

Adapting Access Aber making use of native Android functionality.

Final Report for CS39440 Major Project

Author: Thomas Keogh (thk11@aber.ac.uk)

Supervisor: Dr. Myra Wilson (mxw@aber.ac.uk)

3rd May 2015

Version: 1.4 (Release)

This report was submitted as partial fulfilment of a BSc degree in
Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature

Date

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature

Date

Ethics Form Application Number

The Ethics Form Application Number for this project is: 904.

Student Number



120092673

Acknowledgements

I am grateful to...

I'd like to thank...

Abstract

The aim of this project is to provide an alternative Android native application to the current Access Aber web application, while also improving on the groundwork laid out by the initial work. This work is being completed due to the help that it can provide to users who are visiting the University as well as providing at least some assistance to those with disabilities. Aberystwyth's campus can be difficult to navigate and the benefits provided from a mobile device can benefit users to a large degree. The application was developed for the Android OS using the Android Studio IDE and relies heavily upon the Google Maps API. It provides a range of services to the user, some of which were brought over from the existing application and improved, some new. These include route finding around campus, user refined displays of building locations, route plotter with application compatible outputs and a location based help system. This dissertation covers the original analysis, requirements, design and implementation of the project as well as details on the testing and design decisions made throughout the development. The conclusion is an Android application which met its original requirements specification and adds further complexity to the initial web version of the project. This includes route grading and the representation of this along with an extendible solution to route finding using Google Maps and graph searching techniques.

CONTENTS

1	Background & Objectives	1
1.1	Background	1
1.1.1	Background and Preparation	1
1.1.2	Interests	2
1.2	Analysis	3
1.2.1	Breakdown of Requirements	4
1.2.2	Itemized Requirements	7
1.3	Process	8
2	Design	10
2.1	Support and Development Tools	11
2.1.1	Development Environment	11
2.1.2	Version Control	12
2.2	Overall Architecture	13
2.2.1	Class Diagram and Justification	14
2.3	Application Flow	19
2.3.1	Overall Justification	19
2.4	Sequence Diagrams	23
2.5	Major Algorithms	23
2.6	Interfaces	26
2.6.1	Colour Theme	27
2.6.2	User location	28
2.6.3	Screen Designs	29
3	Implementation	31
3.1	Location Based Help	31
3.1.1	Updating Help Location	32
3.2	Building Display	32
3.2.1	Removal of Render Functions	32
3.2.2	Other Changes	33
3.3	Route Plotter	33
3.3.1	Plotting of points	33
3.3.2	Visualizing and Cancelling	34
3.3.3	File Saving	35
3.3.4	Writing to File	35
3.4	Route Plotting	36
3.4.1	Route Object	37
3.4.2	Implementing Graph	37
3.4.3	Step Free Graph	39
3.4.4	Search Algorithm	40
3.4.5	Painting the Path	40

3.5	UI Design	41
3.5.1	Map	41
3.5.2	Relative Design	41
3.6	Conclusion	42
4	Testing	44
4.1	Overall Approach to Testing	44
4.1.1	Problems Encountered	44
4.2	Automated Testing	45
4.2.1	Unit Tests	45
4.2.2	Stress Testing	45
4.3	Integration Testing	46
4.4	User Testing	47
4.5	Acceptance Testing	47
4.6	Testing Evaluation	47
5	Evaluation	48
5.1	Requirements and Process	48
5.1.1	Requirements	48
5.1.2	Process	48
5.2	Tools and Design	49
5.2.1	Tools	49
5.2.2	Design	49
5.3	Implementation and Aims	49
5.3.1	Implementation	49
5.3.2	Compared to Aims	50
5.4	Future Work	50
5.5	Self Evaluation	51
	Appendices	53
A	Third-Party Code and Libraries	54
B	Code samples	56
2.1	Example Test Code	56
2.2	Route Plotting	56
2.3	Route Finder	57
C	Test Results	59
3.1	Facility Display Test Table	59
3.2	Help Display Test Table	60
3.3	Route Plotter Test Table	60
3.4	Route Finder Test Table	61
3.5	User Testing	62

D	Requirements Specification	63
4.1	Functionality	63
4.1.1	Route Finding	63
4.1.2	Route Plotting	64
4.1.3	Location Based Help	64
4.1.4	Building Display	64
4.1.5	Multi Lingual Support	65
4.2	Interfaces	65
4.2.1	User Interface	65
4.2.2	Google Services	66
4.2.3	Hardware	66
4.2.4	Interfacing with itself.	67
4.3	Performance	67
4.3.1	Offline Performance	67
4.3.2	Searching Algorithm	67
4.3.3	Possible Problem areas.	67
4.4	Attributes	68
4.4.1	Maintainability	68
4.4.2	Expandability	68
4.4.3	Security	69
4.4.4	Design	70
4.4.5	Overall User Interface	71
4.5	Design Constraints	71
4.6	Final Comment	71
E	Design Specification	72
5.1	Introduction	73
5.1.1	Purpose of Document	73
5.1.2	Scope	73
5.1.3	Objectives	73
5.2	Decomposition Description	74
5.2.1	Modules within the Application	74
5.2.2	Design Rationale	77
5.2.3	Significant Classes	78
5.2.4	Mapping Requirements to Class	79
5.3	Important Algorithms	80
5.3.1	Route Searching	80
5.3.2	Plot Path	80
5.3.3	Plot Single Route	81
5.3.4	Paint Path	81
5.3.5	Create Directory	83
5.3.6	Other Algorithms	83
5.4	Design Diagrams	83

5.4.1 Flow Diagram	83
5.4.2 Class Diagram	85
5.4.3 Screen Designs	88
Annotated Bibliography	94

LIST OF FIGURES

1.1	Design Process	9
2.1	Screen Flow	13
2.2	Class Diagram	18
2.3	Flow Diagram	22
2.4	Theme Possibilities	27
2.5	Button Examples	28
2.6	Example Themes	29
2.7	Route Type Example	30
2.8	Route Data Example	30
3.1	Erratic Route Display	34
3.2	Route Before Junction	38
3.3	Route after Junction	39
3.4	FakeView Example	42
4.1	Test Output	45
4.2	Monkey Stress Test	46
E.1	Initial Flow Diagram	84
E.2	Initial Class Diagram	87
E.3	Initial Menu Design	88
E.4	Initial Help Design	89
E.5	Initial Display Design	90
E.6	Initial Plot Design	91
E.7	Initial Find Route	92
E.8	Initial Route Choice Design	93

LIST OF TABLES

C.1	Facility Display Tests	59
C.2	Help Display Tests	60
C.3	Route Plotter Tests	60
C.4	Route Finder Tests	61
C.5	User testing results.	62
E.1	FR Mapping to classes	79

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Background and Preparation

Background research was comprised of analysing existing systems like the original Access Aber [1], previous work completed by the developer that used map technology within Android and general research into the map technology to gather a strong understanding of exactly what was possible.

A majority of time was spent analysing Access Aber and the currently existing criticisms of it, a lot of the problems were clear without the user feedback that had been provided but some criticisms were more subtle but easy to understand, an example being the lack of a main menu, something which left users feeling lost within the program. It was also obvious that at a technical level, the included features were not complex, at least not to complete on the Android platform. Due to the application originally being developed on a system which allowed it to be run both on iOS and Android, limitations probably existed from both sides which lead to having to develop around the weak elements in both platforms. As stated developing for a single platform should help with the avoidance of these problems while also giving the ability to take advantage of what is there. Due to this a large amount of research was completed into both UI design within Android as well as the technical benefits both Open Street Maps [24] and Google Maps [13] could provide the development.

Research was also completed into exactly how the mesh of locations would be created and represented [3], within the design specification it was concluded that a searchable graph was a possible solution and one worth researching further. This also meant analysing how other existing applications had mapped roads to graphs and the general area of map representation [26]. While this gave a fair amount of answers further decisions needed to be made on both how to search the constructed graph and how to make it an extendible system for future additions. This was one of the main conditions brought up in meetings with the original 'customers' of the

Access Aber application, it had to be functional after the initial developers had left. It was also described that it would be beneficial for the application to be able to utilise a central file store in the future. Due to this further research was completed on how to facilitate a file store after this project cycle was completed.

Past work by the developer was also analysed for anything of use, the application analysed was proven to have functional working code and as such was a place where possible solutions could be found. This included implementation of maps in an Android environment, the logging of a route which is something that has been previously outlined as a key requirement and a fair amount of small features relating the monitoring of a users location and the information which can be gathered from that. Most information gathered from this however was fairly irrelevant but it did provide the code for rendering a Map object which provided a start.

Finally the Android API [17] and several Android libraries were examined for the benefits they could provide, along with research on several features that had already been selected for inclusion. This ranged from research on Expandable List Views to research on the best way to present the application in an intuitive way, a lot of factors were gathered from the original feedback which guided research. While a slide function for the screens was visually appealing the possibility of it confusing users was something that removed it as a feature very quickly. Research was also performed into Google's Material Design program [15], a set of guidelines for design within the Android environment. It contained a large amount of information relating to designing professional and visually appealing applications, while the information was too much to be fully applied in the projects time line some key ideas were taken from it relating to the design of elements within an application and general theme layout. Furthermore some sites were analysed for basic colour themes and how to implement a simplistic design in a professional way.

1.1.2 Interests

This project has been chosen due to a variety of reasons, revolving around both the developers interests and the possibility of the project leading to something that can have real world influences. The main source of interest is that the project allows for a development environment that takes input from the real world, mobile devices are of a great deal of interest to the developer due to the data that can be gathered from them and the manipulation of this data. Mobile devices are steadily changing the way we live, with applications that not only help us find where to go but tell us what we can do when we get there. Where other people have liked and even what is best suited to us based on past choices we have made. While some systems outside of mobile development clearly have massive real world influences it is felt by the developer that the easy entry and possibilities provided with mobile development make it one of the most accessible and revolutionary platforms.

With the application in concern providing information to the user it also means it's possible to filter what information is shown based on user details. If the user says they cannot manoeuvre stairs, a function could be implemented to provide paths with no stairs, this could obviously be

expanded to include searching based on a range of filters. This could also be condensed down to a single graph, with different links becoming available with different information. This kind of possibility is what makes mobile such an interesting platform, not only can we show a map and a route through it, we can then guide a user through it, even ask them for information regarding it once they are done. Possibilities provided are wide and interesting, with so many choices available the future of the application is impossible to predict because of the many paths it could follow.

A further source of interest is the real effect this application could have for an organisation such as Aberystwyth University, the confusing and difficult to traverse campus is not a problem that can be resolved easily. With everything on campus set in stone it is simply not possible to move buildings and locations around. However with technology we can facilitate the use of campus without any physical additions to it at all. This means we can change the physical use of something with technology that does not itself directly effect or control the environment it is deployed in. An interesting possibility would be to examine users maneuvering of campus before the application is introduced and afterwards, as well as the effect it has on how comfortable disabled users feel on campus. With the application very literally changing the way a user traverses the campus it is possible that some of the more unused walkways could become more travelled due to their inclusion in the application. Further research will have to be done on most effective travel paths around campus, it is likely that the ones currently most travelled are the quickest but that is not definite.

1.2 Analysis

A broad analysis of the problem effecting the currently existing Access Aber [1] application is that while it provides solutions, they are long, problematic and make the user follow a convoluted process that may not be obvious to the common person. A prime example of the problem posed is in the route finding functionality, it exists but requires the user to scroll through a list of over 200 routes before a summary screen and finally the route display. There are some very obvious, stand out ways, that this can be improved including the implementation of simple features like drop-down boxes for destinations and the removal of the summary screen.

Most of the problems in the application come from simple solutions to more complex problems, simple in the way that they do not fully cover the problem or provide a real benefit to the system. This appears throughout the application and based on the information provided seems to have been caused by the short time period it was developed in. One aim for this project is to provide a much more polished result with each problem solved completely rather than partially.

Access Abers initial form will be treated as a groundwork for development, while no code will actually be taken from it the features will be adapted and built upon. This way something more than a port can be developed, an improvement can be made. Having this groundwork and the feedback on it means that some of the problems are already highlighted, while the time line of the project may not be long enough for some of the features to be fixed most can at least be addressed.

A step by step guide which details various information such as major features and activities can be seen in the Design Specification (Section 2). The following requirements are what has been decided as key to the success of the application, they have been selected due to either their use to the user or the fact that they have always been included in the project.

1.2.1 Breakdown of Requirements

After much analysis the problem has been broken down into a set of requirements, some of the process to arriving at this point can be found in both the Requirements specification with further information in the Design Specification, however the reasoning will be fully discussed here. Each requirement will serve as a benchmark for the project, requirements were set as realistically as possible considering the time given and the work that needs to be completed.

1.2.1.1 Route Finding

Route finding is seen as by far the major feature within the application, not only is it the most technologically advanced it is also the most useful for our prospective users. As detailed, a graph system has been highlighted as the best way to achieve a representation of campus for us to apply a search algorithm to, with Breadth First Search (BFS from now) [20] seen as a possible solution. This feature also has some planned extensions, firstly the path shown to the user should contain extra information pertaining to the path. This includes information about the difficulty of the route and features along the route such as stairs, it would be beneficial to have information for separate segments beyond their rating but this is a stretch goal and not something realistically expected. Due to the expected structure of the graph the implementation of displaying rating of segments within the paths should be attainable, with large routes being made up of many links between nodes it is almost to be expected in a system such as this.

It is required that the route finding solution be easily expandable, this is due to the maintainability of the application being key to its success. There is no guarantee that the original developer will be around for future expanding if the application is deemed fit for purpose and further developed, due to this it must be possible for the addition of new locations to be possible. An ideal solution would use a central file store to pull location information from, however the problem here which has been previously covered is that there is no guarantee the user has access to internet when using the application. Due to this a sufficient solution should see that the extending of route finding should be achievable by a novice programmer, ideally a common user if they are provided with clear instructions. A set of guidelines should be produced if the project is taken further.

A further stretch goal would be an implementation of route finding based on a sort of filter, whether this be a routes rating or features on the route. A current rough idea of this can be found in the Design Specification (2.1.1), this highlights the idea of using a separate graph altogether, this way if the search algorithm is made so that all it needs is a set of files in a set way we can have a variety of graphs present within the application. An ideal solution would use a consolidated

graph, with different routes between different Nodes being considered, however this is seen as problematic in the current time frame.

A final possible addition, though not likely to be achievable in the time frame, is to provide access from anywhere to somewhere on campus. This means having a user be guided into the graph from areas outside of it, meaning dynamic routes would have to be created. Initial research shows this could be possible using the Google Directions API [12] however with other development ongoing this is likely to be a feature developed outside of the initial time frame. However the use of this feature would be a real benefit for the application, a simplistic solution would have the user guided to the nearest node and then the search algorithm would be run. It is even possible that in future the whole graph would be replaced with the Directions API, this however would require the user to have access to the internet at all times and it is simply something that cannot be guaranteed. Due to this the hard coded graph and search related to the graph is almost certain to be present for the lifetime of the application due to the required off line capabilities.

1.2.1.2 Route Plotting

This desired features comes as a result of the original meetings, the 'customers' were unhappy with the current system of plotting routes, which were all done manually. Part of the solution is already achieved through the use of a graph, a location only needs to be linked to one other to have a route available to anywhere on the graph. However this should be further developed for the application to be seen as fully functional. The application should include a feature that lets a user log a route and have it printed out to a file compatible with the current route finding solution, this way the person responsible for the future maintenance of the application can visualize a route before they add it to the current system. A route should be able to be visualized while it is being created, so that the user is not blindly adding points. GPS co ordinates received from some devices can be far from where the actual user is. Functionality within the Google Maps API [14] allows us to show the user where the phone thinks they are, this should help them plot accurate points.

The output file, the files used by the route finder, should also be human readable. This requirement is not as key as the others but should be achievable as the file reader will be coded for this exact purpose. Having a human readable file provides various benefits, mostly that of being easily editable by anyone who wishes to change information in the future. Its negatives are few, while a non human readable may be able to be smaller the benefits time wise will be minimal. This also means that route plotting using the in built feature is not required, files can still be made by hand due to the standard format to be laid out.

Finally the Route Plotter should provide a grading for the route that has been laid out by the user. This is the grading that will be displayed by colour on the route finder, while the initial rating is likely to be numerical this will be translated into the red, amber, green system that has been mentioned in previous meetings. Having this easily identifiable system provides the user with a sense of understanding without having to state the feature on screen and cause

cluttering of the UI. Ideally the route plotter will take into account cumulative distance, steps on the route and elevation. However after initial research elevation is not only sometimes unreliable but a complex overall problem, to do with both the misleading information it can provide and the problems in finding elevation. However an ideal solution would take all three factors into account fairly.

1.2.1.3 Building Display

Building display is a feature currently implemented fairly well within the original application and is something we can mostly just adapt, however there are improvements that can be made, mostly found due to the user feedback. This feature should allow for users to highlight building types and have them displayed on a map of campus, this way a user can browse around what they want rather than have to try and guess where certain areas of interest are. A user should also have their location displayed to them for easy guidance.

One criticism that has come from the user feedback is that the display can be unclear, this includes building departments sometimes not being named and just a general lack of clarity. A good example is buildings named after people just have the person's name, for example 'Hugh Owen'. Problems like this should be fixed for the requirement to be met. Again a facility for easy addition to this feature should be added, preferable through on device files for the time being due to previously highlighted problems with a central file store.

1.2.1.4 Location Based Help

Location Based help is seen as the easiest desired feature, something that the web application also currently lacks. It is seen as more of a test than the other features which seem solid and long lasting, the resources available to the University may not be able to facilitate the full deployment of this feature. In short the feature should be able to provide the user with the details of the closest person who could provide help, currently this is limited to the three libraries which have their accessibility officers details listed on the Universities site. Other help details should be sought out in future.

A possible change to this implementation is the addition of porters to the help listed, if all three locations are too far away, a porter could be contacted instead. However a dialogue would have to be opened up with the University over whether they find these practices acceptable. For this application the basic Help function will be developed as a proof of concept for possible future use.

1.2.1.5 UI Design

Interface Design itself contains a set of requirements that can be mostly boiled down to one key aspect - an intuitive interface for the user. Currently there are several problems with the existing interface that can be fixed fairly easily, like the lack of a menu and unclear buttons. However

a full interface will take time to develop, testing and a good amount of inspiration from other sources. Within this project it is the aim to have a clean, practical layout that most users can just pick up and use without any previous knowledge of the layout. By doing this we guarantee less confusion and problem free use of the project, it also means the use of the Help feature could be reduced which would be a large benefit in a full release.

This requirement can also be broken down into further aspects than just 'intuitive design'. Firstly the theme used within the application must be consistent throughout, it would also be beneficial if it coincided with the Universities colours of yellow and purple but this will not be treated as a requirement, due to the restrictions it places on the development team. Buttons should be labelled clearly with reactive designs, there are few issues worse than unresponsive buttons that do not give the user the feeling of actually starting a process. The design should also be simple with non overpowering colours, the user should feel comfortable and a confusing sea of bright colours over a map is not a good way to achieve this.

Future goals could be varied colour schemes based on colour blindness, with the application aiming to help those with disabilities ignoring those with colour blindness is a large oversight and is something that should be considered in the future. It may be possible to implement this in this version and is something that should be strived for. Another feature which is seen as a requirement is Welsh language support. With the application being developed for a Welsh University the inclusion of this is ideal, however current research points to a range of issues that will need development time to see if they can be worked around.

1.2.2 Itemized Requirements

Following is the list of requirements for the project, some requirements can also be seen in the requirements document however these detail a more up to date version with a greater focus.

- FR1 - Route Finding - Guiding a User around campus, including the display of route grading and a no steps graph.
- FR2 - Route Plotting - Users must be able to plot their own routes around campus and print them to an application compatible file.
- FR3 - Location Based Help - Provide details of possible help available, possibly include the addition of contacting porters.
- FR4 - Building Display - A display of related buildings to be filtered by the user, contains accurate information on buildings and the departments in them.
- FR5 - Multi Lingual Support - Welsh translations provided in the application and a user choice on what language they would like should be attempted.
- FR6 - A clean and simplistic UI with clearly labelled buttons and a responsive design, possibly including colour blind themes.

1.3 Process

For the development of this project an adapted Feature Driven Development style will be used, taking a lot of ideas from FDD but still changing the development style slightly. To begin with an overall design was specified in the Design Specification document, this included the design for the entire program. It covers a range of information, including an overall model which is the starting step of FDD, information on the key expected algorithms and key classes to be expected.

A type of feature list was also developed within the Requirements specification, while more verbose than may be expected with a feature list it still detailed the features expected and the finer details to be included. This document is more abstract and tends to not focus on the actual technology too much, just what is desired from the program itself. The Requirements Specification also outlines key attributes that must be included within the application, this includes details on the maintainability of the project and possible choices for future expansion.

