and an option to allow the user to determine whether that section was accessible for them. The application could then save this option for each point the user chooses and purposefully avoid that area from then on.

In the model this would be implemented fairly easily. Once the Route Modifiers Improvement has been made it would simply require adding a new tag and new check. The View and Controller sections of the code would require marginally larger changes to allow the new functionality but overall programmatically the work would be minimal. The main problem this new functionality faces would be the extra data that would be required for each endpoint and midpoint as they would also require a web address to be associated to each one. Adding a web address to each mid and endpoint could make building the maps take significantly longer. Although there are likely alternative methods that could be completed in less time the addition of this feature would doubtlessly make creating the maps take longer.

## 7.3 Multiple Route Options

Currently most of the competing routing applications offer multiple options for the route you can take between the selected points. Implementing multiple routes could be useful in the app for a variety of reasons. One such reason would be to show users alternative routes in case a particular path is blocked.

Multiple routes could be very simply added to the application because of the way the code has been designed. The view would need a very small tweak to change the section currently drawing a route, to instead loop through the provided routes drawing each in a different colour. The RoutePlannerState class could then be modified to carry this list of routes to the view.

Updating the view to display the routes as outlined above leaves only one problem for this extension: how to determine what these multiple routes should be? One possible way to determine the different routes would be to use the concept of Disjoint paths. Princeton University defines Edge Disjoint paths as paths which “have no edge in common” [28]. Using disjoint paths would be a relatively simple way to provide multiple routes. Implementing the disjoint method of multiple routes would involve creating a copy of the graph. Then once the shortest path had been found, likely using the existing algorithm, these edges could be removed from the graph and the algorithm could be repeated until new paths could be found.

There are several problems with using disjoint paths to create the multiple routes. The first problem is that within buildings typically some edges have to be taken for example the path out of a room with one door will always involve using that door. If the definition of disjoint path was applied to the graph as it is built at the moment it would almost never find an alternative to provide to the user.

Another way would be to use the tags that would be included during the route modifiers section to use different criteria to find the multiple routes. An example of using these tags would be to provide the shortest path in a traditional timing sense, the path which spent the smallest amount of time on stairs and the path that spent the smallest amount of time outside. Creating multiple paths based on different criteria the user may be interested in would provide a more meaningful choice.

Implementing an algorithm to provide these different routes should be fairly simple and would only require editing a small section of the model. To change the model to deal with the new functionality would simply involve changing the section in which it uses the edge weights while calculating the shortest path. To find the path which spends the least amount of time on stairs would mean using zero as the edge weight for most edges and only using the standard edge weights for the edges which involve stairs. A more lenient system could also be developed which simply applies a stronger weighting to using stairs than not using stairs. By applying a stronger weighting system to stairs instead of only weighting those edges on stairs the algorithm would produce a path which would still use stairs if it made the journey significantly shorter.

## 7.4 Reaching More Platforms

In order to reach platforms other than Android, such as iOS, the application would have to be rewritten in another language. This would not be as difficult as starting again as although iOS uses a different language, Objective C instead of Java, both languages strongly support and encourage the Object Oriented (OO) Architectural Style. As Objective C supports OO concepts this means the majority of the design could be used without any changes. The only changes that would really be necessary within the Model and Controller sections of the application would be language specific implementation changes. The View section of the application however would require significant changes as it currently relies heavily on the Android provided User Interface (UI) components. To port the View section over to run successfully on iOS would require research into how to use the iOS provided UI components and integrate with Google Maps from an iOS device. All of the problems mentioned concerning porting the application to work with iOS are possible to overcome however doing so would have taken more time than is available in this project.

An alternative to manually creating the iOS application from the existing design would be to use a transpiler. A transpiler is a specific type of compiler which changes a program written in one level of abstraction and changes it into another language with the same or similar level of abstraction [29]. This could potentially be used to create the Objective C code directly from the already developed Java. There is a powerful and widely available transpiler named j2objc which was developed by Google and should be able to automatically create the Model and Controller classes. J2objc was used to help create Google’s new e-mail client [30]. There are potentially some problems with using a transpiler such as the generated code being more difficult to understand. The time it would save in development however mean it is an option that should be considered.

## 7.5 Release and Updates

The final release of the application would most likely be as part of another application. The main use for the application would be as a service provided by an institution. As a result, including the application as a section in the institutions own application would likely be the best way to provide the service to users. Adding the application to another would involve more work as it would require a way to navigate between the rest of the institutions application and the mapping application. Android applications traditionally work on the concept of “activities” a self-contained piece of functionality displayed on one screen. As the entire mapping application takes place on one screen with very minimal changes to the display throughout it was developed as one Activity. With only one Activity adding the mapping section to an already existing Android application should be fairly easy.

Updating the application once it has been published is very simple using the Google Play Store framework. To do so the version number simply has to be incremented and a new compiled version of the code uploaded to Google. This will then automatically update the application for all users who have allowed automatic updates. For those users who have not enabled automatic updates the new version will be available for them to manually update on the Google Play Store.

As updating the application is so easy there is the potential to release the routing application simply providing outdoor directions to start with and then gradually adding a building at a time through updates. By releasing the map as new sections are available it means users can potentially use the system sooner.

## 7.6 Adding Location Services

Adding location services would be a potentially long and difficult extension however potentially very rewarding to the end user. By adding location services as a way to determine the start point of your route it would allow users who don’t know their current location to navigate around the university. It could also potentially be used to update the route as the user attempts to follow the route. By updating the route as it is followed the user would be able to understand the route more easily and navigate around potential obstructions by simply walking away until the route changed.