Planning by feature was something that was mainly omitted from the process, due to using a single developer all of the work load fell onto the single person. However milestones were still set for iterations of every three days or so, estimations on progress were made. A set feature was either expected to be completed or a quantifiable progress. Some difficult areas of the project led to targets not being met but overall the model provides a target to aim for in a given time frame.

Designing by feature was also a process that only partly took place, due to the majority of design happening near the start of the project a majority of the time the design by feature process involved the editing of existing documentation to better reflect what was now expected. As development progresses designs for future features tend to change a lot due to shifting scope and time restrictions either becoming tighter or allowing more polished features. Having a design from the start meant that development could continually take place without major breaks for planning out a feature, this meant that once the application hit its main development stage it got quicker to develop due to a familiarity with the code base and the final image of the software becoming clearer as time progressed.

Building by feature meant the application had modules added to it weekly, creating a progress that was quantifiable. This development technique was decided to be the best process for a smooth development period. It also helped that the application could clearly be visualized as having several different branches, most of which did not rely on one another. After development was finished tests were performed, both code wise and with the application itself ensuring the code worked as expected. It was also important to test that the code was expandable and that the plans for the future could be included in the way that was highlighted within the Design and Requirements specification.

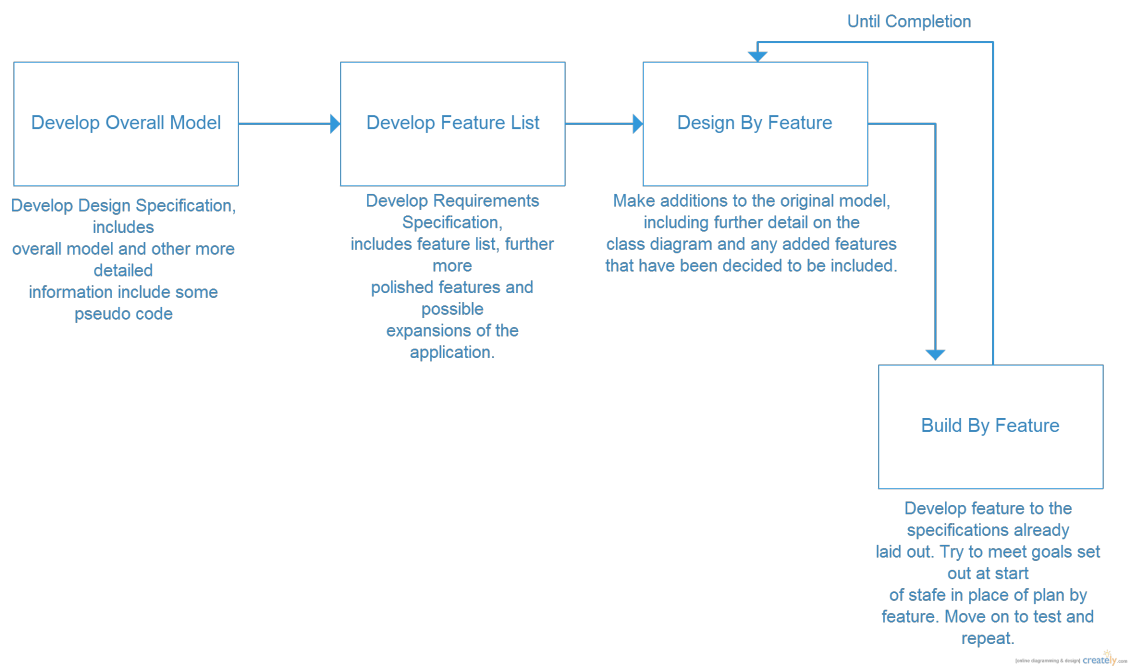


Figure 1.1: Current Planned Design Process, an adapted version of FDD

Chapter 2

Design

The following section will cover the final design of the project, initial design can be seen in the Design Specification. The original design was completed when only a small amount of research had been completed and not all avenues of interest had been explored. Due to this good comparisons can be made between the two making it easy to highlight the areas of the design where large changes were made. The design covered is the cumulative output of the Design by Feature element in the process. This means some changes were still made during the implementation phase but those will be discussed in the relevant section.

2.1 Support and Development Tools

2.1.1 Development Environment

During the initial phases of development and prototyping Eclipse and the Android Development Tools [10] plug in were used to help develop the application, this was due to the developer having experience with this platform and the code being sampled from other projects was also built within this platform. However not long after initial prototyping and tech spikes had started to take place the decision was made to swap over to Android Studio and develop from there. Most of the decision comes down to personal preference but there are definitely good justifications to swap, personal preferences included appearance and responsiveness as well as easy to use features such as inbuilt git support and intuitive interfaces.

One major contributing factor was the initial hassle in updating the ADT to their latest versions, this process alone took hours of research and tinkering to get fully working. During this time it was discovered that Google had stopped support for Eclipse ADT and would be focusing on the now fully released Android Studio [8]. A decision was made, based on the advice from the Android API and other sites, to move over to Android Studio as it would quickly surpass Eclipse ADT with some features already being far more helpful.

One key feature that was seen as a bonus when compared to ADT was the interface editor, while this is present in the Eclipse version the process in Android Studio seemed much more responsive and seemed to give a clearer idea of what XML was being written to support the graphical changes. While nearly all interface development was completed using pure XML sometimes moving elements using the graphical layout gave good suggestions on how a set layout could be achieved. While the code produced by this could sometimes be messy and convoluted it could be cleaned up and improved.

Some of the choice to move over was also personal opinion of the developers, with so much time having to be spent in the IDE it had to be at least a suitable experience. After past experience with Eclipse and ADT causing issues due to freezing and crashes the supposedly much more stable AS was worth adapting to. AS also seemed to provide a much fuller auto complete feature, suggestions would be made sometimes within a single letter of a word being typed out, normally correct suggestions. Once used to this process code could be written up slightly quicker making the experience overall more enjoyable and productive. Overall the experience with AS was that it was far less cumbersome and much more enjoyable than Eclipse ADT. While currently there may not be huge differences the ones included were very helpful, the implementation of Gradle [19] meant single lines could be used for the inclusion of dependencies and possible libraries. Easy inclusion like this meant that it was a lot more appealing to try out libraries and see what they added to the project. While Gradle provides many benefits not all were made of use within the project.

A final reason for choosing Android Studio was that after seeing various comments on Android discussion boards it became obvious that it was quickly becoming the industry standard, unsurprisingly due to the lack of support for Eclipse ADT. Adapting to AS seemed the most

logical decision, made easier by its sleek appearance and performance.

2.1.2 Version Control

Version control was an easy problem to solve, as standard GitHub was used to upload versions of code and documents allowing the history of changes to be logged and development tracked. It also meant that in future the project could be open sourced through the changing of the privacy settings on the repository. Using GitHub also meant that even when on the move the developer could check the code base through the website to see if potential solutions could possibly be implemented.

One alternative to GitHub which was considered was BitBucket, BitBucket provided unlimited private repositories and a desktop client which could have been of a fair amount of use. However looking at reviews and the general design, the developer decided that while there were benefits in using BitBucket over GitHub, they did not really apply to the project due to its small size. In future if the project was undertaken by a larger team, or the developer was part of a larger team for a separate project, BitBucket could be heavily considered due to its unlimited private repositories as standard.

Having the code stored off site also provides a range of benefits to the developers. It mostly frees up the developers to not worry about breaking any code, back ups from various times are all available with the ability to roll back to them with minimal effort. It also meant that the code was always safe, once the code had been moved to the GitHub servers its available for the future, no need to worry about corrupt drives or other issues.

GitHub also provides a simple and easy to use graphical interface as an alternative to the command line interface. This interface was sufficient for the project being completed with only a few clicks allowing for the new code base to be committed and pushed to the on line version. While this may not seem a major benefit it still freed up some time and overall made the task of version control slightly more enjoyable.

An alternative to using GitHub for version control and code backup was either storing back-ups externally using flash drives and external hard drives or using the file store given to students by the University. However the benefits provided by these were more than covered by what GitHub provided, it was also decided that there was a minimal chance of losing any data stored through GitHub while it was possible that access could be lost to the file store for prolonged periods of time without prior notice. This has been previously experienced and was not something that the developer wanted to risk.

Without version control it would be possible to completely lose track of what had changed, as well as losing the project as a whole. Storing everything locally is extremely bad practice as any amount of extenuating circumstances could lead to either the loss or damage of the computer that the information is stored on. Furthermore GitHub provided statistics on when most commits were made and exactly what was committed, this allowed for estimations on not only how long a feature would take to develop based on what time had currently passed, but how long a future

feature would take based on its complexity compared to past features. This meant that during development a fairly tight schedule of development was set out and mainly adhered by.

2.2 Overall Architecture

Initial design for the project was very broad becoming more focused over development time. With the Design Specification it is possible to see the first properly documented steps in preparing for the development time within the project, however other initial steps were taken first to help build a design from. At the start of the project a very abstract set of diagrams were drawn up to describe the desired boundaries for the project, this was mainly as a guide for further development but aids in demonstrating the vision of the project from the start. While these were drawn up on paper the digitized version will follow, it can be seen that the diagram is essentially a flow diagram that only involves the Activities used in the Android application. No screen details are given just the links between them, this has changed significantly over development but the basic blocks can be seen here. A link can be seen between Route Plotter and Route Display, this was to represent the output from one being compatible with the other.

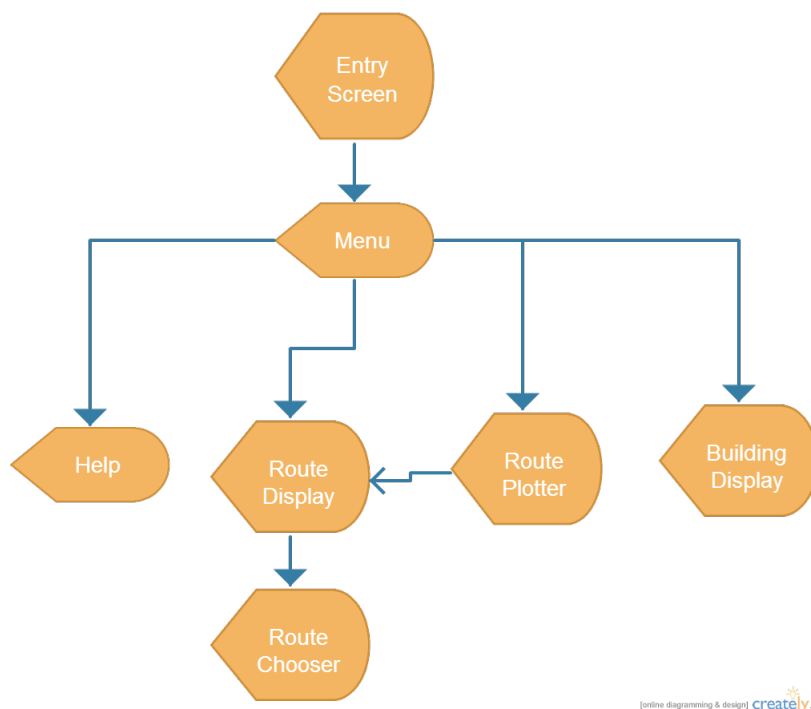


Figure 2.1: Depiction of links between screens including some data compatibility.

2.2.1 Class Diagram and Justification

The final Class Diagram represents what was thought to be the final representation before the Build Feature segment in the process, during implementation some changes were made but the class diagram is a good representation of how the program actually works. Justification of the design will follow with diagram afterwards.

2.2.1.1 Justification

In this diagram the Menu Activity serves as an entry point into the program, in production it was always intended to have a simple screen before this, however for the case of the class diagram representing it is non informative. Within the program the simplest function is the Help Activity, this Activity is created through a button click within the Menu Activity. Within the Activity is an ArrayList of Place, a Place is a simple representation of a location on the planet and is used repeatedly throughout the code for varying uses, in this case it represents a set of potential helpers. Within the Place objects a persons name is the Name variable and their details as the Description, these helper points will be loaded in from a file and then manipulated with methods within the Help function, these methods will load the points, sort them and display the closest one to the user.

One of the possible alternative options that was considered here was the inclusion of a Person object and instead of a Place object have a Building object which could have an array of People. However the choice was made to have people represented as places, this was due to an effort to simplify the programs structure. With Place being used for varying reasons without it being overcomplicated no need was seen to include two separate classes. If all places were buildings it would have been considered a worthwhile effort, however Places are also used to represent points on a Route. As the program stands it is sufficient to have one class represent people and places, including buildings, however in future if more expansions are completed this should change. One area which may change this is the inclusion of porters or open day helpers in the category of 'Help'. Huge changes would need to be made to the structure of the program to support this feature however.

Slightly more complicated is the building display feature which is made up of several classes including the reuse of the Place class. In this case the Place class is used to represent a set of locations relating to a category defined by the user, this ranges from facilities on campus to lecture theatres. Within the Map Activities onCreate method is the code to render different views based on the request of the user, as the Map class is used to represent both route plotting and building display. While the same XML should be used, methods within the Activity should manipulate it, acting as a type of presenter. A choice which arose in the implementation of this is whether it would just be simpler to have two different Activities for the tasks, however with how little they both used and the fact both tasks used a map fragment it was realised it may be just as effective to use the same Activity.

With the Building Display function being used the class Place is now used to represent build-

ings, making use of their 'type' variable. Types are chosen from an Enumeration class which limits the pool of values, while it would have been possible to just use Strings the Enumeration removes the chance of mislabelling a buildings type as the pool of values are the only acceptable choices.

All of the data loaded in is then used within the Map Activity, within the Activity are the methods to handle the representing of a set of locations based on their type variable. With us having a handle to all of the locations the displaying of them is easy to represent with the Google Map [14], using the showByType method and passing it the locations and a desired type. It is best to do this on a button click, likely from a pop up menu. By doing this we can pass the String used in the menu and have that serve as the type, all this requires is correct labelling of the items in the menu. We can then cycle throughout the data objects and show the ones we need as marker objects on the map, showing both their name and description as was set as a requirement due to past user feedback. It is debatable whether all previous displayed locations should be wiped after a new category is selected, or whether a user should instead be able to choose a set of categories for display. To begin with a user will only be able to select one category at a time but this could change with future user feedback.

One of the more complex features which has involved a lot of design decisions has been the inclusion of the Route plotter feature, due to the complexity of route representation on a map many ideal features are simply impossible to implement in a functional way, some of this is down the the unreliable nature of near perfect accuracy from GPS [29]. When a user requests to plot a route they are originally brought to the RouteDataEntry Activity, this is a simple activity which will let the user put in anything they like as the start point, to help with adding new original locations, but limit them to entering a pre existing location on the graph as a destination, thus ensuring a fully connected and traversable graph.

Once the user has completed entering the required data, the Map Activity should render itself based on the users request. In this case, the display button will be replaced with one which contains a menu for the methods required to accurately plot a route. A problem encountered in prototyping is the creation of a Place for the Route object based on a user's location, this has been down to the fact that GPS co ordinates are not always accurate, especially in built up areas [29]. To combat this, it has been decided that a point will only be added to a Route on the user's command, with the user's location being displayed on the Map object. An alternative option would be to have the device log a point automatically every five seconds, however this caused many 'off' locations to be included. With the inclusion of these off locations, an almost jagged path was produced compared to what user defined logging times provides.

To aid with the visual representation of a route each time a user logs a point after the first a poly line [18] should be used to represent the traversed path. This should be done using the connectPoints method which takes two Place objects from the Route. Having a visual representation provides much better results than having a user blindly plot a path, it also means that combined with a user being able to see there location they can accurately tell the line that will be drawn before it is. A small incremental counter should also be included within the XML file to log the steps encountered on a route. Once a user has finished logging the path they wish to

add to the graph they should be able to print a program compatible file. This is handled by the `printFile` function which takes the completed `Route`, within this a variety of functions are called resulting in a final printed file. Methods called from within the `printFile` will be `calcDistance` to return the final distance between points and the `calcGrade` function to return the final grading of the route. Grading will be done by normalizing the total distance and steps to the same scale to figure out there 'rating'. A set of boundaries will be used to define the rating.

One major difference between this design and the previous included within the Design Specification is the change of what the `Map` object considers its route. Initially it was implemented so that the `Activity` stored information considering the grading, start and destination along with an `ArrayList` of `Place`. However it was recognised that this is a bad representation considering we already have a working `Route` object used within route finding. If the object represents a route in one part of the application but not another it could be considered confusing for future developers. An additional change is the inclusion of the data entry `Activity`, this was an oversight in the initial design and something that has been fixed in this version.

Route Finding within the program will be by far the most complex feature to implement. This is down to a variety of reasons including having to implement a search technique which performs reliably. Before any graph analysis is done the user should select what graph they want to use by making a choice between what is represented as two separate features. They should be asked if they want standard routes or easier routes, shown as a 'no step' option. While this appears as two separate features all it should realistically change is the graph used for search and traversal, this also opens up the possibility of having more than two graphs in future, however more complex and effective solutions could be included.

Representation of a `Route` is something that a fair amount of thought has been put into before development, due to the importance of being accurate. Within the proposed system a `Route` is essentially details of two connecting `Nodes` on a graph and the link between them. As it stands the link between the nodes is represented as a set of GPS co ordinates which when plotted in order describe a set of destinations which lead to the final location. By doing this we help represent curves in the links between nodes. In this design a links weight is represented as a grading which is set based on criteria laid out in the `Route Plotter Activity`. Searching could be completed based on route grading but may be outside of the scope on this project.

After the choice of what type of routes they want returned the user should be taken to a `Map` object with a button which will take them to the `RouteChoose Activity`. This `Activity` will contain two expandable list views populated through a file within the relevant assets folder. Another option would have been hard coded values but these do not provide the expandability felt necessary. Once a user has chosen their start and destination they should be taken back to the `Route Display Activity` to see the route, before this happens the route will be found using a search technique within the `Route Choose Activity`. This method will return what is actually an array of `Routes`, by doing this we have segments to our routes and can manipulate its colour in segments allowing for the display of different colours based on the difficulty of small sections.

A choice which arises here is exactly how to search for a `Route` through the graph. Currently it has been decided that a simple Breadth First Search [20] will be implemented. By doing this

we can move through the graph and continue until we have found a small route which ends in the destination and build backwards from there. In future, if the graph expands to be quite large, it is suggested another more complex search technique be included if time is not found within this project.

Once a user is returned to the Route Display a set of methods will be run to handle displaying the routes. First the Route array will be pulled out of the Intent returned from the Chooser Activity, this will then be set using the setRoute method which will in turn use the loadFile method to get access to routes stored within the programs assets. As previously stated the Assets folder accessed will depend on the type of route finding the user has requested. A small window should also display the steps on a route and other information including the distance that has to be covered.

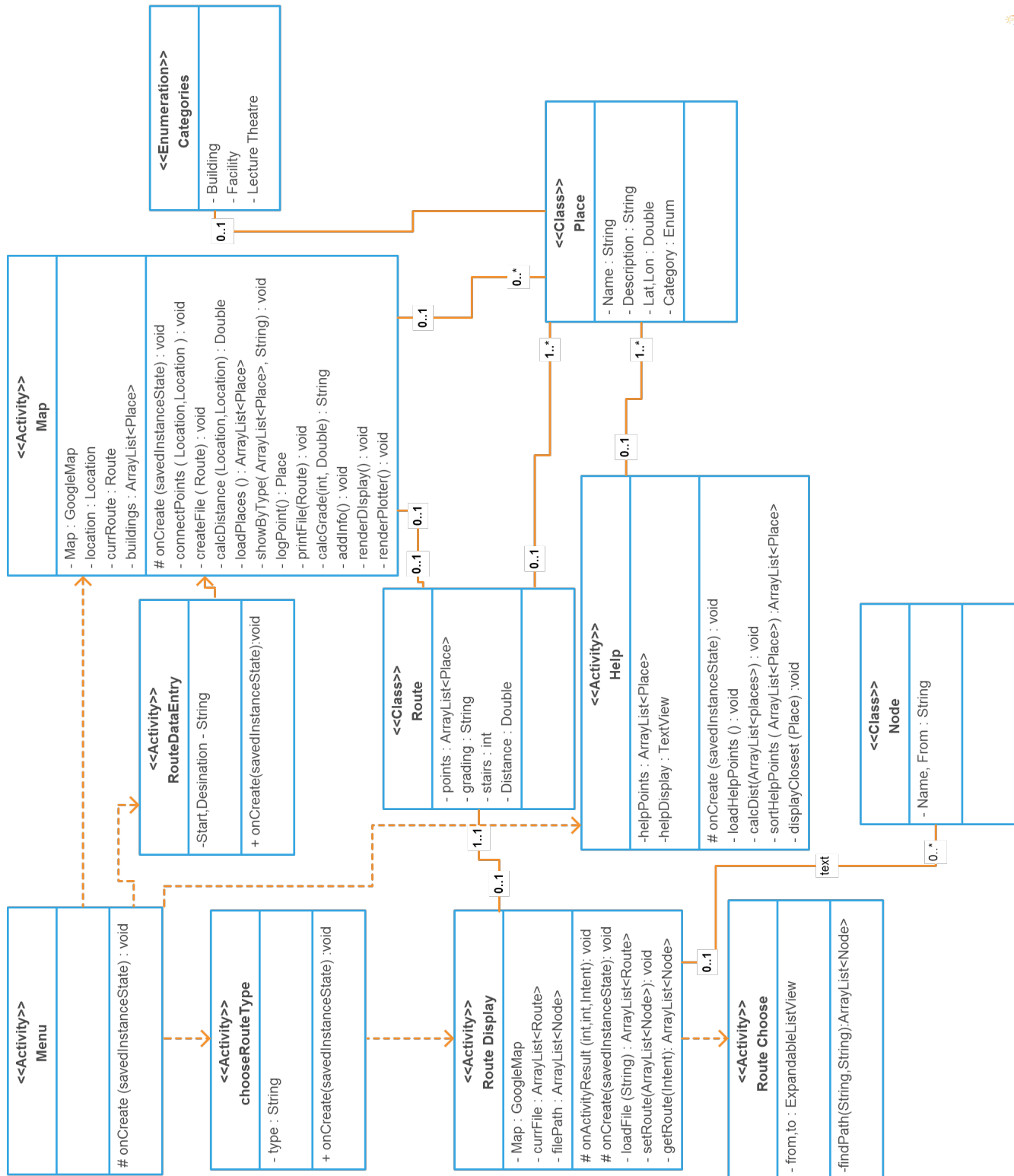


Figure 2.2: Final Class Diagram depicting the plan of the implemented system in UML notation

2.3 Application Flow

With this application most users will be first time users, it is not likely it will be repeatedly used by someone after they are used to the campus. With this in mind it becomes clear that good UI design is important, along with logical flow of the application. A basic flow of screens can be seen in 2.2 however it describes a very broad series of events. In this section a detailed flow of data and screens will be detailed to help with the implementation phase. The diagrams following and the textual descriptions should be used as a framework to build around.

2.3.1 Overall Justification

A similar diagram to this can be seen in the Design Specification, section 4.1, however this revised version details updates including additions of screens and specific details on what should be contained within some algorithms and the alternative considered.

One of the changes included within this diagram is the inclusion of the Route Data Entry Activity, this is used for the user to enter the start and destination of the route they are plotting. While this obviously needed to be included it was not within the original diagram, a major decision revolving around this screen is exactly where it should be placed within the application. It is viable to include the screen between either the choice and the plotting screen, or have it used after the user chooses to print the route they are currently plotting. It has been decided to include this before the user plots anything, this is due to the idea that the user should provide broad details like the start and end before adding information about the detailed route. It follows a logical progression this way.

After these details are chosen the *addInfo()* and *renderPlotter()* functions are run. These should actually be run in the *onCreate()* within the Map Activity. This way the information added in can be accessed from the past Intent to detail what to render and what to name the route. This also applies for the *renderDisplay()* method, the decision on which should be run is actually made in the *onCreate()* it just makes more sense to draw it as happening before the screen fully exists, which is technically true as the *onCreate()* method is what is creating the screen.

Once at the Map activity various methods can be run depending on what was rendered. If the Building Display screen is shown the user can only select the type of a building to show, one type at a time. This has been a feature repeatedly mentioned, the hard decision to make is whether users will want to see multiple categories at a time. Testing after implementation will have to be completed to gather new user feedback as little is mentioned from the original user feedback. With it not being an issue previously it is considered to not be one currently.

If the user has selected to be able to log a point various possibilities become available, the main method being *logPoint()* which allows the addition of a point, this should use a Location Manager to retrieve GPS co ordinates and add them to both the map and a Route object for printing later. Once a second point has been added, and continually from there, lines should be displayed to visualize the route. This is easily done using a poly line on the Map object and the

co ordinates already retrieved. Other options include *cancel()* which will reset the Route object and call *Map.clear()* to provide a fresh start. Another decision was made here, it would be possible on cancelling a walk to bring the user back to the data entry screen, however it is being assumed that the user will have got that information right and if not will return themselves, forcing the restart of the whole process is unnecessary. A smaller method is actually the *addStair()* function, this should be run on the change of a value in the stair counter. It will simply update the value of the stairs in the Route object.

Finally the *saveRoute()* function could be called, to print the Route object to a file compatible with the Route finder. Printing the file is fairly simple, it just has to match the format set out for Routes which will be discussed further on. The file should be printed to the devices memory [9] for access later by the user. A major decision made here, discussed previously in the original Design specification 2.1.2 and here in 1.1.1, is whether to have it uploaded or just saved, in short internet on campus is not always available so the decision has been made to save locally to avoid problems relating to communications. Saving the route should also provide a grading, this will be decided through a small formula which considers total distance covered between points and the steps added on the route.

Most information on the Help section of the application has been discussed previously in Design 4.2.1. It is a fairly simple process, loading in help points and finding the closest using the *greatCircle()* [30] function. However a further design decision has been made which is hard to represent visually. This is if the value should be updated past the first run, whether the help screen should change while the user walks around with it open. However this idea has been omitted due to the problems and uncertainty it causes, if the method is run every ten seconds for an update, poor GPS signal could mean that when around the boundary of a help point the wrong one is shown occasionally causing confusion. To combat this it would mean taking a set of co ordinates over a time span and finding an average using them. However it has been decided to just use the first reading, GPS is mainly reliable [6] and the problem revolving around a boundary is unlikely to appear, most people will be well within a building when they need help, not outside and close to another with a help point.

By far the most complex feature in the application is the route finder and a great deal of thought has gone into how to configure the set up to fit into what the developer thinks is possible during the design phase. While there are many ways to implement what is done, an affecting factor was always the extendibility of the project. At completion the project should have many open ends within it, to allow for the addition of further graphs, locations and make the editing of menus easy.

In the flow diagram the user is initially sent to the Route type activity, this should allow the user to choose between two separate modes, the step free mode or the standard mode. Step free will be a graph that has been built that avoids stairs. While it would be possible to build a more complex base graph and further change the algorithm the problems these implementations pose could create time restrictions on other areas of the project. Once chosen the type will be saved by the Route Display activity.

Once at the Route Display activity initially just a map of Aberystwyth and the campus will

be shown as no route has been selected, the user should have to click a button to be taken to the Route Choice activity. On this page Expandable List Views should be used to simplify the route selection process, these should then be populated from a text file to help with addition of locations further on. After the user has chosen there locations, feedback should be provided on the page of exactly which locations are to be currently used. Giving feedback to the user through the entire project is key.

Once these have been chosen the Breadth First Search will be run to find the path, further details provided in section 2.5. What is likely to happen is that the algorithm will find a set of paths rather than one, this set of paths will build to link a full route from the start to the destination. Implementing the system in this way fixes the problems with the maintainers having to add hundreds of routes just to add a new location, in this version all that has to be built in is one new route connecting the graph to the location and vice versa.

Plotting the route is seen as a separate task to finding it, finding a path through a graph will end with us just knowing the last accessed node which opened the destination. From there we need to plot backwards. Moving through the nodes based on which location opened another should allow us to move backwards from the end node to the start. This also gives us the smaller routes to plot to display the full route to the user.

Painting the route is then fairly simple, we know which location opened another so we can just search for that locations file and paint the single route to the next one. Repeatedly doing this for each node until the destination should lead to the full route being displayed to the user.

Some consideration should be made to logging a users location on the route and showing it to them, however this can be enabled through the use of a Google Maps API method called *setMyLocationEnabled(boolean)* [16] which should give us the required effect. From there this can be customised to provide the user with more information than a dot.

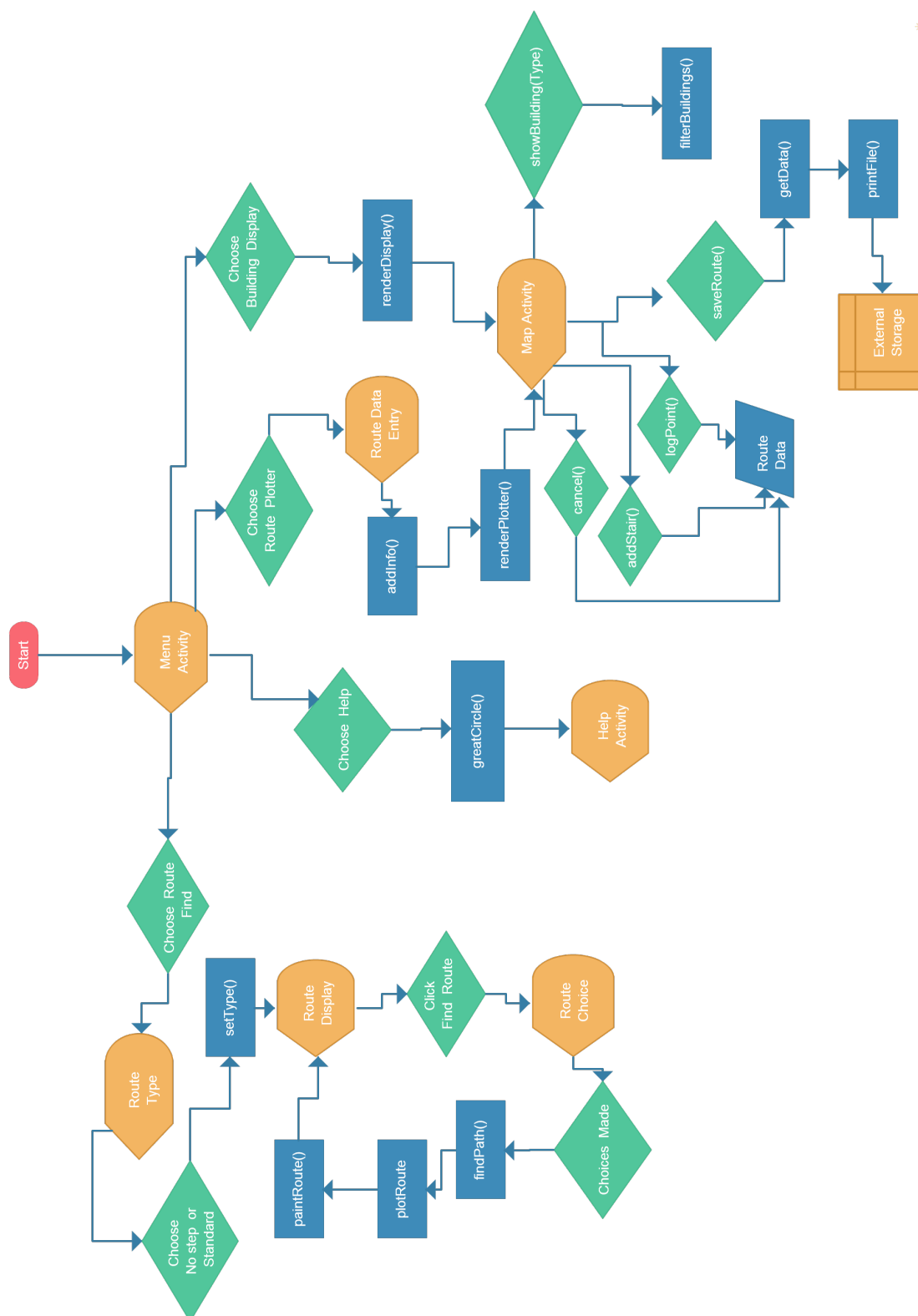


Figure 2.3: Current Flow Diagram depicting the flow between screens and processes within the program

2.4 Sequence Diagrams

2.5 Major Algorithms

Throughout the application there are thought to be a range of major algorithms, most of these relate to the finding of a route and then the visualizing of that route. Currently the algorithm is expected to be a version of BFS when required to go past a single connecting node. Past designs of these algorithms can be seen in Design Specification 3.3. Following are the currently expected algorithms to find routes, calculate distance and read in the file format.

Initially route finding was shown in a single algorithm which is mostly correct, however due to the files containing all of the routes from a point the process has been changed. Initially a check will be made to see if there is a direct connection between start and end, if not then the BFS algorithm will be called. To start, the algorithm should look as follows, this should check the original nodes connecting nodes. It logs the connecting nodes and if none are the destination they will be passed to the *deepFind* which is the BFS.

It does this by using a boolean which is changed on a route being found. At the start of the BFS it should be noted that the start node, which will be stored in the activities variables, should be added to the visited list to avoid the nodes checking that.

Find Initial Path

```

1: startRoutes = read(start)
2: found = 0
3: ArrayList routeEnds
4: for startRoutes do
5:   routeEnds add route - > end
6:   if route - > end = end then
7:     Plot Single Route route
8:     found = 1
9:   end if
10: end for
11: if found == 0 then
12:   deepFind routeEnds
13: end if
```

The deep find algorithm will initially be a BFS, a mostly correct version can be seen within the Design Specification however it does not account for being passed the starting nodes. The actual implementation should look close to the following pseudo code.

In this example you can see that the start node is set to visited and the nodes to be cycled through as those passed in as a variable. While the algorithm looks relatively simple set like this, due to having to implement loading from the devices memory and other functions it is expected to take time to develop. Initial research has been conducted on loading.

A key line which helps in a later algorithm can be seen on line 11. This helps us move backwards from our destination as we can tell which connecting Node each Node was opened from, essentially plotting the path backwards by tracing the algorithms steps.

In future it is advised that more complex search algorithms are considered, including A* [2], to try and find a more efficient algorithm as on a large graph BFS could cause a small delay on older mobile devices.

Find Route

```

1: start set Visited
2: Nodes = passedInNodes
3: while !route found do this
4:   for Nodes do
5:     if Node has been visited then
6:       do nothing
7:     else
8:       set Node visited
9:       get NodeRoutes
10:      for NodeRoutes do
11:        Set Opened from Node
12:        if NodeRouteEnd = Destination then
13:          set route found true
14:          call Plot Path (visited)
15:        end if
16:        if NodeRouteEnd not visited then
17:          Add to Nodes
18:        end if
19:      end for
20:    end if
21:  end for
22: end while

```

As can be seen in the above algorithm the visited nodes are passed in as a variable to the function that will plot the path, this way access is provided to the information about which Node was opened from where, allowing the function to work backwards though the Nodes. This is possible due to the fact that as soon as a single route is found that connects to the end no more are looked for, by doing this we guarantee only one path being painted. However this plan does not work if the search finds multiple paths, something that needs to be noted for future development.

Due to the nested loops the algorithm should always paint all required routes.

Plot Path

```

1: VisitedNodes = visited
2: search = destination

```

```

3: for VisitedNodes do
4:   for VisitedNodes do
5:     if search = NodeName then
6:       Plot Single Route 9Node – > name, Node – > from
7:       search = Nodefrom
8:     end if
9:   end for
10: end for

```

Pseudo code for plotting a single route and then painting it can be found in the Design Specification sections 3.3 and 3.4. These have not changed through further design and are expected to suffice as they are. This also applies to the saving of a route file algorithm, found in section 3.5. However, after research it has been noted that external memory is not always provided and as such further checks should be completed [9], following example code from both the Android API and other sources [27] it is expected the following code should suffice.

Check State

```

1: state = getExternalState
2: possWrite = false
3: if state = readOnly then
4:   Cannot create files
5: else if state = readWrite then
6:   possWrite = True
7: else
8:   Something is wrong, we only have one desired result which we have checked for
9: end if
10: return possWrite

```

An algorithm previously overlooked due to an initial assumption of it being fairly simple was that of the file writer. After further research which resulted in the memory check algorithm [27], it has been decided that its process should be noted due to the various ways it could be completed. A further decision which has been made is to include the saving of a file into the file writer function, this way when a user initially goes to save a file the folder will always be present. Initial design should be as follows.

Save File

```

1: root = getMemory
2: MyDirectory = root/routes
3: if My Directory exists then
4:   Do Nothing
5: else
6:   Make Dir – > MyDirectory
7: end if

```

```
8: New File- > root
9: out = OutputStream- > File
10: out- > Route- > Start
11: out- > Route- > End
12: out- > Route- > Grading
13: for Points do
14:   out- > Pointi
15: end for
```

2.6 Interfaces

User Interface was an area in which the developer had little to no experience, while basic GUI's had been developed before they did not truly consider a users habits and motivations. From the initial design specification to the beginning of implementation, research was completed on both design guidelines and the options provided through Android and its use of XML.

While UI was seen as a less important task than the actual underlying code, a set of basic rules were set out from the start, taking influence from other applications and the Google Material Design outline [15]. While the material design outline was not adhered to fully it provided good insight on the vision Google has for the Android platform.

- 1 - Make best use of white space within design, make it act as a barrier rather than having to add an actual barrier in.
- 2 - Bold colours will help cause distinction between background and active UI members.
- 3 - Theme should be carried throughout pages as best possible, little to no room for exceptions.
- 4 - Develop a system which supports the same experience throughout devices. XML allows for relative layouts so no fixed measurements mostly.
- 5 - Interactive elements need to be responsive, users need feedback on their actions. This can be textual or a change in colour. Noise is not practical.

2.6.1 Colour Theme

Colour theme was something that the developer felt was key to a successful UI, without it a good layout would go to waste. Due to this, several palettes of colours were chosen at the start for trial upon the full completion of the application. Initial design highlighted a yellow and purple theme as the likely choice, due to matching the universities website. However, as design furthered and prototypes were developed of basic layouts, it became obvious this may not be the ideal theme. These themes were both developed internally and taken from on line resources. The following were chosen due to them being seen as palettes that would not only provide good distinction between members of the UI but provide an easy to view screen. Using many bright and clashing colours could possibly create a visually offensive UI which is something to be avoided. By using safe, strong colours a higher chance of success was predicted.



Figure 2.4: Theme Possibilities that could be applied to the application

As visible from the above diagram, the palettes were based on what seemed to be sequential colours, and while no actual maths was applied to come to this decision it was visually apparent. However this provides the developer with a range of possibilities, with the shades being far enough away from each other, text of one colour should always be visible on another due to the lack of overlap. It is advised during development though that text of one colour be on the background of a colour at least one away.

For example, text from the middle of the first palette should only be applied to backgrounds of the two ends, while viable to put on the two connecting colours the impact of the distinct colours will be lost slightly.

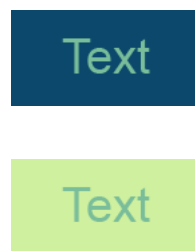


Figure 2.5: Demonstration of coloured buttons using the possible colour schemes

As can be seen in the Theme Possibilities above, the principle of using five progressive colours should provide a theme that is not only easy on the user's eye, but enables a distinction between UI members and blank space. With the above three options to be tested, it is likely that either 1 or 2 will be the final theme. With 3 being viable, the colours seem to be bland, while possible with the right arrangement it is suggested more effort be put into a functional theme built from 1 or 2.

A similar palette to that of 2 can be seen at outlook.com [21]. In this design, the colours are fairly similar with white acting as the background and then a colour close to the middle acting as interactive elements. Distinction between the two can clearly be seen, and as such, they stand out as having a purpose; it also helps with the clear display of text whilst removing the clutter multiple colours may cause. In the design, responsiveness is achieved through a brief colour change in the element, something that should be considered within this project. Other considerations to be made in terms of colour design include the implementation of themes for colour-blind users. However, this is a stretch goal and unlikely to be implemented.

Mock up screens of the application with the prospective colour schemes can be seen below, with these it is clear to see why 1 and 2 have an advantage over 3. It should be possible in future for the user to define a theme from a set of options, not only will this help users who make frequent use of the application but it will provide a way for the colour-blind themes to be implemented without a separate application.

By completing tests on actual users not only can we plot out the decision making and adapt around it, we can get feedback on clarity, simplicity and overall operability. A fully working application that a user can easily navigate through is one of the prime objectives of this project.

2.6.2 User location

Displaying the user's location to them is seen as a key element to route finding, displaying a route is only so much use if a user cannot see where they are on it. While the solution to this problem has already been discussed, some additions can be made to it. Mainly the addition of custom graphics to benefit the user.