As discussed during the 3.4 Location Services section of the report the best way to provide Location Services within the application is using a combination of the already available technology and Wi-Fi based location. To do this the Google provided location will be constantly checked and if the accuracy level is above a certain value it will be assumed the user is outside and the Google provided location will be used. If however, the accuracy level falls below that value then the Wi-Fi hotspots the user’s phone is aware of and they’re relative signal strengths an approximate location will be calculated using triangulation as discussed in 3.4.2 Cell Tower Based Location.

The functionality would fit well within the already existing design. By creating a new special case node named My Location it would allow the user to select it using the text boxes in the same way they can currently select a room or building. This node would implement the INode interface allowing the model to treat it the same as any other node. However, hidden behind the interface would be a complicated series of listeners waiting for a location change from the currently preferred method of detection. When the location changes the Node would detect which of the already existing nodes in the graph it is closest to and change the edges to mirror this node. By mirroring the edges of the Node which most resembles its’ own location this allows My Location to constantly integrate with the graph regardless of where it is.

There would also need to be slight changes to the view to incorporate the location services functionality. The changes needed within the view would be relatively simple however, only requiring that if the view recognises the start or destination to be “My Location” then it should draw a marker at the node nearest that location.

# 8 Future Applications

There are many potential uses for the application both in its’ current state, provided the additional data can be obtained, and even more if some of the progress alluded to in the future work section is completed.

## 8.1 Without Extensions

As it is currently the application could be useful in a wide variety of circumstances to many different departments within the University of Strathclyde and potentially any institution mapped indoors within google maps.

### 8.1.1 Recruitment

The University of Strathclyde, and indeed many other University Campuses, faces an issue when trying to attract new students. Although campus maps are provided and signs are placed throughout the University it can still be difficult for prospective students to find the rooms events are being held in. Making this application widely available to prospective students would enable them to find the event’s far more easily. In addition to this it may even make them more comfortable joining the University if they feel they will be able to navigate around it easily. Finally, it will also reduce the workload placed on the university as the extra directional signs will no longer have to be posted.

### 8.1.2 Conferences

As with any visitor’s conference attendees will not be expected to be familiar with the campus. This can make it difficult for them to find their way around. By supplying a version of the mobile application created during this project navigation could be made far simpler. A room number could simply be given to the attendees and with the application it could be assumed they would easily be able to find the room.

### 8.1.3 Deliveries

The application is easily envisioned in a delivery setting. Again this is largely as the University doesn’t tend to have one obvious and easy to find front desk. As a result, deliveries can potentially be made throughout the university. This can cause serious problems as the people making the delivery may not be familiar with the campus making it difficult to find the correct location. The “avoid stairs” option may also be particularly useful if the delivery is difficult to manoeuvre. TODO Ask Derek about it?

## 8.2 With Extensions

By altering the application very slightly and improving upon certain aspects there are even more usages that can be envisioned easily.

### 8.2.1 Disability Services

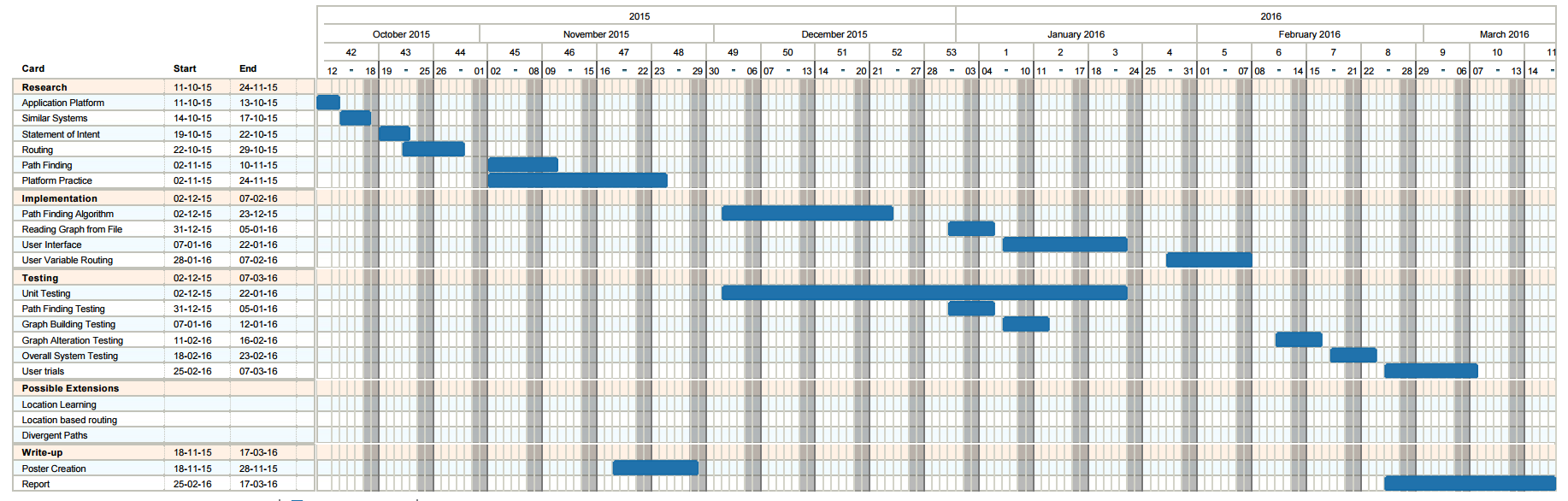
By adding the ability to avoid hills or other such obstacles as well as the already implemented “avoid stairs” feature, the application becomes very powerful in allowing mobility impaired users to navigate around the university. Currently, navigating around large campuses such as the John Anderson campus at Strathclyde can be very difficult for students, staff or visitors if they can’t use stairs or climb steep hills.

### 8.2.2 Fire Safety

Within Non-Domestic properties in Scotland, and indeed most of the United Kingdom, it is the responsibility of those in control of the premises to provide evacuation plans for all of the people in the building [31] [32]. This is typically carried out through the creation of a Personal Emergency Egress Plan (PEEP) for individual disabled visitors or the creation of standard PEEPs in busy public areas. For a large campus this potentially means detailing the quickest route out of a building for every room in every building under a variety of different conditions such as avoiding lifts or stairs. This could potentially take a long time however with a very small modification to allow the application to route to any external location the fire safety officer could potentially create these far quicker. By adding an additional option to allow the user to select and area to avoid as well this would essentially create dynamic PEEPs constantly available on the users’ phone.

# Appendices

## Appendix 1 Original Project Plan



This Gantt Chart was created using a combination of Trello ([www.trello.com](http://www.trello.com)) and Ganttify (www.gantt-chart.com)

## Appendix 2 Final Project Plan

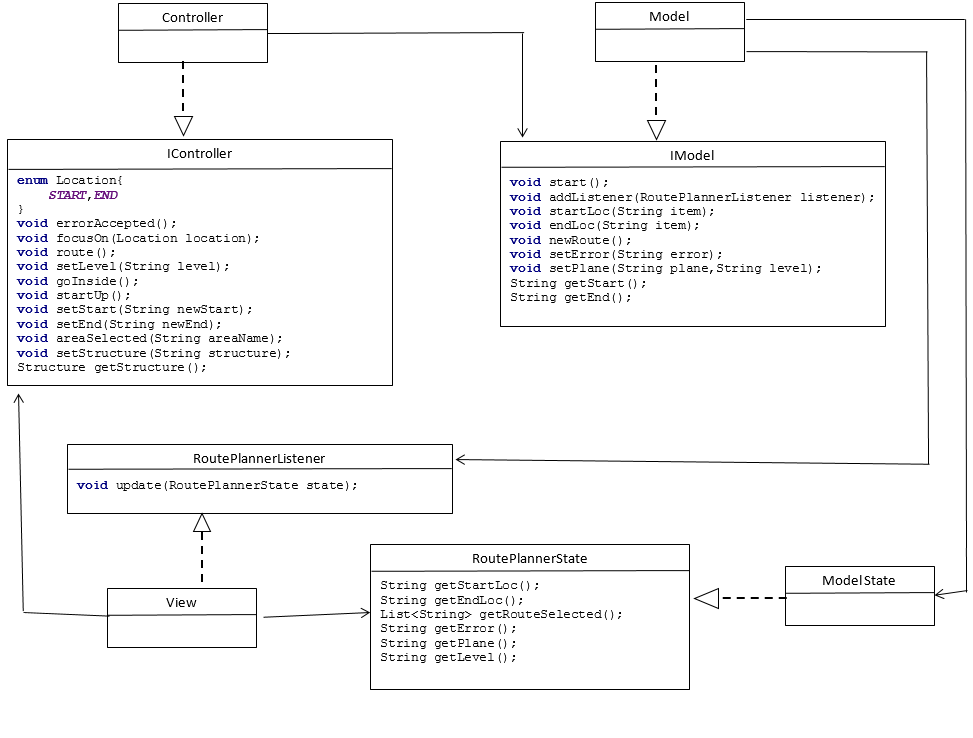


This Gantt Chart was created using a combination of Trello ([www.trello.com](http://www.trello.com)) and Ganttify (www.gantt-chart.com)

## Appendix 3 Comparison of Native and Web Applications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programming Language | Access to phone sensors | Performance | Portability | Issue to compare |
| Uses Object Oriented languages (Java or Objective C) which makes Encapsulation, Abstraction, etc far easier than they would have been otherwise. | Virtually full access to all of the phone sensors. | Developed specifically for each platform which means greater performance. | Can only be used on the OS it was designed for. | Native Application |
| 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. | More restricted access to phone sensors. | Lower performance. | Can be used on virtually any Operating System that includes support for a web browser. | Web Application |
| Native | Native | Native | Web | Better for the project |
| It provides a better framework for producing an error free application more quickly. | This could be important for location services. Although Web applications have access to geolocation information [33] they do not appear to have access to information on Wi-Fi access points they are not currently connected to [34]. Native applications by comparison have access to this information [24]. | 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. | 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 | Reason |