This is something which is covered on the introduction page to the Maps API [14]. From

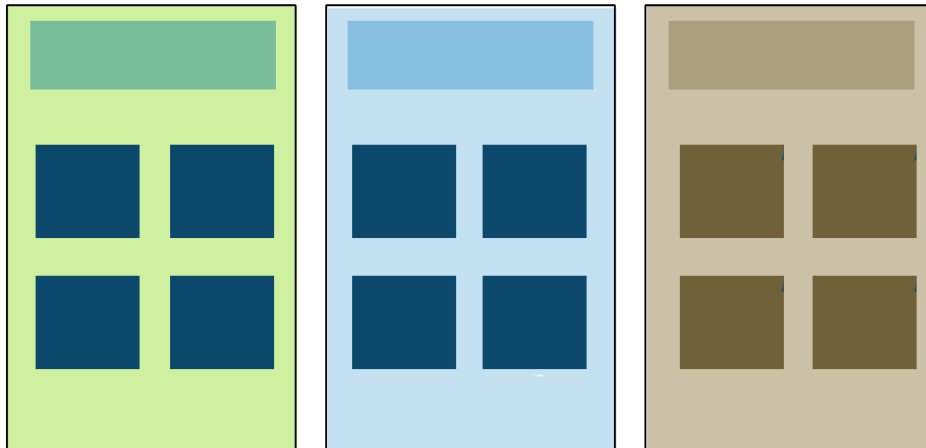


Figure 2.6: Example Themes demonstrated as to how they would look on the application

that and some basic research it is possible to see that custom graphics and a movable camera are both possible. Giving the developer to implement an almost sat-nav feel to the application.

As the method stands the arrow used to display the users location is small, some research should be completed on whether it is viable to include a custom graphic.

2.6.3 Screen Designs

Some initial screen designs can be seen in the Design specification, while rough their overall display will not be altered too much. However the theme used and some of the options made will make it hard for the screens to be implemented as they are on various screen sizes, especially on the menu screen.

In the original designs the Menu Activity is shown as a screen with a list of buttons which would then link the user to the respective Activity. However, on multiple screen sizes this poses problems due to the list looking out of place. Due to this the decision has been made to implement a grid system on the main menu, by doing this we can make the segments relative to the screen and have them fill it. White space should also be left between the segments to provide clear non verbal instruction that the buttons are separate and interactive.

Further changes include the addition of several screens which were not identified during the start of the project. First of these being the Route Type Activity. In this activity the user is given a choice between the two types of graph they can choose to find routes through. Following is a

representation of what should be a fairly simple screen.

Choose Route Type

No Step

Find Routes without any steps on them, easier to travel by but can be longer.

Standard

Find Standard Routes, includes steps and normal entrances.

Figure 2.7: Example design of the the route type selection screen, final will be either identical or a close variant.

Another addition to the set of activities will be the Route Data Entry activity, this will allow the user to enter information about the route they are planning to take. To accomplish this the user should enter the start and end points, from there they can then move onto the actual logging of the route. Again a simple screen due to it being two text entry boxes, only restriction being that of the theme.

Enter Route Data

Enter Start Point

Enter Destination

Go

Figure 2.8: Example design of the the route data entry screen. Will only be two text entry boxes and a button to proceed.

Chapter 3

Implementation

3.1 Location Based Help

Throughout the design phase of the project the Help Activity is referred to as most likely to be the easiest feature to implement, when it came to implementation these assumptions were mostly correct. Due to past work the developer was already aware of the Great Circle function [30] which saved some time.

There are a fair amount of changes to the implementation shown in the class diagram, some of this was done to simplify the process and stop code being used twice in two different places. The largest change is that the *calcDist* function shown in the diagram has been pulled into its own class, this was done due to the route plotter also needing a function to calculate distance, it was best practice to only have the code exist once in the application.

Another change is the removal of the *loadHelpPoints* function, while loading from a file is completed throughout the application in this case problems arose which made the process seem to be a waste of resources. While looking for information regarding people who could be contacted for help, little turned up. Details relating to three buildings were found so for the current iteration they are hard coded.

A small change within the program is also what the *sortHelpPoints* function returns, in the diagram it returns the sorted array. However in actuality it now returns the index of the closest location, this removes some stress on the device and is overall simpler yet still effective. A further small change is the removal of the *displayClosest* function. In actuality this was not needed, all methods are called in the *onCreate* due to the simple nature of the task, once the right index has been found what the method was meant to do just took a single line of code so the inclusion of this method was seen as unnecessary.

The location based help function was actually simpler than initially thought, due to its very small size. A users location is easily found using the Location Manager and from there its basic object manipulation mostly. Implementing the great circle function took some time until a fitting

one could be used, taken from Stack Overflow [28]. Other proposed solutions seemed overly complex for what was being considered.

3.1.1 Updating Help Location

One idea that was experimented with was having the activity update when it moved into a new 'area' of help. This was done by repeated updating of the user location and rerunning of the methods which calculated the closest position. While it works in practice the major issue was what was seen as a good degree of uncertainty in the calculations.

While the Great Circle function works well, with its results being tested against measurements completed on Google Maps, reliability of the GPS co ordinates were not as certain. While co ordinates give us a very close estimate of our position, they are not always perfect. After some research it was clear that civilian GPS while in good conditions could return results within a metre [29], other factors that affect the accuracy are present on campus. Looking at figures, most GPS queries return accuracy to within 3.5 metres [6]. However the impacting variables of receiver quality and signal blockage are not something we can plan for, due to these reasons a series of decisions were made.

As seen on the official GPS site [29], with the boundary between some buildings being so small, an error of 5 or so metres, something that has been seen, would provide false readings. Due to this the implementation of updating values on the Help page was omitted, the text displayed clearly displays the building the help is for, if this is wrong a button to update values has been included.

3.2 Building Display

During implementation a large amount of change was made to the way that the Building Display worked, a lot of this was down to it sharing an Activity with the Route Plotter. Initially this was done as the activities used similar resources and seemed small enough to not cause too many issues. However into development it was decided that the two activities really should have been separate, if nothing else than for clarities sake. However this was not decided until the features were complete, so as the project stands the screens have not been split.

3.2.1 Removal of Render Functions

In the program the largest change is the removal of the *renderPlotter* and *renderDisplay* functions. In past documentation these were said to be the methods that would actually render what was needed, however these have been removed and the problem handled in the *onCreate*. This is done by the press of either the Building Display Button or the Route Plotter button initially sending through a string which represents the users desire. From there the *onCreate* is split, if the user has said they want to plot a route then a different part of the method is entered,

in both cases the visibility of a set of buttons and other interactive elements is set. By doing this we can view the activity as having two *onCreate* methods, its just that they are nested within the same parent method.

3.2.2 Other Changes

A change seen previously is the removal of the *calcDistance* function from this class as well, it now relies on the class that handles all of the distance algorithms. A similar change is the removal of the *loadPlaces* function. Loading of places is handled by the actual Place class, called locations in the application. By being moved away we keep the code in this class purely related to the displaying of locations, removing any blur between the classes. *showByType* is implemented in the suggested way by the class diagram, it is called using the *setOnClickListener* functions used in the *onCreate* function. These listeners set the behaviours for an interactive element in the design, in this case the *showByType* function is called with the buttons type being passed in as the string variable. For showing Lecture Buildings the String Lecture is passed in and the array searched for any location that matches. *showByType* was actually initially implemented in each *onClickListener*, while no real problem for the user it meant the method was implemented for however many types of building there were. When the design was finally adhered to it vastly simplified the process and created a much better class in terms of code quality. This also means that in the future any addition to the type of buildings will only need a few lines of code instead of the repeated fifteen or so which were initially being used.

Throughout development other applications and documentation were reviewed which showed the use of custom markers for buildings, due to this the feature was also included within the application.

3.3 Route Plotter

Route plotting was an area in the application where a fair amount of issues arose during implementation, while the feature was successful many hard decision had to be made and the result of those will have to be tested. With once again using GPS co ordinates for the positioning, problems arose with accuracy and exactly how a user would want to plot a route. Most of the class diagram relating to this feature is correct, the only difference is the movement of the log point method into the *onClickListener*.

3.3.1 Plotting of points

Plotting of points is an area which had similar issues to the updating of values in the Help feature. Mainly the testing of automatic point logging, this turned out to be a much bigger problem than in the help section. With the error in GPS co ordinates sometimes being over 3 metres [29] the automatic logging of points is not something which could be trusted to return viable results.

Tests using automatic logging returned results, which when visualized, returned results similar to below. Any stop in a users movement would cause problems as any change GPS made would then cause a small difference meaning that it looked like the user was taking an erratic route.



Figure 3.1: Example of issues posed by automatic logging of points when route plotting

As can be seen from the pictures above this was not an ideal way to implement the feature, what is now done is a point is only added when a user asks. Having the feature implemented this way means the user is free to explore paths, yet not plot them. Providing what is seen as a more user friendly experience. Another benefit found was that by using the Google Maps *setMyLocationEnabled* [16] the user would be able to see their position, however the benefit goes further than that, in locations with good signal the point is plotted exactly at the displayed position. It is thought the reasoning is that both Google Maps and *logPoint* method are both using a Location Manager which will always return the same results to both as its running on the same device.

3.3.2 Visualizing and Cancelling

Visualizing a route was implemented in the way detailed throughout the application, poly lines are drawn between each consecutive point causing a route which displays the users position throughout. Within the application every time a point is logged, except for the first time, the *paintRoute* function is called, it takes the latest and previous logged point and simply paints a line between the two.

An implemented method which was not expected to be within the class is the *showDialog* method, this is run when the user wants to cancel a walk. Absence of this method in the class diagram and other specifications is due to what was a lack of research on the possibility of a

dialog box. It was wrongly assumed that the displaying of one would be smaller than it actually is, however the method is fairly simple. On the user wanting to cancel a walk, all points are wiped and the map cleared, on choosing no nothing is done.

3.3.3 File Saving

Saving the plotted route to a file was a task which took much longer than expected, this was due to a misread of the documentation. Through research it was found that an Android device typically has two types of storage, internal and external. It was assumed that external storage would only be possible with an SD card in the device, this was the first mistake which caused the long delay. A second compounding error was the assumption that saving to the internal memory would make the file accessible by the user [9], something which is only possible in cases where a device is rooted. These compounding factors caused the process to take a length of time which simply was not expected and set the development off by around two working days.

During this time it was assumed that the file was simply not being created, however in actuality it was, it was just not available through any file browser, only through the application itself. This was due to the assumption that internal memory was accessible by anyone at any time, in reality files saved by applications to internal memory are only available to that application. Due to this, looking through the file system for the relevant folder was of no use, it was there but hidden. To combat this various techniques were researched of saving files without having to rely on an SD card as not all users would have them.

One possible method that has been mentioned before was the uploading of a route so the user could retrieve it from any web connected device. Legacy code from previous projects was used to send a JSON which included all of the relevant file information, using the LogCat feature in the Android Studio IDE [8] showed that the data was being sent out, so the development could continue.

However, with further reference to some Stack Overflow answers to relevant questions [27], it was discovered that external memory is a fairly inaccurate name for what the memory actually is. In reality external memory represents one of two things, that an SD card is present or the device in question has inbuilt memory to store files [9]. With this discovery the viability of storing files locally vastly improved. As most devices are present with one or the other the final decision was that saving to external memory was the solution. With Google themselves trying to set the trend of removing the necessity of SD cards in Android devices it should be a smart decision for the future as well.

3.3.4 Writing to File

Implementing this feature was simple once the problem of saving a file and accessing it were resolved. This functionality is provided by the *printFile* method. This method takes the Route object and uses its variables to print a file that represents the route plotted. It is printed in such a way that it is compatible with the route finding side of the project.

Printing the file is done with an *outputStreamWriter* which takes a String and prints it into the file, each time a *write* is called the writer uses a new line. Formatting for the file is as follows

Start

Destination

Grading

NumberOfPoints

Points

Destination

Grading

NumberOfPoints

Points

The format was dictated during the development of the route finder, having the file formatted as such allows for an easy implementation of a file reader. With the first 4 lines being static throughout files and the next being decided by the fourth line the code is easy to implement. From there we can see that another destination is listed, the above file contains details on a Node which connects to two others and the path between them.

Once the file is saved it is up to the user what they wish to do with it, this could range from testing of a path to full implementation into the graph. While a printed file links a Node into the graph, changes still need to be made to detail the path to the Node for the use of other Nodes. This means changes need to be made to at least one other Nodes file for the Node to be fully linked. While this is not ideal there are very few ways around it, one possible way considered was to also create another file for the place the path connects to, which would contain the new path. However if multiple new paths were implemented at once, each time the new file for the Node would differ, addition by hand is not ideal but at least lets the user keep track of which files need what adding to them.

3.4 Route Plotting

Route plotting was the key area of the application, this was due to the poor implementation of it in the original Access Aber [1]. With so much feedback from the original version it was easy to set out a set of guidelines, some of which have been mentioned in the previous Design Specification. To begin with in the implementation phase a set of guidelines were set out which, if met, would cause the feature to be seen as a success for the current time line.

- 1 - Adapt the Route selector to be two selectors, a start and destination, rather than one which contains all possible routes.
- 2 - Implement an extendible graph that represents the campus and building locations.
- 3 - Have the route display gradings for the route, if the route is in segments display each segment in its respective colour.

- 4 - Give the user the opportunity to take a route with no steps, ideally this would be done on the same graph as standard route finding if possible.
- 5 - Display basic information about the route, stairs and distance ideally.

These five criteria were seen as fairly reasonable targets, with the only worrying task being the no step feature, something which was assumed to be difficult. Item 5 was seen as a stretch goal as it would require the editing of the file format and as such a fairly large amount of code from some small prototypes and the route plotter.

3.4.1 Route Object

Initial development time was used to implement the route object, something fairly simple. As an object itself the Route contains the information relating to its start, destination and the points along the route. This means it contains an Array List of what are called locations in the application, labelled places on the class diagram. This implementation provides us with all of the information needed to find a path through the graph. When a Node is analysed, what is really happening is that a file is being loaded in which creates a set of routes, each are checked for where they end. If the route ends at the right place the solution is found, if not its end place is added as a Node to be searched next.

Paths between nodes are stored as a set of locations, this is an object used repeatedly throughout the application, a 'locations' object contains a range of information, most importantly the co ordinates of the point. For route finding the rest of the information possible is irrelevant, other variables are used elsewhere. Having this one object which describes a location was a real benefit during development, it saved creating classes for what were meant to be different types of location like buildings. Due to this locations can have names and descriptions, it is just that they are not important for route finding.

3.4.2 Implementing Graph

Throughout development a range of issues were encountered, one which changed the layout of the application was where files could be opened from. Access to the *Assets* folder which is where the files are stored is only given to Activities within the application, near the start of development the desire was to move the route finding out of the Activity and into their own class, this still would have been possible but would have caused code that solves the same problem to be split up through two classes. This was an unwanted confusion, due to it all route finding is now done in the route display activity, having been moved from the Route Choose Activity.

Each file loaded in represents a Node and the connections to the other Nodes around it. By loading in the whole assets folder it would be possible to visualize the whole graph and the connections. One of the key problems encountered when implementing the graph was that once the search was complete, which will be discussed further on, routes would often double back on

themselves as that's just how the Nodes were set up. After some research on how others had implemented representing real world maps as a graph it became obvious why.

In road travel there are decisions to be made beyond locations, routes are not planned from location to location which is what was currently being done. Junctions have to be considered when implementing a representation of the real world, these became known as 'Points of maximum traversal' throughout development. These points were where a majority of paths would have to pass through, implementing them stopped the double back issue. Campus mostly consists of some key roads, mainly the one which contains IBERS, Edward Llyd and the entrance to the Llandinam building. With this in mind there weren't too many junctions which had to be implemented, the two key junctions of travel are the area around the Students Union which links to both Hugh Owen the Union and further on most accommodation as well as the junction at the bottom of the hill, a point which most routes pass through. This connects to the row of buildings where most lectures take place as well as the sport facilities and the National Library. An issue junctions also resolved was with routes doubling back on themselves, while the route was found the representation of the real world was poor so the resulting route was also inherently poor. The issue is represented below.

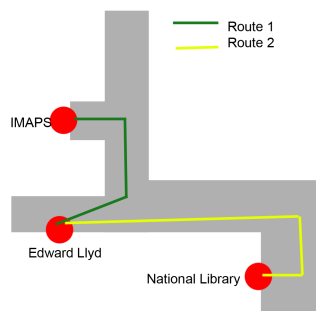


Figure 3.2: Example of route issues caused by the lack of junction consideration in the graph.

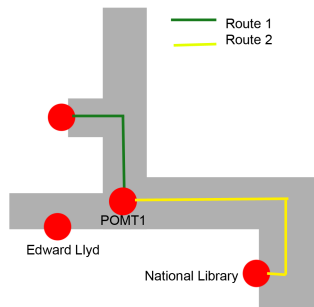


Figure 3.3: How the consideration of junctions improves route finding through a more accurate representation.

3.4.3 Step Free Graph

Implementing the step free graph was a fairly simple process in its current version, though it is an area that could do with improvement. Currently upon a user asking to find a route they are asked if they would like their route to be step free, if so the same activity is loaded but the route finder logs that the user wants a step free route. By doing this the user has essentially set which graph will be used. Currently the step free and standard graph are completely separate, contained within two folders nested in the assets folder. During searching the algorithm uses a keyword passed in to determine the folder files are being fetched from. This applies to the expandable list views in the route choosing activity as well. In future it would be beneficial to have one consolidated graph, including full route information and different routes between the same points allowing for path finding with variables.

3.4.4 Search Algorithm

Initially fairly little research was done on how to build the search algorithm, throughout documentation it had been referred to as a Breadth First Search but when it came to implementation it was thought that creating any search would be a good base to build upon. However it quickly became obvious that what was being attempted was a BFS [20] and as such references should be made on how to implement it properly first time. A large amount of delay came from implementing the Search algorithm as not only was it a search, it was a plotting of a set of routes and representation of them on the Map object.

After the basic search algorithm had been implemented the initial issue was a lack of tracking the path that was moved through, this was down to not implementing the Node class shown in the Class diagram. In the class diagram the Node objects represents each file opened and where it was opened from. By logging these it was then possible to track back from the destination Node to the start. A Node is added when it has been checked for two things, first that it has not been previously visited and secondly if its is not already in the queue to be visited. This stops the duplication of paths to Nodes being considered when following the route back. All that was being returned to begin with was the final route found, so the destination and the Node that opened it.

In development this was the first issue to be fixed, currently the algorithm finishes its run through and goes back through the graph, moving from a Node to where that Node was opened from. While doing this it passes the start and destination back to the algorithm which starts the search, however the search does preliminary checks to see if the route is already stored in the start Nodes file, meaning the two are already directly connected. This is guaranteed to be true in this case so the actual BFS is never hit. From there the Route information is passed on to the paint function.

3.4.5 Painting the Path

Painting the path was an area were little changed from the planned design, it is near identical to the imagined solution. To paint the path a Route object and the Map object are taken as parameters, a loop is completed through the points along a route, a line is painted between the two points by turning them into *LatLng* objects which are used when painting a poly line. To accommodate the grading feature the line is painted in its respective grading using the *parseColour* function provided. This continues until the last point where the function stops to prevent crashing. While simple no improvements could be thought of regarding the method, some visual improvements were considered including a fading of colour between segments however a solution could not be found.

3.5 UI Design

3.5.1 Map

Implementing the map feature was a fairly easy process, once the relative keys were set up and the correct package approved in the API console. One major decision throughout the process was the type of map that would be appropriate in each activity. Throughout the application the map differs based on what the user is doing, if they are trying to find a building the map gives an abstract representation as it is all that is really needed. If a user is route finding then a photographic representation is given. This can be changed in future and is another possible area for user testing.

3.5.2 Relative Design

Designing in such a way that an activity adapts to its screen was a task that was incredibly time consuming compared to what was expected. This is put down to the fact that the developer had little experience with XML and how to use it to be most effective. Despite this Android Studio provided a range of features which provided great assistance causing a quicker rate of development in the closing stages of the project.

One of the main problems was finding a menu layout which could be made to be adaptive to any screen, it was decided that a grid layout was a good option as no matter what we should be able to split a screen into the four possible functions. Early on research was done into the *GridLayout* however it required API level 14, it was felt the application could be developed to a lower point than this. Due to this a way had to be developed to have the grid work.

This was achieved through the use of fake views, these were view objects with no height or width on the screen, by including these it was possible to align buttons around them. From there setting buttons to fill their possible space was easy. Further problems arose with using white space around the buttons, something set out in the Design section. With issues in padding and margins not working, for still unidentified reasons it was necessary to resort to the graphical editor provided. With this the changes were made in minutes, while learning what can be done with XML it was also important to make best use of the tools available as the feature was closing in on taking too long for the benefit it would provide. A representation of fake views can be seen below.