## Appendix 4 Initial High Level UML Diagram

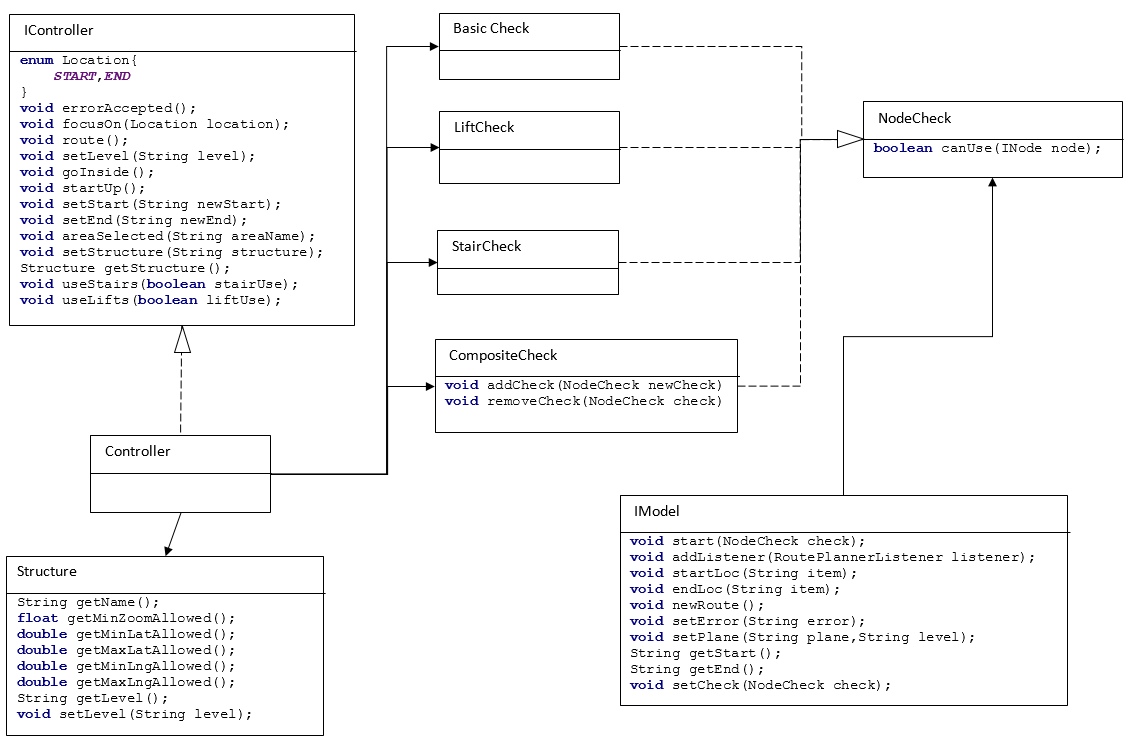


## Appendix 5 View UML Diagram

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

## Appendix 6 Model UML Diagram

## Appendix 7 Controller UML Diagram

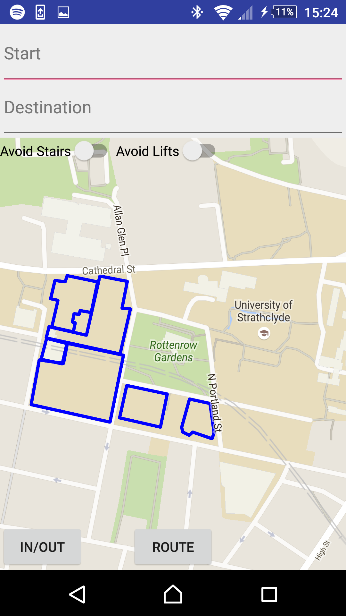


## Appendix 8 Full System Test Results

### Test One

Selecting an endpoint by tapping it is a fairly quick test and simply involves loading the application and tapping one of the endpoints that is displayed. The successful result of this test is shown in Figure 19 with before shown on the left and after on the right.

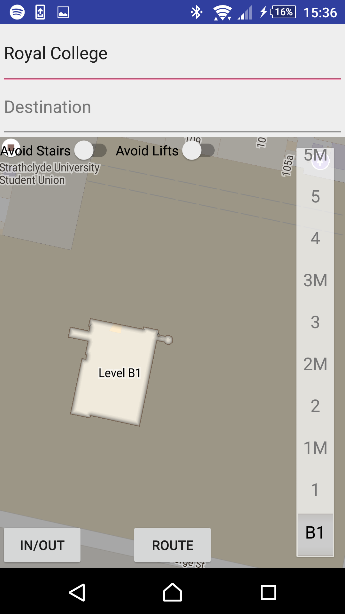
###### Figure Test One Result



### Test Two

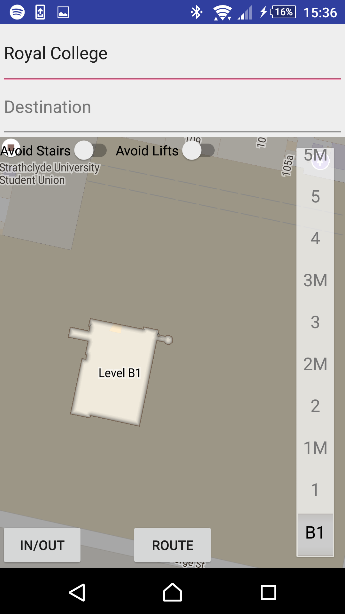
Moving inside a building using the In/Out building can follow on from Test 1 by simply clicking the In/Out button once a building has been selected. The successful result of this test is shown in Figure 20.

### Test Three



###### Figure Test Two Result

Changing a level inside a building can be tested by moving inside a building and then selecting another floor from the floor picker. The successful result of this test can be seen in Figure 21.



###### Figure Test Three Result

### Test Four

Moving outside of a building requires pressing the In/Out button while viewing the interior of a building. The successful result of this test can be seen in Figure 22.



###### Figure Test Four Result

### Test Five

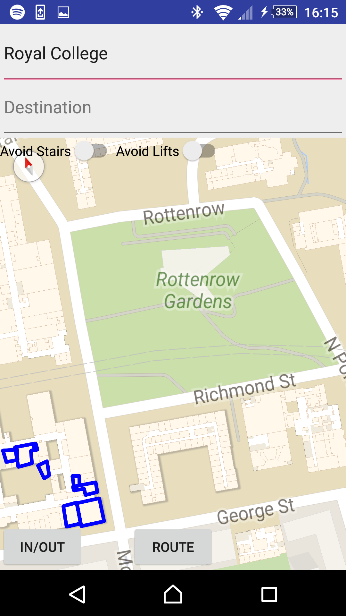
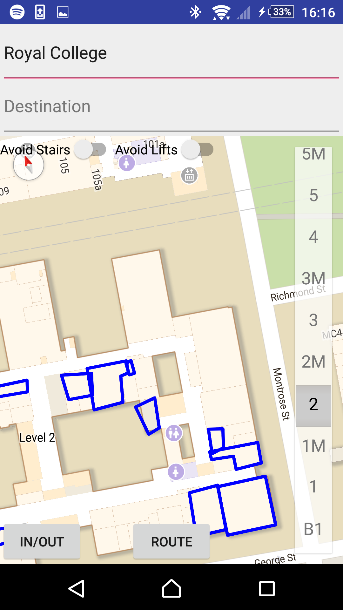
Ensuring the buildings maintain a knowledge of the last viewed floor involves: moving into a building, changing the floor, leaving the building, then returning and checking the level is the same. If the tests have been performed in order this simply involves returning to the inside of the previous building. The successful result of this test can be seen in Figure 23.



###### Figure Test Five Result

### Test Six

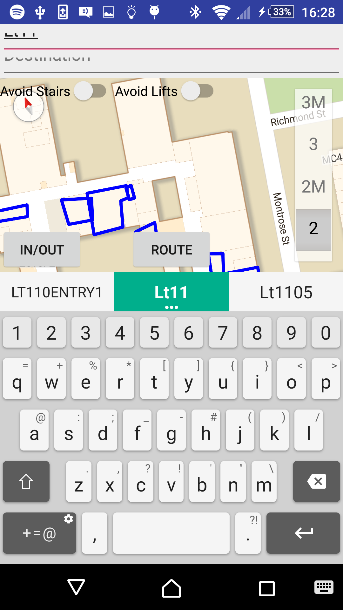
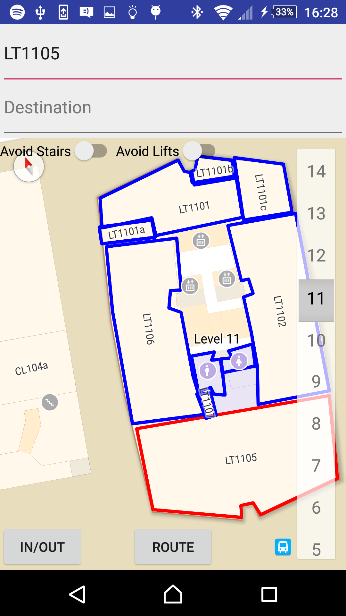
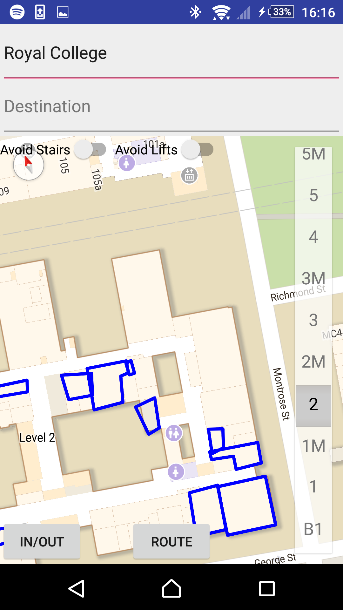
Ensuring building views are limited involves: viewing the inside of a building, scrolling outside the bounds of that building and ensuring the camera returns to that building. This can be proved by zooming further out than is allowed simultaneously as well to ensure it is also limited correctly. The successful result of this test can be found in Figure 25.



###### Figure Test Six Result

### Test Seven

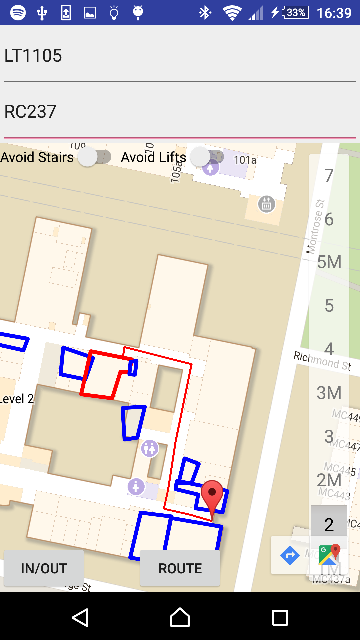
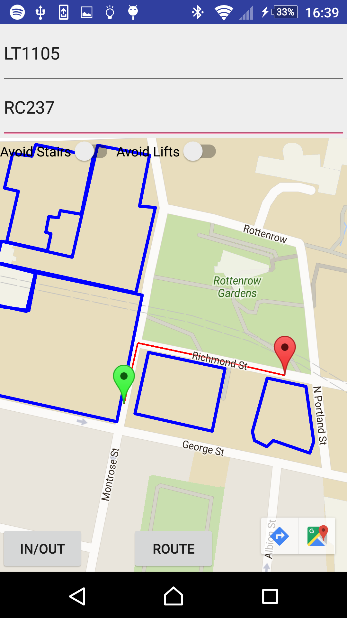
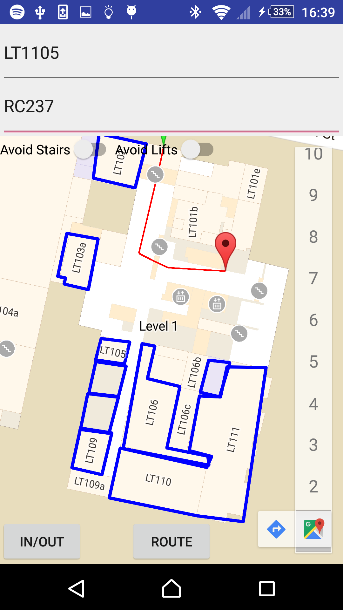
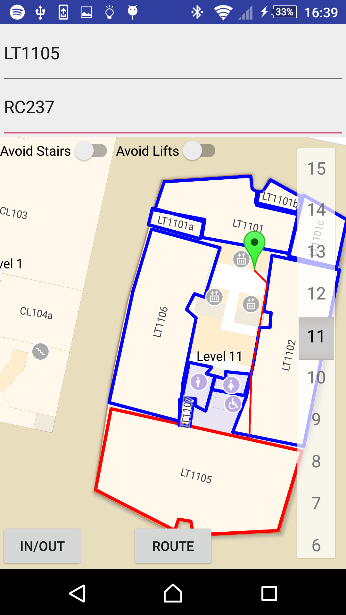
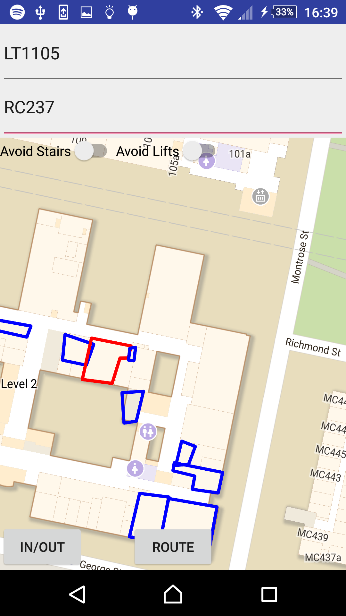
Selecting an endpoint using text entry involves: beginning to type the name of the endpoint and selecting the name from the menu that appears. The camera should then pan to that endpoint and show it is selected. The successful result of this test can be found in Figure 26. Please note that a dropdown does appear when the user begins typing however does not appear on screenshots.