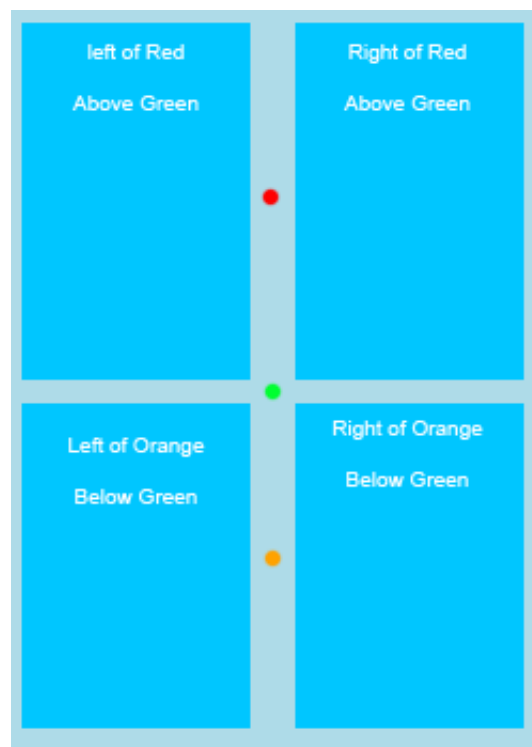


Figure 3.4: An example of using fake views for alignment, buttons are set to be a specific side of the view.

Custom buttons were one of the least complicated features within the application, the facilities provided by Android were far from perfect, with some researching it was noticed that the best solution would just be custom buttons. In this case the application makes use of an adapted on line resource [22], with the base code being taken from the site and edited to fulfil the developers needs.

From there the screens are fairly simple, buttons are set at the bottom of maps and maps fill the screen unless the device used is very large. With some testing already done on various screen sizes it is fair to say an adaptive design has been achieved. UI implementation was by far the most underestimated task throughout the project design, at times it set the project far behind schedule.

3.6 Conclusion

To conclude the current achievements will be referenced to the functional requirements (FR) set out in the requirements specification section 4.1.2 in this document. Moving through the FR's it is possible to see most of the requirements have been met, with some missing but other alternative implemented. In the list set out the first requirement is the Route finding function,

further detail provided specifies the use of colour coded routes. This has all been achieved while maintaining a high level of extendibility including menu portions and the graph itself, this can be considered a strong success. One area for improvement however would be the improvement of displaying information along a route.

FR2 was a location based help service, that gave the user a person who could possibly help them. While this has been met it is by far the functionality which has the most room for improvement. While implemented to what has been stated it is felt to still be fairly underwhelming compared to the feature it could be. An area where future work should be heavily considered.

FR 3 was the ability for a user to display buildings, filtering the type. Another are which can be considered a success, partly down to the simplicity of the functionality. It is further highlighted in the specification that the locations should be loaded from an easily update able file, another implementation. From there further modifications were made, including information on the buildings department requested from the previous feedback form and custom markers, small but adds some difference to the application.

FR 4 was to implement a route plotter for the user, including the grading of the route. Another feature fully implemented, not only can a user plot a route but they can visualize what it will look like in the route finder as well as log the steps on the route and have the distance logged for them. A file is also output for the user to do with as they wish, with the file being fully compatible with the route finder. Another functional requirement fully met, possibly exceeding initial expectations.

FR5 was a feature which was possibly lost along the way of development due to varying issues. Mainly the lack of support for Welsh within Android. Due to this the welsh locale would have to be set programatically, an area which seems to cause issues for any who have done it previously. Due to this a test feature has been implemented, this sets the locale to French but the strings file used actually uses a Welsh translation. While a hack like implementation, it is still functional and is a proof of concept if Welsh locale is added.

A functional requirement which should have been included is a good UI design, while referenced throughout it is never seen as key, which it is now seen as after development. In the applications early development, its use was hindered by poor layout, design and flow. While this has changed considerably the memory of how badly the design effected user experience made the effort in including what is reasoned to be solid design worth it.

Overall the project is considered a success, with all functionalities being mostly met, some exceeding expectations while one or two needing more work to be the fully finished product. For the time line given the developer is satisfied with the achievement.

Chapter 4

Testing

4.1 Overall Approach to Testing

Overall approach to testing was fairly simple, with the completion of each feature any relevant tests were either coded or completed by the developer. This included unit testing, stress testing and acceptance testing. Due to the applications aim of helping a user it was important the UI was received well.

4.1.1 Problems Encountered

Throughout testing many issues arose, many more than were found in development. Getting a working test environment functional for the first time was much more complicated than expected, with references to this around the web it is not an isolated incident. Problems included -

- Out of date documentation on the official Google Developer site.
- Bugs in the current (1.2) version of Android Studio which will show an empty test suite despite the tests being there, ran and results printed.
- Lack of official support for unit testing in Activity classes, while it seems possible the issues it proves are numerous. Alternatives were discovered near the end of the project time line but a lack of time meant other more important tasks were completed.

In future projects more time will have to be dedicated to the testing of the application. It is thought the issues encountered with testing were down to mix of issues, mainly the early stages of Android Studios official release and the tight schedule for testing including final acceptance testing.

4.2 Automated Testing

Automated testing was completed throughout the duration of the project, problems with unit testing were previously discussed however Android did have other important utilities to make use of which helped with thorough testing, especially looking for crash cases.

4.2.1 Unit Tests

Despite the issues encountered, unit testing was completed on the non Activity classes. This included the testing of correct behaviour from the classes to handle distance calculations and from the Objects which made up a Route Object. While the tests results would not appear within the IDE due to a known bug it was still possible to see the results in a created HTML document logging pass percentage. While it would have been beneficial to complete more Unit testing it would have required the re structuring of the program and reversal of many design decisions. While an attempt to follow a MVP pattern was made, as can be seen with the XML, Route Display Activity and Route object, in future it is advised it is stuck to more thoroughly to aid with testing.

Class `com.example.walkingtour.Data.DistanceTest`

[all](#) > [com.example.walkingtour.Data](#) > DistanceTest

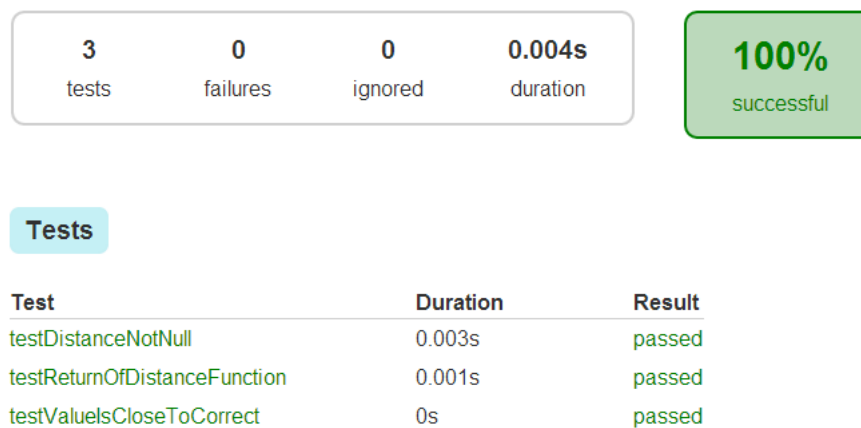


Figure 4.1: Image displaying the output from running a set of tests, these are results from a single small set of tests.

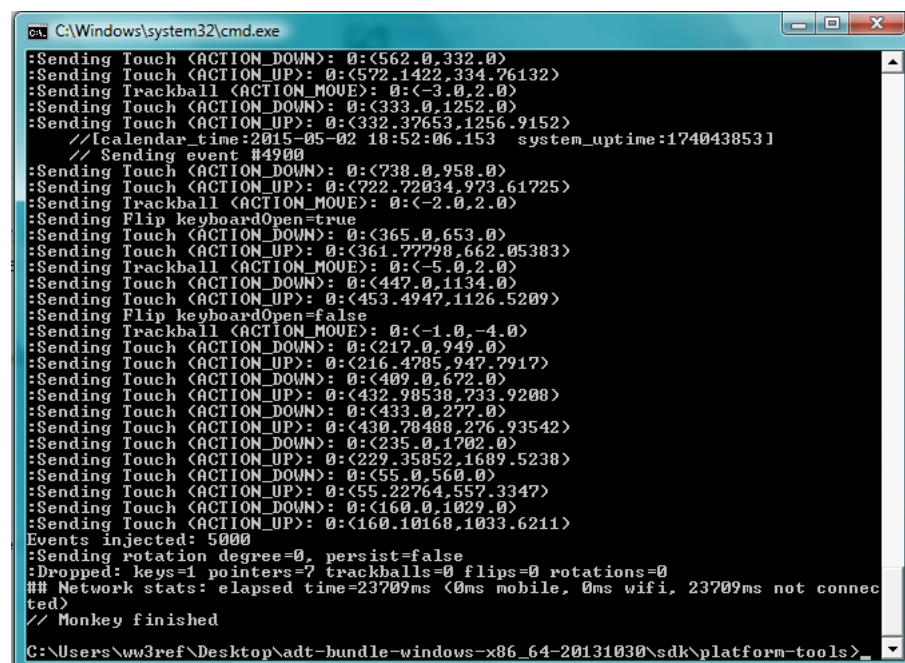
4.2.2 Stress Testing

Stress testing was completed with the use of 'The Monkey'. 'The Monkey' is a tool which sends a range of inputs to the application and logs the results of this at the end of its run. By doing

this the application is spammed with events and any area of the application where hang ups in performance can cause crashes should be found. Through final stress testing well over 200,000 inputs have been simulated and no serious faults can be found in the application. However this does not mean that faults are not there, just hard to find if they are.

Throughout stress testing one issue which has been prevalent however has been the lack of support for the cancelling of activities, this has been noted in both the route data entry screen and the route choice screen. Since then these have been handled along with the addition of further tests in the code to deal with a user selecting the same location as a start and destination.

Using 'The Monkey' tool was a real benefit for the developer, as pointed out while it found no significant errors the ones it did find would have caused a real effect on the usability of the application. In future it is advised any improvement to the application be tested in such a way, the tool can be limited to the use of a single package and as such can be fine tuned to send demands to the right class.



```

C:\Windows\system32\cmd.exe
:Sending Touch <ACTION_DOWN>: 0:<562.0,332.0>
:Sending Touch <ACTION_UP>: 0:<572.1422,334.76132>
:Sending Trackball <ACTION_MOVE>: 0:<-3.0,2.0>
:Sending Touch <ACTION_DOWN>: 0:<333.0,1252.0>
:Sending Touch <ACTION_UP>: 0:<332.37653,1256.9152>
//Calendar_time:2015-05-02 18:52:06.153 system_uptime:1740438531
// Sending event #4900
:Sending Touch <ACTION_DOWN>: 0:<738.0,958.0>
:Sending Touch <ACTION_UP>: 0:<722.72034,973.61725>
:Sending Trackball <ACTION_MOVE>: 0:<-2.0,2.0>
:Sending Flip keyboardOpen=true
:Sending Touch <ACTION_DOWN>: 0:<365.0,653.0>
:Sending Touch <ACTION_UP>: 0:<361.77798,662.05383>
:Sending Trackball <ACTION_MOVE>: 0:<-5.0,2.0>
:Sending Touch <ACTION_DOWN>: 0:<447.0,1134.0>
:Sending Touch <ACTION_UP>: 0:<453.4947,1126.5209>
:Sending Flip keyboardOpen=false
:Sending Trackball <ACTION_MOVE>: 0:<-1.0,-4.0>
:Sending Touch <ACTION_DOWN>: 0:<217.0,949.0>
:Sending Touch <ACTION_UP>: 0:<216.4785,947.7917>
:Sending Touch <ACTION_DOWN>: 0:<409.0,672.0>
:Sending Touch <ACTION_UP>: 0:<432.98538,733.9208>
:Sending Touch <ACTION_DOWN>: 0:<433.0,277.0>
:Sending Touch <ACTION_UP>: 0:<430.78488,276.93542>
:Sending Touch <ACTION_DOWN>: 0:<235.0,1702.0>
:Sending Touch <ACTION_UP>: 0:<229.35052,1689.5238>
:Sending Touch <ACTION_DOWN>: 0:<55.0,560.0>
:Sending Touch <ACTION_UP>: 0:<55.22764,557.3347>
:Sending Touch <ACTION_DOWN>: 0:<160.0,1029.0>
:Sending Touch <ACTION_UP>: 0:<160.10168,1033.6211>
Events injected: 5000
:Sending rotation degree=0, persist=false
:Dropped: keys=1 pointers=? trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=23709ms <0ms mobile, 0ms wifi, 23709ms not connected>
// Monkey finished
C:\Users\vw3ref\Desktop\adt-bundle-windows-x86_64-20131030\sdk\platform-tools>

```

Figure 4.2: Image displaying the results of a Monkey run using 5000 inputs resulting in no dropped actions and no crashes on the device side.

4.3 Integration Testing

While some integration testing was completed there is little to document due to the nature of the application. This is down to the fact that the functionality within the application are essentially singular branches accessed through the Menu Activity. As this is the structure, very little is

passed between multiple functionalities, the only place this can be seen is the file from the Route Plotter being compatible with the Route Finder graph structure. So while testing was completed, the initial unit testing combined with final acceptance testing was seen as sufficient.

4.4 User Testing

After initial development a small user group was used to determine some UI elements, this mainly related to the use of the application for first time users and the overall visual appeal of the application. For these tests users were asked to perform a variety of tasks relating to the functional requirements, the results of these can be seen in the Appendix section C. Users were also asked for their opinion on the colour scheme of the application, this was done through the use of different themes for different pages on the test application. While some of the features were a little bare at the time they were all fully functional, just not polished. Results from these tests were used to set the theme however little was discovered from the small user group. In future it is advised a much larger group is used.

4.5 Acceptance Testing

Acceptance Testing was performed by the developer of the application, a list of tests was drawn up to determine the full functionality of the application including cases which may cause crashes from poor quality code. The test tables which represent this information can be found in Appendix section C. Some were completed after full development, others during development.

4.6 Testing Evaluation

While the developer feels a sufficient amount of testing was completed for the time given, it is still lacking in terms of a full project. Ideally a whole restructuring of the program should be undertaken to help with unit testing and a better adherence to the MVP pattern which is ideal for this situation and Android in general.

Chapter 5

Evaluation

5.1 Requirements and Process

5.1.1 Requirements

The requirements, as stated before, were mostly correct. However it has been shown that one could have been a lot more clear, mainly down to a lack of research relating to that specific requirement. In future it would be wise to do more than skim the surface of what seems a simple requirement as it can be deceiving. It could also be argued that there was a lack of detail surrounding some requirements, while it was fleshed out in design it does not give the designer much to build around. However there were requirements that had the sufficient detail, its just making sure that in future the developer makes all requirements equal in depth. While an area which may need improvement, requirements can be seen as a strong area in the report, with all key functions included.

5.1.2 Process

Working with the process has definitely been seen as a success in this project, with the process being adapted from other more concrete versions it was unsure if it would provide the developer with a strong enough base to develop from. However with design being completed twice it was more than sufficient, it also meant that once one feature was completed it gave a framework for further features. This was due to some problems already being encountered through other development cycles, a problem that occurs a second time is easy to fix. In hindsight the process may have been a little vague for a full team to follow, with a single developer it is easy to visualize something you've planned out in your head. The description included within the document could be more detailed, an area to focus on when working in a team.

5.2 Tools and Design

5.2.1 Tools

With this project tool use was very inflexible, as seen in the document it was necessary to use a specific IDE to gather what were seen as needed skills if this was a field the developer wanted to enter in the future. GitHub was used for version control, again a standard tool that is likely to be used in almost any software project. One improvement which could have been made is very early planning on tools, by not doing this the developer took time away from learning Android Studio because they were not aware that it had become the preferred development environment compared to the now deprecated Eclipse with ADT. A solid effort was made to use the best tools, however it was not a complex task.

5.2.2 Design

Design was another area which can be seen as a success in a way, but with definite room for improvement. The outline of the program is well set out, the initial class diagram looks like it could have been implemented to a working degree. While this was built upon in further iterations it worked as a general idea. With it being developed, the final design being in this document, it turned into a detailed description, including justification of what was built. While some of the application differs it was due to either further improvements on design or a change in plan. With design also including sequence diagrams, overall structure and flow diagrams a very strong outline of the project was built. In depth design was achieved through the pseudo code of important algorithms, some of which were literally translated to code line for line and worked with little to no issues.

An area where improvement could have been made was a better highlight of exactly *how* the program would be extendible. While many references are made to the idea little detail is provided, causing effort to be put into design in the implementation phase which could have been done previously.

5.3 Implementation and Aims

5.3.1 Implementation

Implementation was by far the strongest stage of development, it is thought this was down to solid requirements and design specifications. All the necessary features were set out with possible design solutions also there to rely on. Throughout it was also required to make the application easily expandable, an area that is felt to be the strongest of all features developed. While not a feature in itself, it should make the most different to future maintainers. Route Searching is also seen as a success, with grading and route plotting being relating features. With this current project the improvements on route finding has been huge compared to the original Access

Aber [1], which is what has been referenced for areas for improvement.

However improvements can still be made, even with the areas considered strongest. A single graph for route finding would be a benefit, as well as better help function though more communication with the University would have to be undergone for some of the possible ideas previously outlined. Route plotting is another area with improvements in mind, to be outlined in the future work sections.

5.3.2 Compared to Aims

When comparing the finished project to the initial aims, it is sufficient to say they have been mostly fully met, with improvements in some. While some are not implemented ideally, it was not expected in a project with a fairly short time span. With reference to the Requirements Specification, as has been mentioned previously, four out of the five outline requirements can be considered fully met with additions, with one being an outlier due to extenuating circumstances.

5.4 Future Work

Future work is an area that could branch out into any number of directions, purely down to who is developing the project at the time. Currently the developer would like more time be put into the route finding implementation, with more support for disabled users and a more technically challenging solution to the graph situation. A set of possible future work, all of which interest the current developer are

- Implement a single graph and a more complex search algorithm.
- Allow users to input variables about themselves and find routes based on that.
- Separate the route plotter from the current application.
- Add a touch to log point feature, a user then has the choice to walk a route or not.
- Further filters on buildings, department and theatres possible solutions.
- Improve the Help function significantly, possibly implement a system which contacts helpers on open days.

All of these are areas which will benefit the current application. While not all of them may get much use, they at least provide experience for the implementer and as such benefit someone.

Providing features for disabled users is something the developer feels strongly about, a separate list has been set out to demonstrate possible ideas.

- Multiple colour schemes for visually impaired users.

- Touch to talk text, for those who again have visual impairments.
- Read out instructions during movement on a route, for example 'turn left'. Sat nav like functionality.
- Log what a user says they are capable of and have them never have to specify again.
- List buildings based on accessibility, find entry to buildings on routes based on accessibility.

It is felt the application can be developed to be something truly useful for the University, if development were to continue the benefits it could provide to a range of users would make the development time worth it.

5.5 Self Evaluation

Overall as a developer i am happy with the work i have completed and the standard it has been completed to. Working on a project this large is not something i have done before and as such it was an experience i was nervous about. One area i am particularly proud of is the adherence to the requirements set out, as well as meeting most of the smaller requirements mentioned in meetings. While progress has not always been smooth the amount of technical knowledge i have picked up along the way has made the development time worth it. While developing the speed at which features were being completed was much faster than anticipated at points, if a larger amount of research had gone into one or two of the features before hand they also would have had their development time cut down significantly. Once i was a good way into the project my familiarity and memory of the architecture of the application was a huge bonus, the only reason for having the in depth knowledge of how the application should work was due to the thorough documentation done before development.

However there are areas i have noticed are lacking, one of these related to the reading of documentation. There were several times during development that the skimming of existing documentation for Android caused delays, one time for over two days. An area that definitely needs improving on. It has also been noted that as a developer i tend to be side tracked by features i did not set out to work on, doing this creates a sort of non organised progression and leads to design notes being ignored and an overall lack of cohesion until all of the changes that weren't meant to happen have been documented. However it is another area i should be able to easily change. A final area i feel i can improve on is the use of the tools available to me, while in this project it was a decision made to try and not rely on libraries at the end of the project i feel that may have been wrong. By refusing free help i think its possible i have detrimentally affected the project for my own gain when really i could have learnt what i wanted at a later date.

In conclusion i am happy with what has been achieved, having found areas i can improve on is another bonus as it will help me become a better developer. I feel the major gain however is that i have now identified an area of development i really feel passionate about. While before

the project i was a fan of mobile development and its potential power i am now planning future projects for the platform. Working for so long and barely scratching the surface of what's possible just through the Android API has made me want to keep developing and possibly seek a career in Android development.

Appendices

Appendix A

Third-Party Code and Libraries

A small amount of third part code was used throughout the project.