###### Figure Test Seven Result

### Test Eight

Requesting a route which must cross several planes involves choosing two nodes that would require the route to pass through several planes and clicking route. The successful result of this test can be found in

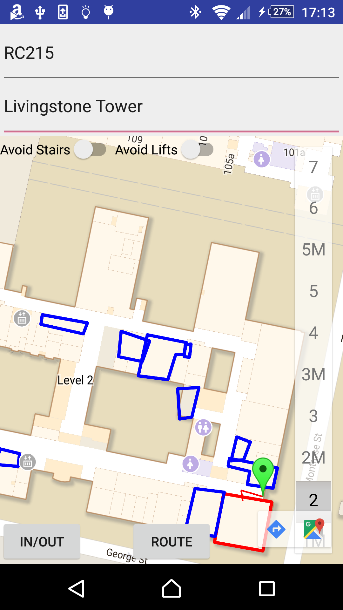
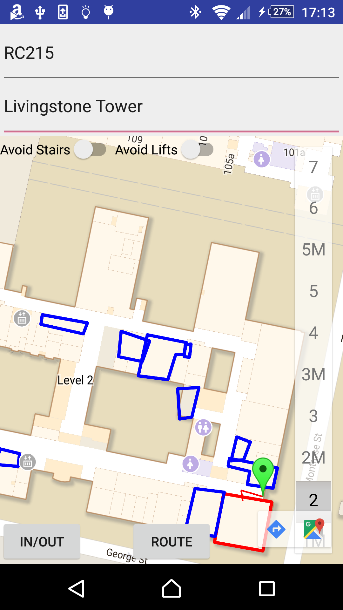
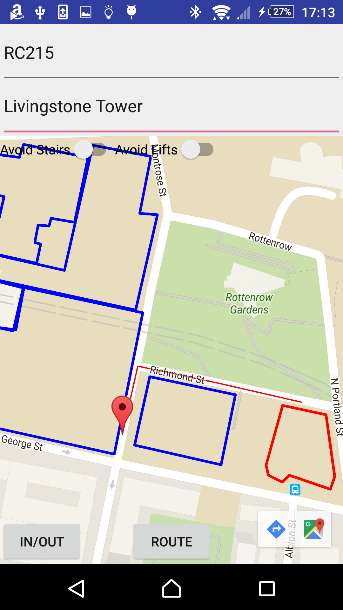


###### Figure Test Eight Result

### Test Nine

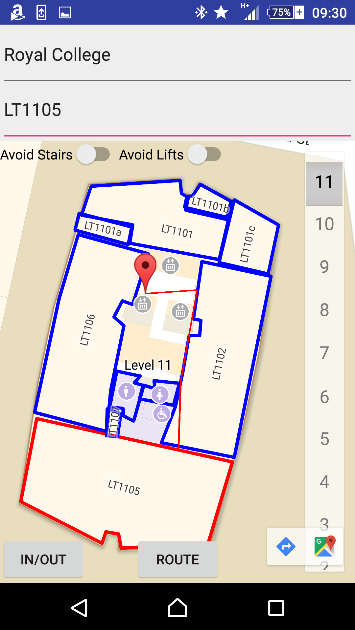
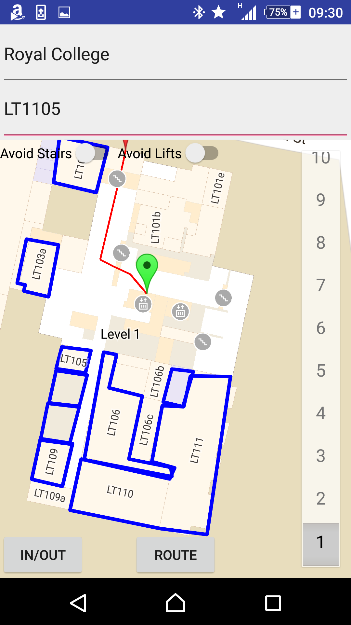
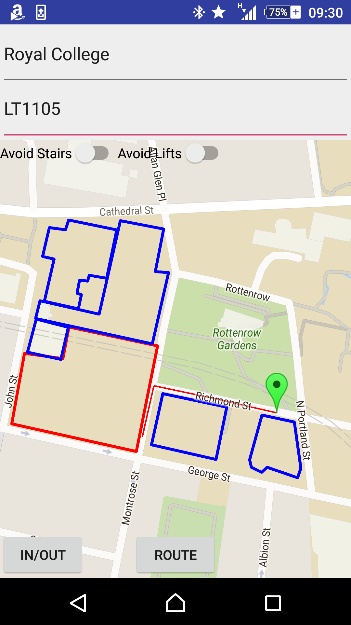
The ability to follow the route using markers requires requesting a route that moves between planes then clicking the marker that appears. The application should then switch to the next or previous plane depending on the colour of the marker. The successful result of this test can be found in Figure 28.