Greater circle function. [28]

```
private double distance(double lat1, double lat2, double lon1, double lon2,
    double el1, double el2) {

    final int R = 6371; // Radius of the earth

    Double latDistance = deg2rad(lat2 - lat1);
    Double lonDistance = deg2rad(lon2 - lon1);
    Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
        + Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2))
        * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // convert to meters

    double height = el1 - el2;
    distance = Math.pow(distance, 2) + Math.pow(height, 2);
    return Math.sqrt(distance);
}

private double deg2rad(double deg) {
    return (deg * Math.PI / 180.0);
}
```


Custom Button development. [22]

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true" >
        <shape android:shape="rectangle" >
            <corners android:radius="3dip" />
            <stroke android:width="1dip" android:color="#83776b" />
            <gradient android:angle="-90" android:startColor="#5f4427"
                android:endColor="#cbb59d" />
        </shape>
    </item>
    <item android:state_focused="true">
        <shape android:shape="rectangle" >
            <corners android:radius="3dip" />
            <stroke android:width="1dip" android:color="#83776b" />
            <solid android:color="#92806c"/>
        </shape>
    </item>
    <item >
        <shape android:shape="rectangle" >
            <corners android:radius="3dip" />
            <stroke android:width="1dip" android:color="#83776b" />
            <gradient android:angle="-90" android:startColor="#cbb59d"
                android:endColor="#92806c" />
        </shape>
    </item>
</selector>$

$<style name="btnStyleSandrift" parent="@android:style/Widget.Button">
    <item name="android:textSize">15sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:textColor">#FFFFFF</item>
    <item name="android:gravity">center</item>
    <item name="android:shadowColor">#000000</item>
    <item name="android:shadowDx">1</item>
    <item name="android:shadowDy">1</item>
    <item name="android:shadowRadius">0.6</item>
    <item name="android:background">@drawable/custom_btn_sandrifft</item>
    <item name="android:padding">10dip</item>
</style>$
```

Appendix B

Code samples

2.1 Example Test Code

```
@Test
public void testValueIsCloseToCorrect(){
    double ret = distance.distanceTo(52.413121,
        -4.086836,52.413372, -4.069066);
    assertTrue(ret < 1.5 && ret > 1);
}
```

2.2 Route Plotting

```
public void plotPath(ArrayList<Node> nodes) {
    Log.i("hit plot path", "sfd");
    int counter = 0;
    String search = end;
    Collections.reverse(nodes);
    for (int i = 0; i < nodes.size(); i++) {
        for (int j = 0; j < nodes.size(); j++) {
            if (search.equalsIgnoreCase(nodes.get(j).getName())) {
                plotRoute(nodes.get(j).getName(), nodes.get(j).getFrom());
                search = nodes.get(j).getFrom();
            }
        }
    }
    plotRoute(start, search);
}
```

2.3 Route Finder

```
public ArrayList<Node> findPath(ArrayList<String> endings) {
    boolean missingroute = true;

    Log.i("going from", start);
    Log.i("going to", end);

    ArrayList<String> visited = new ArrayList<String>();
    ArrayList<Route> processed = new ArrayList<Route>();
    ArrayList<Node> fileInfo = new ArrayList<Node>();
    ArrayList<String> nextfiles = new ArrayList<String>();

    visited.add(start);

    while (missingroute) {

        ArrayList<Route> possRoutes = new ArrayList<Route>();

        for (int i = 0; i < endings.size(); i++) {

            if (visited.contains(endings.get(i))) {

            } else {
                visited.add(endings.get(i));
                //Node n = new Node(last, endings.get(i));
                Log.i("Added to visited : ", endings.get(i));
                Log.i("test ", Integer.toString(endings.size()));

                possRoutes = reading(endings.get(i) + ".txt");
                // visited.add(endings.get(i));
                for (int j = 0; j < possRoutes.size(); j++) {
                    Route curr = possRoutes.get(j);

                    if (processed.contains(curr)) {

                    } else {
                        processed.add(curr);
                    }
                }
            }
        }
    }
}
```

```
    }

    if (!nextfiles.contains(curr.getTo()) &&
        !visited.contains(curr.getTo())) {
        Node b = new Node(curr.getFrom(), curr.getTo());
        fileInfo.add(b); // Logging which
        file is opened from where, track back.
        nextfiles.add(curr.getTo());
        Log.i("added to next files: ", curr.getTo());
    }

    Log.i("getting pst logic", "");

    if (end.equalsIgnoreCase(curr.getTo())) {
        Log.i("found route", curr.getFrom()
            + " " + curr.getTo());

        missingroute = false;

        return fileInfo;
    }
}

}

}

    }
    endings = nextfiles;
}

return fileInfo;
}
```

Appendix C

Test Results

3.1 Facility Display Test Table

Test	Result	Error	Change	Pass
Location Displayed	Pass	-	-	-
Functionality maintained on rotation	Pass	-	-	-
View Button responsive	Pass	-	-	-
Pop Up Inflates	Pass	-	-	-
Markers Shown By Type	Pass	-	-	-
Name and Information Display	Pass	-	-	-
Custom Markers Shown	Pass	Always uses same marker	Modify display code	Pass
Old Markers removed on click	Pass	-	-	-
Map Centers on Aberystwyth	Pass	Too zoomed.	Change camera settings	Pass
Cancelling Activity returns to menu	Pass	-	-	-

Table C.1: Table displaying acceptance tests ran on the facility display functionality and their results.

3.2 Help Display Test Table

Test	Result	Error	Change	Pass
Page Renders Text	Pass	-	-	-
Displays Closest Person	Pass	-	-	-
Renders Theme	Fail	Old XML file being used	Add new XML	Pass
Changes Person Chose on Location	Pass	-	-	-

Table C.2: Table displaying acceptance tests ran on the Help display functionality and their results.

3.3 Route Plotter Test Table

Test	Result	Error	Change	Pass
Taken to Data Entry Screen	Pass	-	-	-
Can enter text for start	Pass	-	-	-
List shown for locations	Pass	-	-	-
List populated from file	Pass	-	-	-
Cant move on without information	Fail	No checks	Implemented Checks	Pass
Taken to Map screen	Pass	-	-	-
Step counter shown	Pass	-	-	-
Limit on Step Counter	Pass	-	-	-
Location Shown	Pass	-	-	-
Dialog Shown on cancel	Pass	-	-	-
Cancelling route wipes data	Pass	-	-	-
Options Menu Inflates	Pass	-	-	-
Log Point creates point	Pass	-	-	-
Points Linked by PolyLine	Pass	-	-	-
Create Route creates directory	Pass	-	-	-
File created within directory	Pass	-	-	-
File format correct	Fail	Missing new lines	add in new line character	Pass
Multiple Files work	Fail	Directory overwritten	Perform checks for existing	Pass
File compatible with graph	Pass	-	-	-
Cumulative distance logged	Pass	-	-	-
Grading calculated	Pass	-	-	-

Table C.3: Table displaying acceptance tests ran on the Route Plotter functionality and their results.

3.4 Route Finder Test Table

Test	Result	Error	Change	Pass
Taken to Route Type selection	Pass	-	-	-
Map Loaded Correctly	Pass	-	-	-
User position shown	Pass	-	-	-
Responsive Route Button	Pass	-	-	-
Route Selection Screen loaded	Pass	-	-	-
Lists populated from files	Pass	-	-	-
No Step List separate	Fail	Loads from Standard Graph file	Implement new file	Pass
Lists expand on touch	Pass	-	-	-
Lists record use selection	Pass	-	-	-
Data passed back to route finder	Pass	-	-	-
Search uses quick solution on connecting starts and destinations	Pass	-	-	-
Deep search started on non connected nodes	Pass	-	-	-
User is stopped from having the start and destination the same	Fail	No checks done	Implemented checking	Pass
User is displayed route on activity resume	Pass	-	-	-
User is shown grading of segments in the route	Fail	Grading shown for the opposite direction route.	Swapped round variable entry.	Pass
Route makes use of other locations to plot through.	Pass	-	-	-
Route removed on request for new route	Pass	-	-	-

Table C.4: Table displaying acceptance tests ran on the Route Finder functionality and their results.

3.5 User Testing

A small set of test were ran on a small number of participants, this table is a simple representation of the results. It is clear that most functionality works well with some issues maybe revolving around the multiple screens to find a route. It was discovered that for an average user they do not know where to find a route, no feedback is given on result something which needs addressing.

Task	Pass %	Success
Navigate to Help screen	100%	Yes
Navigate to Building Display Screen	100%	Yes
Navigate to Building Display	100%	Yes
Display Only Accomodation	80%	Yes
Display a Route from IBERS to IMAPS	80%	Yes
Plot a small two point route	70%	Mostly
Print your plotted route	100%	Yes
Cancel your plotted route	100%	Yes
Can you find the printed file	0%	Fail
Find a No step route from National Library to Edward Llywd	90%	Yes

Table C.5: User Testing results displaying percentage of users which passed a task.

Appendix D

Requirements Specification

4.1 Functionality

The major aim for this project is to provide an Android alternative to the already existing Access Aber web application [1]. Currently the project exists solely as a web application which can be run on both iOS and Android through the use of a web browser. However the project is limited due to the time it was developed in and it is my aim to show that developing the application native to one environment can provide benefits which make the time and effort worth it. The idea is to not only port over the existing application but to improve upon it in ways that are seen fit. I feel that with the power given to a programmer within not just Android, but mobile devices in general, the application can be made to be not only effective but intuitive and an asset to those who visit campus. The application will consist of several areas that will benefit the user and will heavily rely on the Google Maps Android API [14].

4.1.1 Route Finding

One of the main features to be included is that of a route finding system around campus. This differs from the web application due to the system actually finding a route, as it stands the application has around 200 stored routes which a user can select from. However these are displayed as a list and require the user to spend a large time searching through with no guarantee of there route actually being stored.

The Android application will be developed in a different way. The user must be able to find there way around campus without having to search through 200 routes. The plan to combat this is to first have a much simpler way of route selection, ideally this will be two drop down boxes where a user selects where they are and where they want to go. This means that if we cover 20 locations, we just need 20 options instead of representing it as over 400 hard coded routes which is what would be needed in the current set up.

The second plan to combat the current issue is to remove most hard coded routes. All that

ideally has to be implemented is for each building to connect to at least one other. If this is implemented correctly we can instead search for a Route rather than store them. This saves the user time and means that the application should be easily extendible.

One issue which arose in a meeting with the section of the University which initially set up the project was that while routes were shown not much else was. No information regarding the route and no indication of the difficulty of it, both of these would be of a lot of use especially to disabled users of the application. It is already known that colour coded routes should be possible within the application, combining this with a small window which shows information such as distance, elevation and steps on the route should fix this problem which has been highlighted.

4.1.2 Route Plotting

Another issue that arose within the meetings was that the plotting of a route was done manually, the only visualization the user inputting it had is whatever setting was provided through Google Maps. It was suggested that a suitable addition would be a route plotter which let the user walk a route and logged it for them.

Due to the planned nature of the system all that will hopefully be required is for the user to plot one route that connects to the graph of locations. By following this process we can fulfil another requirement that was brought up within discussions, the adding of information about a route. It may be possible to build a step free graph, thus allowing guaranteed step free routes. Some buildings are inaccessible so they would have to be omitted from this feature.

4.1.3 Location Based Help

A further requirement is the fulfilment of a help service for those on campus. Aberystwyth's campus can be a hard place to traverse along with confusing if it is your first time visiting. Due to this it is very possible that a user will require help to either exit a building or enter another, while help will be provided through another feature mapping the indoors of buildings will not be included in the application due to time constraints. To combat this a help service will need to be included, this will display to the user the closest person who may be able to provide some assistance to them.

Not only will this help the user if its needed, the presence of it should assure them even if it is not required. It has been noted previously that even some members of staff with mobility issues do not enter some buildings due to them being poorly labelled and other factors. This may be somewhat of a help to a complex situation.

4.1.4 Building Display

This is a feature that will be brought straight over from the existing application with only a few changes. It will provide a list of categories and let the user choose which they would like to view

and then provide a display of where they are.

The feedback from the initial application is something which will be of great use here. There were several possible additions including pictures instead of just markers, something which is very possible and the inclusion of building names above names of lecture theatres as it can be unclear exactly where something is due to departments sharing buildings.

4.1.5 Multi Lingual Support

Due to this being a Welsh University it is required that welsh language support is provided for the likely event that a native welsh speaker uses the application.

This is an area which can pose problems. Android appears to have little to no support for the inclusion of Welsh language however new locales can be created and used which should help solve the problem.

4.2 Interfaces

4.2.1 User Interface

4.2.1.1 General User

The General User must have access to all features except the route plotter. The route plotter gives them no added actual use for their specific needs, however it is currently undecided if the project will be split up into two separate projects or just built as one and adapted from there. Having the Route plotter within the main application may confuse the user to a degree as unless you are aware of what its doing it will not appear to provide much of a benefit. It is likely it will be left in unless further use can be seen in splitting it up from the main part of the development.

Future development could focus on providing different User Interface based on the user type. This could be focused down further to provide different functionality based on a users mobility level. However in the time scale provided it is unlikely that this feature will be done now, if anything some level of customisation may be provided relating to route finding but not much else.

4.2.1.2 Staff Users

Staff users need access to all features, for addition to the system and to check everything is up to date. This is not a problem.

4.2.1.3 Encountered User Criticisms

Using the already completed user feedback results for the web application we can already start to set out some rough requirements for the look of the application and what needs to be shown. One major criticism was the fact that users are dropped straight into the map, this means that from there you have to work out what does what within the application. This also links to the problem of the buttons used currently being small and non intuitive which causes issues in itself.

This means its become a requirement to fix these base problems, most other problems highlighted link back to the confusing nature of the current User Interface. A huge improvement needs to be made to this, while some elements like the colour scheme are suitable for the purpose overall it is clear why users have issue with it. A possible solution is having a menu of sorts that is clearly labelled which then launches separate features meaning a simpler UI on each screen and no cluttering or extensive menus.

Other criticisms which effected user experience were related somewhat to the labelling of information within the site. It was not always clear if a labelled room was correct due to it appearing around other rooms for other departments. This is due to departments sharing buildings and is a fairly simple fix as all that needs to be done is to label each room with its respective information.

4.2.2 Google Services

The application will have to interface with several of Google's services. This mainly relates to interfacing with the map service which requires an API key, in this case API keys are free and will give us more than enough requests a day to develop the application. Using the Google Map service gives us a large range of benefits due to their thorough API [14], this allows for the development of elements like visualising a route on screen, showing a user there progress and markers for displaying building locations.

Having this thorough API is key to the success of the project, other possibilities have been explored including Open Street Maps but due to the developers past experience and what are seen to be gaps within the OSM API the Google Service has been chosen. While the service lacks some of the detail that may be seen as useful the benefits it gives more than make up for it and means that we can cover any gaps with our own implementations. The ability to customise elements of the maps is also a welcome addition and should help with the development of the labelling of buildings and displaying route paths and their grading.

4.2.3 Hardware

The application will also have to interface with hardware elements within the mobile devices. This mainly means the GPS module and possibly sensors to detect the gradient of a slope. Interfacing with the GPS module is made easy through the use of the Location Manager available through Android, this provides us with a periodic update of the users position allowing us to plot

routes, display their position to them and to make estimations on who is best to contact for help.

It is also possible that the Camera will be used to help with plotting routes however it is more of a suggestion than a requirement. If this is implemented it means a picture would be displayed of both the start of the route and the end destination. Doing this will mean that the user has a visual aid to help with finding their building as there could be some error with very close together destinations.

4.2.4 Interfacing with itself.

The application will also have to interface with itself in a way, the files created by the route plotter need to be ready to just add into the projects resources and work straight away. While this is less of a straight one to one interface the elements of the application need to be compatibly with each other, maybe an obvious requirement but necessary.

4.3 Performance

4.3.1 Offline Performance

There are several areas within the application where performance can be effected by what is to be completed. One major issue with this is that the application simply cannot work with no Internet Connection, at least not the first time round. The application has to load in the Google Map object and to do that and have the relevant imagery a connection is required. In this case the only real requirement is to at least notify a user of this and notify them if their is no current connection. A blank screen is shown by default when there is a lack of connection which needs to be remedied. It is suggested that a tech spike into caching imagery of the surrounding area is completed to make the application fully functional even when offline.

4.3.2 Searching Algorithm

A further area which could provide small problems, especially on older phones, is the searching for a route. However it is unlikely with the expected size of the graph that the algorithm created will cause too much effect. It is hard to set a requirement regarding the search, we need to it perform well but without an existing set up it is hard to test what we feel is a good range. However for the current stage of the project it is sufficient to say we need to test on a range of devices to find a good middle ground.

4.3.3 Possible Problem areas.

In addition to the performance issues previously we also need to ensure the performance of the GPS module, it is very possible that errors are made using cached values for the latitude and

longitude but this should be a fairly easy fix. By guaranteeing new readings we improve our accuracy and provide the user with the route they actually plotted.

Performance wise we also have to take into account the responsiveness of the application. It is a common problem to encounter freezing and unclear interfaces within applications. To combat this steps need to be taken to not only improve code quality but provide a view that keeps the user updated on what is happening. Its not too much of a problem if it takes two seconds for a route to be found if the user is at least told about what function is being carried out.

One last problem area involves the performance on a range of screen sizes, the UI has to be coded in such a way that it does not depend on fixed sizes and instead relative ones. By doing this we ensure a consistent and appealing design despite device.

4.4 Attributes

4.4.1 Maintainability

A big issue with the application will be the maintainability of it; there is no guarantee the initial developer will always be around to change or fix it and as such this provides some restrictions on development. It means a high level of code quality needs to be maintained, while this is ideal in all projects it is maybe more so in this project as the chances of it being passed on could be quite high. This means full documentation, concise and clear comments and possibly JavaDoc but this will be decided at a later date.

Maintaining up to date information regarding locations of buildings and facilities is also something that needs to be considered. The issue here is whether we want to trust that a user can access the internet and gather the needed information from there or if it should come on device. Keeping the information on device means that an update is needed to change any of the information. Maybe not an ideal solution but possibly the best.

4.4.2 Expandability

This is a section with a large amount of requirements, the solutions to which will be fully considered in the design documentation due to the in depth analysis needed. The first problem is that its very possible the application will need expanding in future, because of this we need to make it as simple as possible to expand it for users that might not have large technical experience.

Due to this the code will have to read in information, and analyse it, using no built in constants. It needs to be developed in such a way that the files that contain the information guide the software as such. By doing this we keep a solid sensible structure to the information which will be easy to pull out with file reading algorithms in the code itself. The problem with this is that an error in a file could cause huge problems, however error checking in the file reader should help us solve that problem in somewhat of a sensible way.

4.4.2.1 Expanding the Graph

The big issue with expandability is the ability to add new locations to the route finder and as such expand the graph. It is possible that users with low technical expertise could be the ones responsible for expanding the project in future and as such the initial development has to make it easy for changes in the future.

To do this it is suggested that a simple set of instructions is clearly written up which describes in detail the process of adding in a new location. Ideally this will just mean creating a new file that contains the information of a locations name and a route into the graph of nodes. By implementing it in this way the maintainer does not have to understand the workings of the search or the traversal of a graph, just how to write out a basic file. A location file should ideally contain the start name, then details about its connecting nodes and the latitude longitude points which make up the link between the nodes.

It is essential that no code has to be changed to include these additions, relying on fixed variables in the code will cause large issues further down the time line of the project, beyond the initial building.

4.4.2.2 Expanding Choices

A further issue which arises with expandability is with the adding of new locations and how we can make it so they are recognised and added to existing menus in the application. This mainly effects the selecting of the start and destination in route finding, the current plan is to load all possible places from a single file so adding to the menu would simply mean attaching a location to the end of a word file, something most people are capable of.

By having adaptive menu choices we keep further developers away from old code that may not be as intuitive as it might need to be. Having the application set up so that it can be expanded without the editing of existing code will be a real bonus as was described in the initial meetings.

4.4.3 Security

Security issues are something we do not really need to consider given the nature of the application. The user does not provide us with any personal information and there is no sensitive information included within the application. All data used will be publicly available and the code open sourced.

One way in which this might change in the future is the inclusion of directly uploading routes to the application and having devices load them from there. If this is developed in the future it is advised that great care is taken due to the damage that can be done through the inclusion of fake locations and misleading information.

4.4.4 Design

Design is an attribute that will need further research, this is due to Google releasing design guidelines [15] to improve the quality of Android applications due to an increasing opinion of iOS applications looking much cleaner and professional. However one consideration is access for those with vision impairment.

During the project time line and definitely in any further development a consideration to take is assistance to those with impairment. This can come in the way of multiple colour themes to help those with types of colour blindness as well as a possible sort of Sat Nav feature.

This feature would read out directions for the user, not only benefiting those with vision impairment but any general user. Maps can sometimes be a little unclear so if we provide instructions for when a user is at a specific GPS co ordinate then we know we are giving them enough information to properly follow the route provided.

4.4.4.1 Route Design

The design of the route displayed is also an important element with a range of requirements. First of all as previously described the user needs to be able to see the difficulty of the route. With the plan being displaying large routes made up of much smaller ones we will be able to display the difficulty of smaller sections than the whole path. This means users can see where the difficulty may lie and possibly find a way around it. With campus being fairly condensed with winding paths and roads its sometimes difficult to tell what is coming up. For disabled users to know that a path further on has a high incline could be valuable information.

To build on route design it is also required that the display is intuitive. This should be helped by the facilities provided by the Google Maps API [14]. Being able to show a users location and orientation will be very useful especially if a custom arrow graphic is used rather than the standard dot. It is also important that the start and end points be clearly labelled to help with orientation.

4.4.5 Overall User Interface

The application must follow a consistent and appealing graphical interface throughout. This will help build up the users familiarity and help improve the intuitiveness of the design. This includes several guidelines which will be laid out in more detail in the design document but include -

- Options are selected from Expandable List View
- Pop up menus used for small actions on map screens.
- Custom Button XML to provide a theme appropriate display.

4.5 Design Constraints

The design constraints for the project are fairly obvious. The project must be Android compatible and will be developed in the Android Studio IDE. It will be tested using both unit tests and black box testing with users who have no experience with the application.

It must be compatible with the lowest Android API [17] level possible however what that currently is will not be clear until further on into development. The current aim is no higher than API level 14 which should be possible with the current estimated feature list.

4.6 Final Comment

The project will be developed as closely to this requirement specification as possible but may deviate due to the nature of software development, while this document details what is thought to be the current best practice it is possible that the final project deviates slightly from it.

Appendix E

Design Specification

CONTENTS

5.1 Introduction

5.1.1 Purpose of Document

The purpose of this document is to set out an initial design for the project 'Adapting Access Aber to android making use of native features'. It will encompass various design decisions and estimations of what the final system should look like on a code level. The final Design may differ but should lie around the boundaries set out here.

5.1.2 Scope

This Design Specification will split up the application into several components and describe any interaction within the application. Some components may essentially be stand alone except for the link to them however. This document may reference the requirements specification previously set out, it will likely reference the required functionality set out within the 'Functionality' section. For ease of use the following are the main areas of functionality required, these will be referenced by their key displayed here throughout the document. More details on each functional requirement can be found in the requirements specification.

- FR1 - Route Finding - Guiding a User around campus
- FR2 - Route Plotting - Users must be able to plot their own routes.
- FR3 - Location Based Help - Provide details of possible help available.
- FR4 - Building Display - A display of related buildings
- FR5 - Multi Lingual Support - Welsh translation

5.1.3 Objectives

The objectives of this document are to -

1. Describe the main components of the Access Aber Application (AA) [1]
2. Provide Details of interactions performed by the Application
3. To provide details about likely class interfaces
4. To provide diagrams of significant data structures and details of significant algorithms

5.2 Decomposition Description

The decomposition description will provide details on the division of the modules which make up the application. It describes the structure of the system and the function of each significant module. This should give a good overview of the expected design in an abstract way.

5.2.1 Modules within the Application

This section will describe how we expect the application to be split up for development. Due to the singular nature of most of the application, it is likely development will take place by feature rather than as a whole.

The application itself is expected to be split into 4 main modules which will be accessed through a central menu.

1. A route finder for users to enter a start and destination and have a route displayed to them.
2. A help service which provides details on the closest possible person who may be able to provide help.
3. A display of buildings that the user can filter.
4. A Route plotter to help the maintainers plot routes while seeing them and the current route they're plotting.
5. It is also possible a separate application will be developed to test the loading of route files created by the application.

The following section will detail further design elements related to the expected modules.

5.2.1.1 Route Finder

The route finder is where the main source of issues is expected to come from, the representation of data and searching of that data is expected to pose problems and take a fair amount of time to perfect. However some steps have been taken already to research possible ways to solve this task. A searchable graph is required to avoid the issue seen in the initial web application of having too many saved routes.

It is expected that a Breadth First Search (BFS) [20] will be used, this is due to the guarantees it can provide us and past experience. Using BFS should mean we will provide the shortest path in terms of locations traversed, while this may not guarantee the definitive shortest path the layout of campus and the graph that is expected to be used means the least locations should always mean the least distance. While BFS can sometimes be heavy on memory use due to it storing every node on a specific 'level' it is not expected that the graph used will be overly large

so issues relating to this should be mostly avoided. This also helps us combat the time issue. In a larger environment we would have significant problems and if this application is adapted in future this search may need to be swapped out for another.

Each Node will essentially be a traversable location, whether the location is a destination or not is another matter. With initial research it is obvious that the graph will have to abide by the road and walkway system within the Universities campus, so some locations may be junctions in walkways. Due to this each link between each Node should be represented as a set of Co Ordinates which we can then plot on the map provide through Google Play Services.

It has already been decided that in an ideal solution no location will be hard coded and instead loaded in from files who follow a set format and as such can be made with the Route Plotting application to meet FR2. The source of these files is expected to be within the applications own assets, however research will be done on fetching the files from a server to see if it provides any significant benefits. Files retrieved from the internet would mean we did not need to push an update every time something changed, however it is unlikely campus will change significantly and updates should not be too common. Thus meaning local files is probably an acceptable solution especially considering the issues posed by poor signal on campus for mobile devices.

Routes will also have a grading with them, something requested in meetings with the initial 'customers'. A Green, Amber, Red solution should be implemented letting users know the difficulty of the route they are about to follow. Due to the plan to have smaller routes make up larger ones it will be possible to highlight small segments of the route which may provide difficulty. A set of overall information should also be provided, with a user being able to see how many steps they have to take, distance and possibly elevation figured using the Google elevation API [11].

A further possible feature is the inclusion of a step free graph, this will be exactly like the current graph but a version that includes no steps on the way. By doing this we make the application more accessible. It should be fairly routine to expand to this idea, all that should be required is a new set of files for locations and their links.

5.2.1.2 Route Plotting

Route plotting should be a fairly simple implementation but still fill the needs of the User. Route plotting will take place using a Google Map object and the Location Manager provided within the Android API [17]. A user will be able to click a button and the onClick() Event will call a refresh on the users position. By doing this we should be able to provide accurate co ordinates. Each time a user has clicked a Poly Line object will also be added to the map detailing the current route shown. By doing this the user has a visual representation of what is happening. Poly Lines will also be used for showing routes on the route finder. It should also be possible to add information relating to inclines and steps if possible within the time scale of the project.

The Route Plotter will provide functionality to cancel the current route or write it to a ap-

plication compatible file. The file will fit the format expected from the reader, by doing this we guarantee a working way to create new files, it stops errors occurring on the human end of expanding the application.

It is possible to remove this module and make it its own application but the benefits of this seem insignificant. Due to this it is most likely this module will be accessible through the main menu, clearly labelled. This way we hope to avoid confusion, no negative actions can be instantiated through the route plotter due to the files saving to the device. While possible to upload to a site, the added work created by doing this for a small part of the application makes it difficult for the development time to be worth it.

Route plotting should also include the generating of a grading for the route, again taking into account total distance, steps included and elevation. By doing this we create a general idea for the user about what is on the route they've been given. Routes with no steps should be at a maximum graded amber for ease of use, the defining factor between a route being green and amber will be based on total distance and elevation difference in comparison to the distance.

5.2.1.3 Location Based Help

The application should also provide the user with information about receiving help. However it is possible to take this one step further and suggest help based on location. By doing this we minimize the possibility of someone requesting help from a person who cannot provide it from where they are. It should also make users feel more comfortable by knowing that help is available, this was something else that was brought up in the relevant meetings.

This will be completing by using the Great Circle function [30] and a set of points that represent people who can provide help. Each area of campus tends to have a building and a relevant person who is dedicated to the accessibility for that building. To begin with these people and buildings will serve as the points that we will calculate the distance between. The great circle function provides the shortest distance between two points on a sphere, in this case the Earth, it provides us with accurate results even over long distances, it can be mildly inaccurate over short distances but there are ways to fix this. The function should take the users current position probably shown as a Location object, and an array of other locations, it should return a sorted array so we can just get the first object out and know its the place that help is most likely to come from.

5.2.1.4 Building Display

As mentioned in previous documentation this is a feature that can be ported over from the existing application without much of a change. As it stands there are some problems but they have been found with user feedback and can now be fixed in this iteration of development for Access Aber. One of the main problems is simply with labelling of the locations on campus, it can sometimes be confusing with exactly what something is and what department it belongs to, easily fixed using marker objects for Google Maps and their name and description variables.

A feature that should help different facilities stand out is custom markers, if we use custom markers to represent different types of building, for example lecture theatres. Not only will it provide a good level of customisation to the application it will also create a familiarity feeling for someone and let them instantly identify what they are looking at.

The building display should also be able to be filtered by the user through a pop up menu. By doing this they only have to see what they want and can easily swap between views and do not have to zoom too far into the map to tell the difference between locations.

A repeating issue which we can see here is the fact that the data will most likely be updated with new additions to the campus. Again an easy fix, just have a small text file with a simple layout which stores a building's name and the category that building fits into. By doing this we make it much easier for future developers to maintain and extend the application in the future.

5.2.1.5 Multi Lingual Support

Multi Lingual Support is an area where some problems have already occurred due to what seems to be documentation that is not as clear as it could be. Android comes packaged with a set amount of locales, in this case most large countries like Spain and Germany. However it tends not to support small countries like Wales out of the box. Some ways to support Welsh seemed very poor and less than ideal solutions which could easily become broken by future changes to the way Android works.

However what seems like a solution has been found, it is possible to create a locale and set it as the user's locale on the push of a button. This should allow us to create a `strings-cy.xml` file and have that replace the standard `strings.xml` whenever a user requests it. This will require us to have `strings-en` and `strings.cy` folders however but that will not cause too many issues.

Translations will be completed by a Welsh student currently studying at the University as online translation services can be slightly off.

5.2.2 Design Rationale

As detailed above the program will be split into a set of modules, each of which has been described above. The decision to design based on feature has come as a result of several factors. One being the development style that is desired during the project, it has been decided three fully working features is much better than all of the features being there but having issues. By splitting the program up we have created an environment where no feature directly relies on another for most of its development time. The lack of dependencies between modules allows us to just focus on one feature at a time and take a style similar to FDD in the coding approach, using this document as a guide and testing after implementation.

5.2.3 Significant Classes

Several significant classes and activities are already expected, these will be detailed below. These may change further into development but should be fairly consistent.

5.2.3.1 Menu Activity

The Menu activity is something that needs to be both simple, but effective in its performance. It is a requested feature due to the feedback from users who tested the web application, being dropped straight into the map is both confusing and not a perfect entrance, it can make it difficult for a user to find what they want. A good menu that links to separate features should be implemented, it should also follow the colour scheme which will be decided during development.

5.2.3.2 Map Activity

The first map activity will solve both of the building display and route plotting functional requirements. Due to the slightly smaller expected size of these two modules it has been decided to put them into one activity and render different models based on the users request from the menu screen. This way we cut down on the amount of activities used and by rendering the model from other classes should not clutter up the map activity. Depending on what is chosen either a button which allows the user to filter the buildings will appear or a button that will let users log points and print files.

5.2.3.3 Help Activity

The map activity will simply render one of a set of texts based on which location is returned by the sort based on distance. This should include the persons name, department and contact information. Due to the application being on mobile its best a phone number is provided in case of someone really requiring the help.

5.2.3.4 locations Class

The locations class will represent a variety of information within the application, from the buildings that will be used in the building display to the buildings created to find the distance between them and the user for the Help screen. One problem which could occur is the difference between locations and Location, Location is a class provided by Google which stores among other variable a latitude and longitude, locations is essentially a modified version of that but with added features such as distance. If problems do occur the class name will have to be changed to something more suitable.

5.2.3.5 Route Finder Activity

The Route Finder Activity will contain the map for route finding, while it may have been possible to merge it in with the other map view that provides the base for building displays it has been decided due to the expected size of the route finding class that it be split up. The activity will deal displaying a route, passing on the information to figure the route out and providing a base to enter the users start and destination. The class should mostly just be method calls to other classes.

5.2.3.6 Route Choose Activity

The Route Chooser Activity should contain two Expandable List Views, the content of these will be populated by a file. This way a user chooses a route from far fewer options than they would if every route was represented as a single option. The activity will pass these details back to the Route Finder Activity.

5.2.3.7 Route Finding Class

The route finding class will take the start and destination from the Route Finder Activity and pass the finished route back to be displayed through the use of a poly line. The search will be completed using what is likely to be a Breadth First Search and will return the list of locations followed to find the final end point. It should log where each location was accessed from, this way we have a trail to follow back. The Nodes followed will be represented as a Node class which simply contains the name of the location and which file caused it to be opened.

5.2.4 Mapping Requirements to Class

Requirement	Classes providing requirement
FR1 - Route Finding - Guiding a User around campus	Route Finding Class, Route Choose Activity, Route Finder Activity, locations class
FR2 - Route Plotting - Users must be able to plot their own routes.	Map Activity, locations class
FR3 - Location Based Help - Provide details of possible help available.	Help Activity
FR4 - Building Display - A display of related buildings	Map Activity, locations class
FR5 - Multi Lingual Support - Welsh translation	Help Activity

Table E.1: Table showing mapping of functional requirements and the classes which will implement them

5.3 Important Algorithms

The following section will contain what are expected to be the key algorithms within the application, ranging from searching to plotting. Some may be either not needed or simplified in the final production. The algorithms are currently fairly vague but should serve as good groundwork.

5.3.1 Route Searching

The following algorithm should take a starting node and search for the path to the final node by continually analysing the routes (indexes) from that Node. Each Route End is another possible Node to be analysed. If the current Route being analysed is the destination chosen at the start then our search is complete and we can plot a path.

Find Route

```

1: while !route found do this
2:   for Nodes do
3:     if Node has been visited then
4:       do nothing
5:     else
6:       set Node visited
7:       get NodeRoutes
8:       for NodeRoutes do
9:         Set Opened from Node
10:        if NodeRouteEnd = Destination then
11:          set route found true
12:          call Plot Path
13:        end if
14:        if NodeRouteEnd not visited then
15:          Add to Nodes
16:        end if
17:      end for
18:    end if
19:  end for
20: end while

```

5.3.2 Plot Path

The following algorithm should take a set of visited nodes created in the search algorithm, due to knowing where each Node was opened from we can find the path back.

Plot Path

```

1: search = destination
2: for VisitedNodes do
3:   for VisitedNodes do
4:     if search = NodeName then
5:       Plot Single Route Nodename – > Nodefrom
6:       search = Nodefrom
7:     end if
8:   end for
9: end for

```

5.3.3 Plot Single Route

The following algorithm takes a single Route, not a whole path. Taking a single Route means the path can be found quickly as it is guaranteed to be in that Nodes file. It should only be called from the Plot Path function but separating it out allows for cleaner modular code. It should take two Node names as parameters. The Method 'reader' seen here will just be a file reader which loads in the routes from a file, its algorithm will not be shown here as final format has not been fully decided and reading in a file should be simple enough. The method calls Paint Path which is the method that will take the Route and place into onto the Map Fragment.

Plot Single Route

```

1: Routes = reader From
2: for Routes do
3:   if Routeend = To then
4:     Paint Path – > Route , Map
5:   end if
6: end for

```

5.3.4 Paint Path

The Paint Path method takes a route object and a map that the route will be painted on. Due to a route being made up of a set of latitude longitude points we can then paint what is called a poly line between each set of points, creating our final route. A colour is also set based on the single routes grading which is what allows us to set a grading for smaller parts of large routes.

Paint Path

```

1: Points = Route – > Points
2: for Points do
3:   Current = Point

```

```
4:   if Current position != PointsSize - 1 then  
5:       Next = Points + 1  
6:       Colour = Route - > Grading  
7:       Map - > PolyLine - > CurrentNextColour  
8:   end if  
9: end for
```

5.3.5 Create Directory

The Directory method creates a directory for saved route files to be saved into, it checks if the intended save path currently exists and if not creates it. External memory should be used if possible for easy access [9].

Save File