###### Figure Test Nine Result



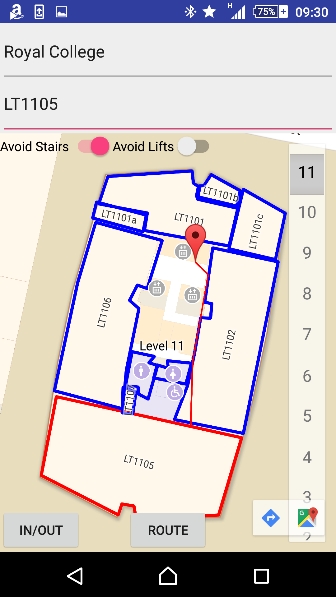
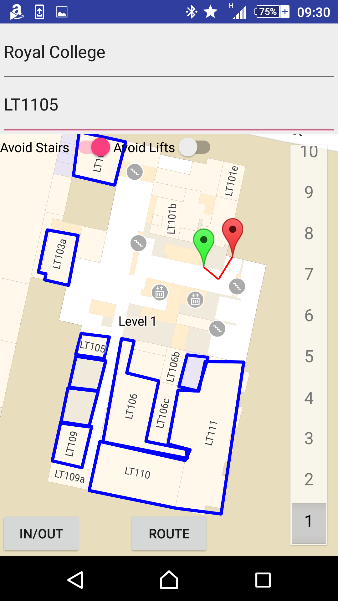
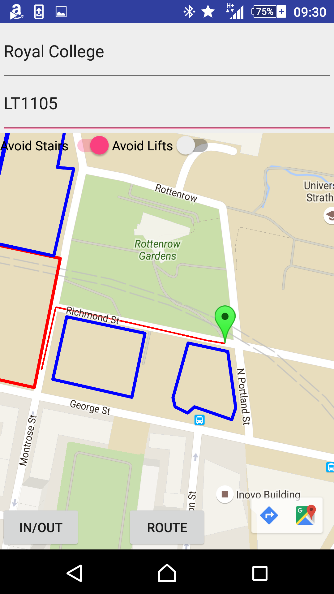
### Test Ten

Testing each of the route modifiers changes the route as is expected involves choosing a route which could potentially use either stairs or lifts. Then requesting the route with every possible combination of route modifiers enabled and ensuring the routes change in an appropriate way. A route with neither modifier enabled can be found in Figure 29.



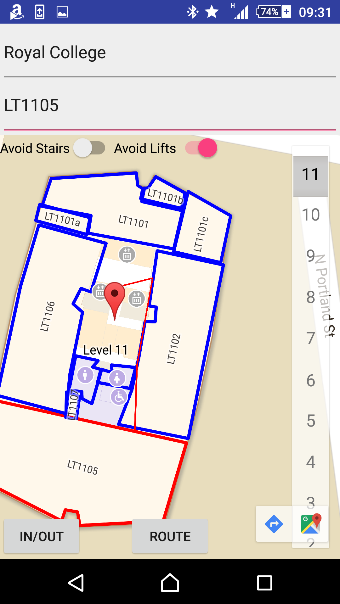
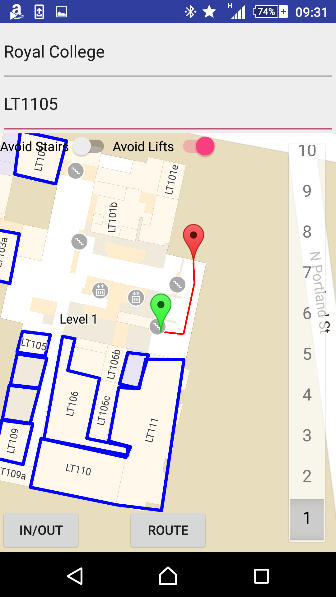
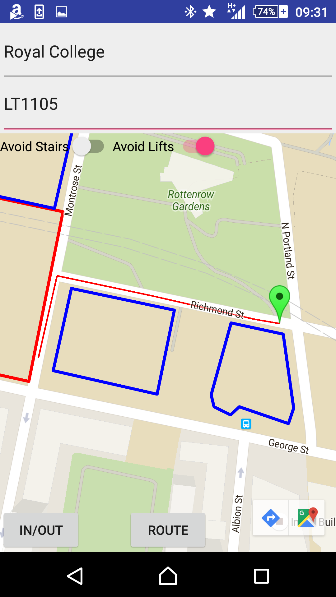
###### Figure Test Ten Result One

The same route with avoid stairs enabled can be found in Figure 30.As can be seen from Figure 30 the route does now avoid stairs.



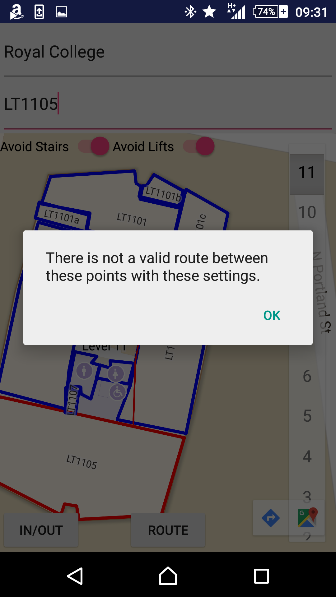
###### Figure Test Ten Result Two

The same route with avoid lifts enabled can be found in Figure 31. As can be seen in Figure 31 the route does now avoid lifts.



###### Figure Test Ten Result Three

The same route was attempted with both avoid stairs and avoid lifts enabled. This route does not exist and the user is alerted to the fact as can be seen in Figure 32.



###### Figure Test Ten Result Four

## Appendix 9 User Guide

## Appendix 10 Installation Guide

Link to github again for the download bit.

# References

|  |  |
| --- | --- |
| [1] | Trello, “Trello,” [Online]. Available: https://trello.com/b/6cUNFaP3/university-route-finder. [Accessed 25 March 2016]. |
| [2] | TomsPlanner, “Ganttify,” [Online]. Available: https://www.gantt-chart.com/. [Accessed 25 March 2016]. |
| [3] | Github, “Github,” [Online]. Available: github.com. [Accessed 25 March 2016]. |
| [4] | 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]. |
| [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] | Google, “Google Play Store,” Google, 18 March 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.maps. [Accessed 25 March 2016]. |
| [11] | Smithsonian Institution, “Google Play Store: Smithsonian Mobile,” 31 January 2014. [Online]. Available: https://play.google.com/store/apps/details?id=edu.si.sim3. [Accessed 26 March 2016]. |
| [12] | radKOtev, “Google Play Store: Glasgow Uni Map,” 29 September 2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.radkokotev.glasgow.uni.map. [Accessed 26 March 2016]. |
| [13] | 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. |
| [14] | M. Rouse, “TechTarget,” September 2005. [Online]. Available: http://searchnetworking.techtarget.com/definition/GNSS. [Accessed 7 March 2016]. |
| [15] | E. K. a. C. Hegarty, “GPS Overview,” in *Understanding GPS: Principles and Applications*, Artech House, 2005, pp. 3-4. |
| [16] | 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]. |
| [17] | 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]. |
| [18] | 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]. |
| [19] | 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]. |
| [20] | iPosi, “iposi.com,” 2014. [Online]. Available: http://www.iposi.com/tech-talk/the-technology. [Accessed 7 March 2016]. |
| [21] | D. Vaughan, “Geographic Location,” in *Windows Phone 7.5 Unleashed*, Sams Publishing, 2012, pp. 529-552. |
| [22] | J. Yang, A. Varshavsky, H. Liu, Y. Chen and M. Gruteser, “Accuracy Characterization of Cell Tower Localization,” UbiComp '10, New York, 2010. |
| [23] | Federal Communications Commission, “Accurate Location Detection 911 Help SMS App,” 2015. |
| [24] | Google, “Location Strategies,” [Online]. Available: http://developer.android.com/guide/topics/location/strategies.html. [Accessed 8 March 2016]. |
| [25] | 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]. |
| [26] | T. Cormen, D. Balkcom and K. A. C. C. Team, “Representing Graphs,” [Online]. Available: https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs. [Accessed 9 March 2016]. |
| [27] | TranSafety, Inc, “Study Compares Older and Younger Pedestrian Walking Speeds,” *Road Engineering Journal,* no. October, 1997, p. 1, 1997. |
| [28] | K. Wayne, “Princeton,” 2005. [Online]. Available: http://www.cs.princeton.edu/courses/archive/spr05/cos423/lectures/07maxflow-applications.pdf. [Accessed 22 March 2016]. |
| [29] | S. Fenton, “COMPILING VS TRANSPILING,” 18 November 2012. [Online]. Available: https://www.stevefenton.co.uk/2012/11/compiling-vs-transpiling/. [Accessed 23 March 2016]. |
| [30] | R. Amadeo, “How Google Inbox shares 70% of its code across Android, iOS, and the Web,” 20 November 2014. [Online]. Available: http://arstechnica.com/information-technology/2014/11/how-google-inbox-shares-70-of-its-code-across-android-ios-and-the-web/. [Accessed 23 March 2016]. |
| [31] | Safer Scotland Scottish Government, “Practical Fire Safety Guidance: The Evacuation of Disabed Persons from Buildings,” [Online]. Available: http://www.gov.scot/Resource/0040/00402451.pdf. [Accessed 22 March 2016]. |
| [32] | HM Government, “Fire Safety Risk Assesment Means of Escape for Disabled People,” [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment\_data/file/422202/9446\_Means\_of\_Escape\_v2\_.pdf. [Accessed 22 March 2016]. |
| [33] | w3schools, “w3schools,” [Online]. Available: http://www.w3schools.com/html/html5\_geolocation.asp. [Accessed 8 March 2016]. |
| [34] | A. De Rosa, “envatotuts+,” 8 August 2014. [Online]. Available: http://code.tutsplus.com/tutorials/html5-network-information-api--cms-21598. [Accessed 8 March 2016]. |