```

1: root = getMemory
2: MyDirectory = root/routes
3: if MyDirectory exists then
4:   Do Nothing
5: else
6:   Make Dir – > MyDirectory
7: end if
8: New File – > root
```

5.3.6 Other Algorithms

There will also be several other algorithms that could be fairly important but are small and simple. One example is the great circled function [30] which has been previously described. Documentation for the great circle function can be found on line, the version used in this application will not differ and rely on proven working algorithms.

Other small algorithms include the file reader for files containing location information and some that have to be implemented to populate expandable list views.

An algorithm that has not been fully decided on yet is how routes will be graded. This is due to the scale being currently undetermined and needing more time to perfect. As it stands steps and overall cumulative distance will be used to determine difficulty of a route. It may be possible to implement accurate elevation changes using Google's Elevation API but that will be seen further on into the project.

5.4 Design Diagrams

5.4.1 Flow Diagram

The following flow diagram displays the desired flow throughout the application including background methods.

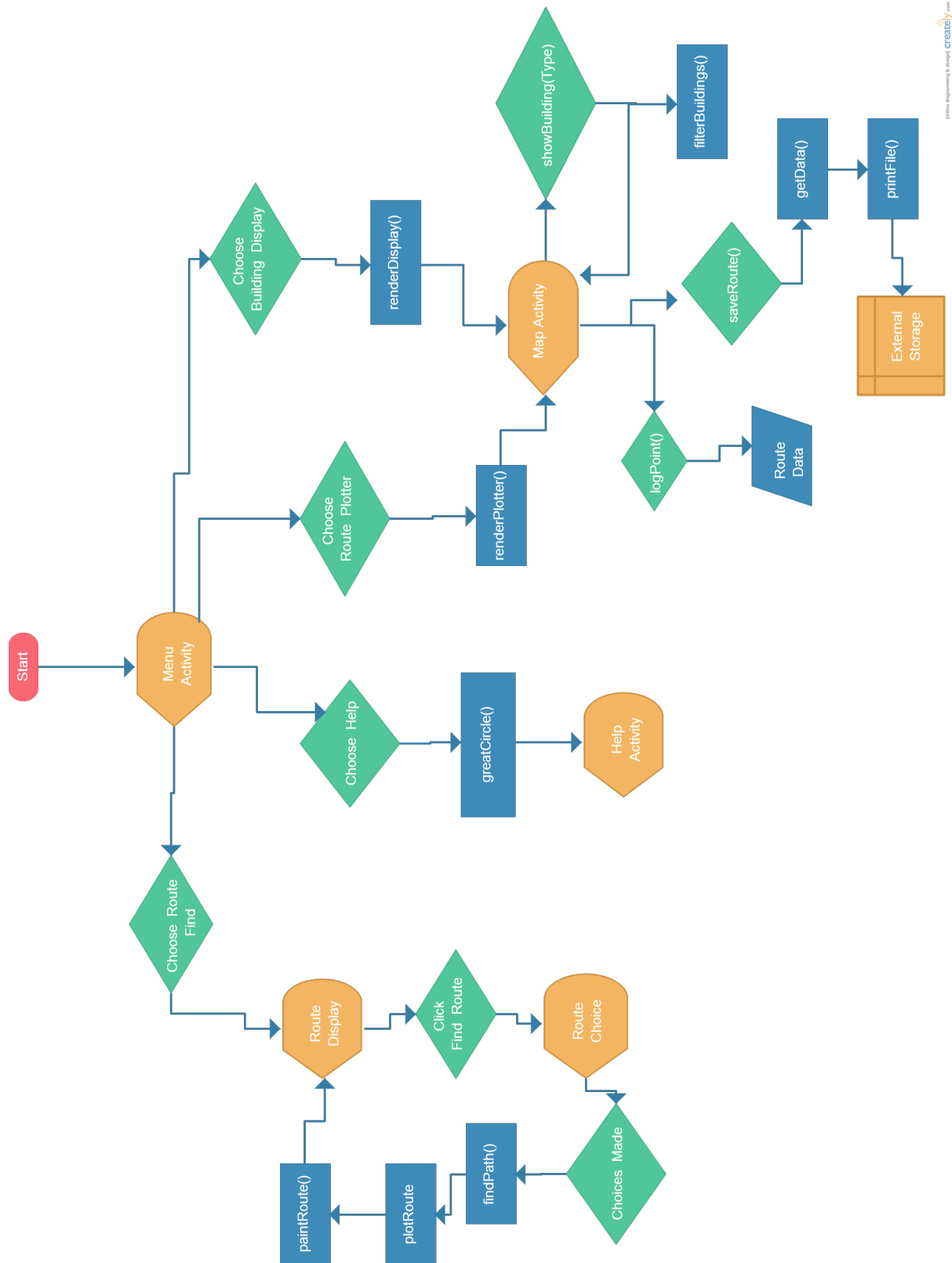


Figure E.1: Initial Flow Diagram depicting the path a user can follow through the application

5.4.2 Class Diagram

The diagram on the following page describes the estimated architecture of the application, in following is the justification of the current estimated design.

5.4.2.1 Justification

The diagram begins at what is intended to be the root of the application, which branches out into the various activities. The simplest of these activities is the Help Activity, the Help activity has an ArrayList of Place, these represent the locations which will be loaded in through the loadHelpPoints() method. These will then have a distance assigned to them using the calcDist function before finally being sorted by the sortHelpPoints function. The closest will then be displayed using the displayClosest. While split up into several methods here it may be reasonable to merge a couple of them together, this will be decided upon implementation.

The second activity is the Map activity, which will be rendered based on what the user wants. If the user wants a plotter the user will be able to log a point using logPoint which will be followed by connectPoints, causing a path to appear on the map. Finally the user will want to create a file so createFile will be called causing the creation of the File Printer class which will handle the formatting and output of the data into the correct directory.

The Map Activity will also be responsible for rendering the view for the viewing of buildings based on filter, filter will be chosen by the user. The buildings will be loaded in using the loadPlaces function which will read the buildings in. A Place also can have a Categories represented by an Enumeration, by doing this it should be easy to put a filter in the showByType function which will then display different Categories based on what the user wants.

The most complex piece of the diagram is probably the Route Display and its connecting classes. The activity has an ArrayList of Routes which will make up the entire path. This allows us to have different gradings along different parts of our final Paths. The Route Display class will start the Route Choose Activity for a result, in this case the result will be an ArrayList of Routes which will be contained within the returned Intent.

The Route Choose activity has two expandable list views, which will be populated by the available nodes to travel to in the graph, not all nodes can be travelled to. The options chosen will then be taken and a search performed using the findPath function. The findPath function will return an ArrayList of Routes which will be put into the intent and the Route Display will be returned to.

On return to the Route Display Activity the onActivityResult will be run, which will grant access to the returned Intent object and remove the extras from it using the getRoute method. The path will then be set, which is really setting a possibly long list of smaller Routes. The poly lines used to set the Routes will also be coloured based on their grading. This will be displayed to the user and they will be able to move around the route and have their position displayed using methods native to the Google Maps Android API [14].

This design will be kept to as close as possible during development, due to the design being created in such a way that it provides the modular application we want to finish with.

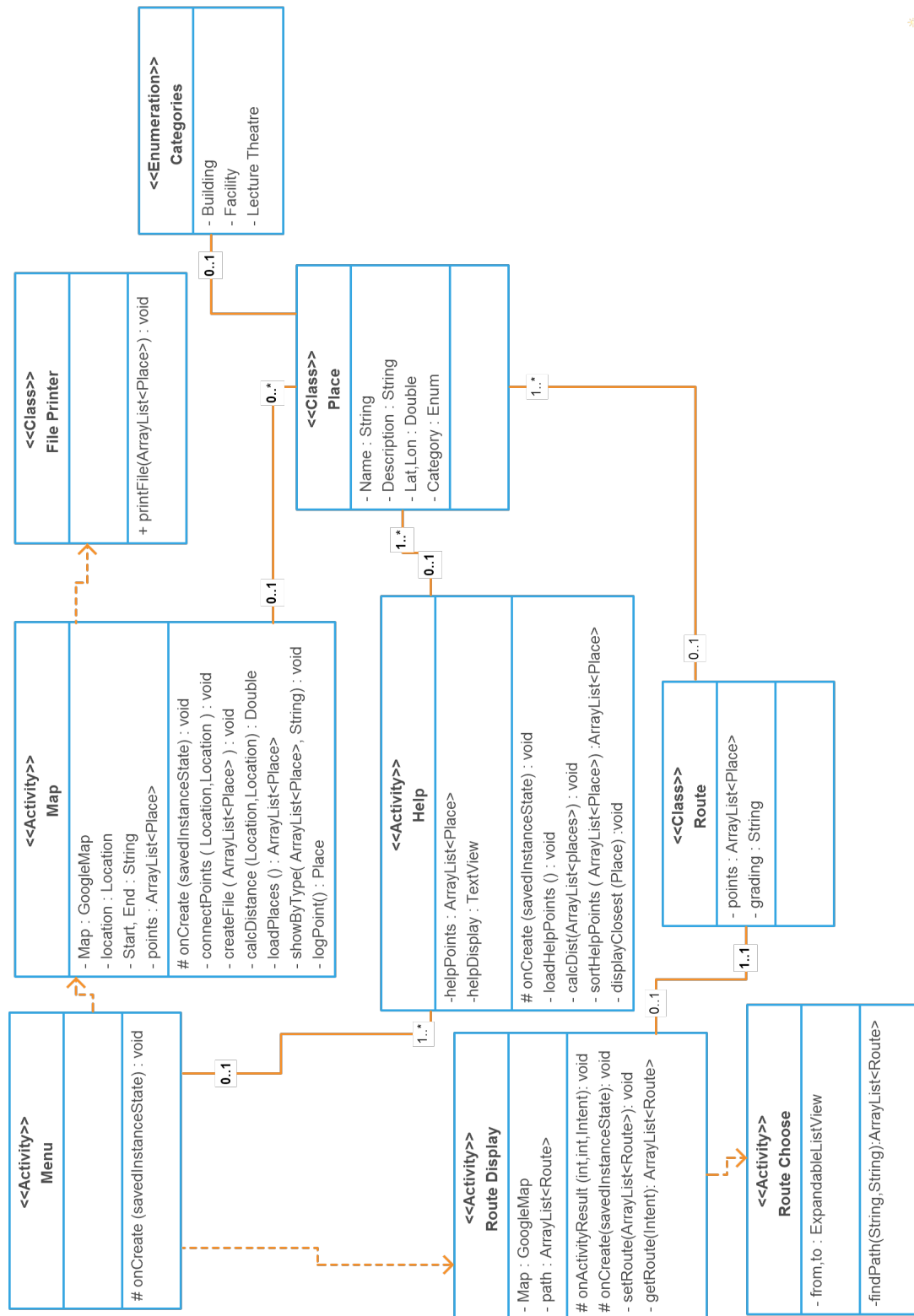


Figure E.2: Initial Class Diagram depicting the links between classes in UML notation

5.4.3 Screen Designs

The following section will include information about the planned Screen Designs and the justification for the proposed designs. Overall the screens have been currently designed to match the colour scheme of the University and as such look as in fitting with the University as it can. It should not be too much problem to have the Screens very closely resemble the designs laid out.

The button style used throughout the design images shown below will be a custom style written up in XML, various resources can be found on line for both colour picking and XML button design.

5.4.3.1 Menu Screen

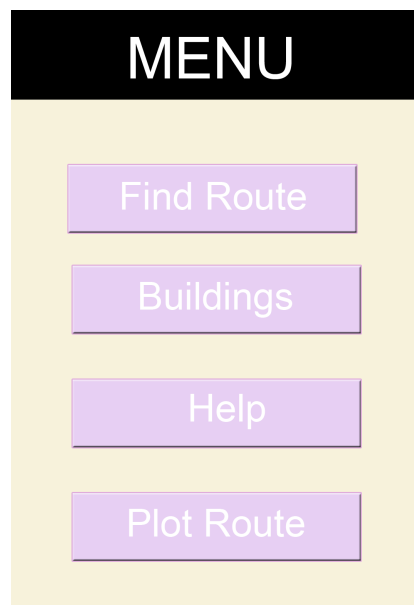


Figure E.3: Initial Menu Design depicting the predicted layout of the main screen.

As can be seen from the above diagram the user will have the label of the current page across the top of the activity, as is common within android applications. The options to move onto the connecting pages will then be arranged within a linear layout and have there respective linked activities as button text. Other options included horizontal buttons but with large screens it could look out of place and be too stretched.

Another possible choice was to have the screens swipe able, so all of the activities are essentially in a row. However after initial research the implementation of this could take long enough to slow down the development of some of the key features. Other possible choices included the inclusion of external libraries, especially some from the 'awesome-android-ui' [7] package,

however it has been decided all design will be developed without the use of outside libraries. If time is available at the end of the projects time line it could be possible to try a test of this for future development.

5.4.3.2 Help Screen

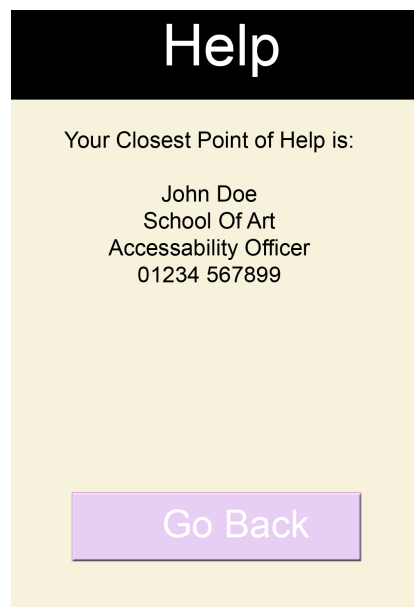


Figure E.4: Initial Design for the Help screen, displays information based on user location.

The Help screen is again a fairly simple design, the text shown will be a text view which contains the decided location with a button which takes the user back to the Menu Activity.

5.4.3.3 Display Buildings

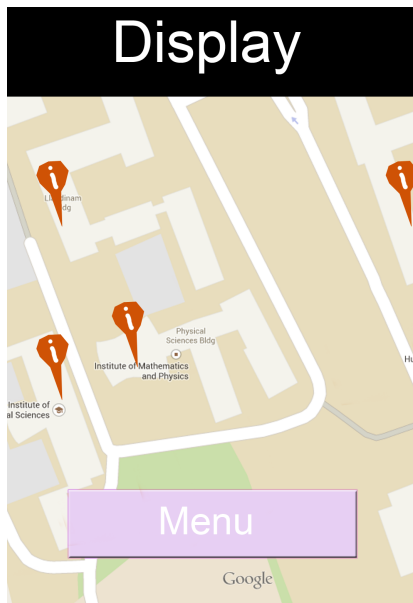


Figure E.5: Initial Display Design, view of buildings based on user filtered options.

The Display buildings screen will contain a Google Map and have a button which allows for the customisation of which types of buildings are shown. The Menu button will bring up a pop up menu which will allow users to set categories thus showing different markers and there relevant information.

5.4.3.4 Route Plotter

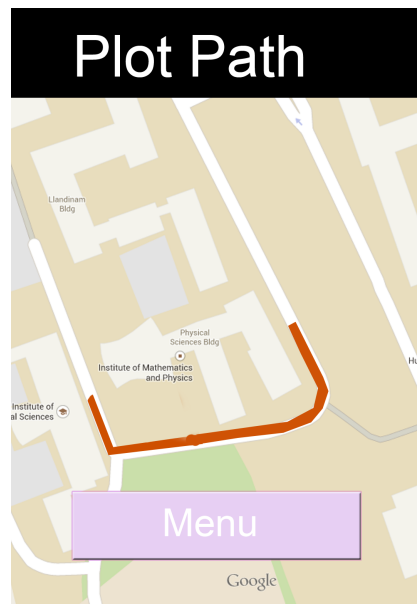


Figure E.6: Initial design for the route plotting screen for the use of future maintainers.

As can be seen from the design the screen will again hold a Menu button, with a pop up menu, which allows the user to choose from a range of options including loggings points, cancelling the current route and saving it to a file. It should also allow for incrementing steps on the route. The Route being walked can be shown on the screen through a use of a set of poly lines.

5.4.3.5 Route Display

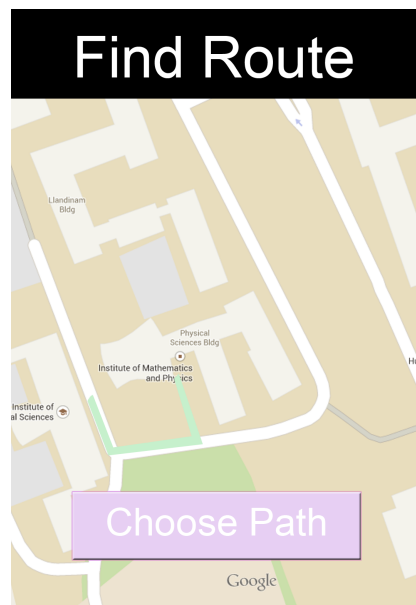
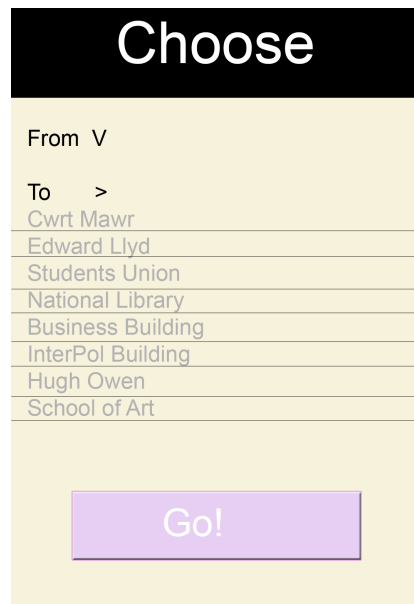


Figure E.7: Initial Design for the Route Display Screen including map object.

The Route display page the first time it is launched will hold nothing but the map object and a button, as no choice has been made. Afterwards however, as is shown here, the route will be displayed with different colours for segments that range in difficulty.

5.4.3.6 Route Choice



The image shows a mobile app screen titled "Choose" in a black header. Below the header, there are two input fields: "From V" and "To >". The "To >" field is currently selected, and a list of location options is displayed below it. The options are: Cwrt Mawr, Edward Llyd, Students Union, National Library, Business Building, InterPol Building, Hugh Owen, and School of Art. At the bottom of the screen, there is a purple button labeled "Go!".

From V
To >
Cwrt Mawr
Edward Llyd
Students Union
National Library
Business Building
InterPol Building
Hugh Owen
School of Art

Go!

Figure E.8: Initial Screen Design for the Route Choice Activity

The final screen that will be included is the route choice screen, containing two expandable list views which will then add the chosen start and destination to the extras for the activity to be passed back. The Button this time will return the user to the Find Route screen which will display the chosen route.

5.4.3.7 Design Notes

As can be seen the screens try to keep a very consistent style between them, it is felt this is necessary to provide the user with an intuitive experience. Especially in this application due to the criticisms that have been made about the Web version and our access to those criticisms. Other feedback will also be gathered from users meaning that this design could change once that has been gathered. Initial design feedback will be gathered once all screens are functional, it may not be necessary to finish features just their proposed layouts.

Annotated Bibliography

- [1] Aberystwyth University, “Access Aber,” <http://www.dcs.aber.ac.uk/~nst/accessaber/>, 2015.

The original Access Aber application, still fully working as of reference date. Developed as the initial iteration of the project with this project being considered the second.

- [2] Amit Patel, “A* Information,” <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>, 2015.

Information on the benefits of A* and possible ways to implement.

- [3] —, “Map Representations,” <http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html>, 2015.

Site containing various information on pathfinding and representation of road systems as graphs. Not all relevant but some important information was gathered from here.

- [4] H. M. Dee and D. C. Hogg, “Navigational strategies in behaviour modelling,” *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

This is my annotation. I should add in a description here.

- [5] S. Duckworth, “A picture of a kitten at Hellifield Peel,” <http://www.geograph.org.uk/photo/640959>, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.

- [6] Federal Aviation Administration, “FAA GPS Performance Analysis Report,” http://www.nstb.tc.faa.gov/reports/PAN86_0714.pdf#page=22, 2015.

GPS performance analysis, contains details on how accurate most GPS coordinates received are.

- [7] GitHub User : wasabeef, “Awesome Android UI package,” <https://github.com/wasabeef/awesome-android-ui>, 2015.

A collection of Android UI libraries, some were experimented with during development though none made it into the final version. Could be essential for future work.

- [8] Google, “Android Studio,” <http://developer.android.com/tools/studio/index.html>, 2015.

Home page of Android Studio the IDE used for the majority of development and the official IDE of Android.

- [9] —, “Data Storage in Android,” <http://developer.android.com/guide/topics/data/data-storage.html>, 2015.

Details on Internal and External Storage. Description of what they mean and who can access them. A useful resource in solving issues through the development of file saving

- [10] —, “Eclipse ADT,” <https://developer.android.com/tools/help/adt.html>, 2015.

Description of ADT and install instructions. Includes information regarding its deprecation and the full release of Android Studio

- [11] —, “Elevation API,” <https://developers.google.com/maps/documentation/elevation/>, 2015.

API for the elevation service provided by Google, not used in full application but could be essential for future development.

- [12] —, “Google Directions API,” <https://developers.google.com/maps/documentation/directions/>, 2015.

API page for Googles Directions API, allows for the loading of directions from any one point to another.

- [13] —, “Google Maps,” <http://www.google.co.uk/maps>, 2015.

Google Maps was used to provide the Map objects within the application. It was a key reason for the success of the project.

- [14] —, “Google Maps Android API v2,” <https://developers.google.com/maps/documentation/android/>, 2015.

Base screen for the Google Maps Android API v2. Contains information relating to map gathering, manipulation and further details. Large amounts of the applicaiton are based on information gathered from here.

- [15] —, “Google Material Design Guidelines,” <http://www.google.com/design/spec/material-design/introduction.html>, 2015.

Homepage for Google’s Material Design Outline, discusses the ideology behind how an Android application should look and the reasoning.

- [16] —, “GoogleMap Object Documentation - setLocationEnabled,” [http://developer.android.com/reference/com/google/android/gms/maps/GoogleMap.html#setMyLocationEnabled\(boolean\)](http://developer.android.com/reference/com/google/android/gms/maps/GoogleMap.html#setMyLocationEnabled(boolean)), 2015.

Information on the setLocationEnabled function provided within the maps API, including details on parameters and function.

- [17] —, “Introduction to Android,” <https://developer.android.com/guide/index.html>, 2015.

Home screen of the Android API. Base of all development, within all details can be found on native methods used. Essential to the project.

- [18] —, “Poly Line information.” <https://developer.android.com/reference/com/google/android/gms/maps/model/Polyline.html>, 2015.

Developer page for the polyline functionality within the Android API, use repeatedly through development for visualization of routes.

- [19] Gradle Inc, “Gradle User Guide,” <https://gradle.org/docs/current/userguide/userguide>, 2015.

User instructions for the use of Gradle, a build tool included within Android Studio used for the project.

- [20] Khan Academy, “Uses and description of BFS,” <https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/breadth-first-search-and-its-uses>, 2015.

A descriptive page about BFS and its implementation, describes the process in fairly simple terms before moving onto more detailed discussion

- [21] Microsoft, “Outlook website,” <http://www.outlook.com>, 2015.

Used as inspiration for one of the possible colour schemes laid out in the design section.

- [22] Mind Freaker Stuff, “Custom XML for Android buttons.” <http://www.mindfreakerstuff.com/2012/09/50-useful-android-custom-button-style-set-1/>, 2015.

Used as a base for the custom buttons included within the application, good resource for future development.

- [23] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, “Don’t touch me, I’m fine: Robot autonomy using an artificial innate immune system,” in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.

This paper...

- [24] OpenStreetMap Contributors, “OpenStreetMap,” <http://www.openstreetmap.org>, 2015.

Open Street Map is a map service built through contributions, all data is available freely and as such was considered for the map service within the Application

- [25] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.

This is my annotation. I can add in comments that are in **bold** and *italics and then other content*.

- [26] Stack Overflow, “Examples of Map representations,” <http://stackoverflow.com/questions/177343/map-navigation-project-how-is-road-data-generally-stored-represented>, 2015.

Stack Overflow post with various references to material which describes implementation of graphs as a representation of maps, while the implemented system was simpler it still took some ideas from here.

- [27] —, “External Storage check code,” <http://stackoverflow.com/questions/7616974/how-to-check-internal-and-external-storage-if-exist>, 2015.

Solution for checking if external storage is accessible or not, was used as a reference when drawing up plans for future features

- [28] —, “Great Circle Distance - Java Solution,” <http://stackoverflow.com/questions/3694380/calculating-distance-between-two-points-using-latitude-longitude-what-am-i-doing>, 2015.

Solution the great circle function implemented in Java. Takes elevation into consideration, something which was unused in this programs implementation.

- [29] U.S. Government, “GPS Performance Accuracy,” <http://www.gps.gov/systems/gps/performance/accuracy/>, 2015.

Official information page about the GPS system and its accuracy. Includes information on accuracy and the difference between civilian and military GPS.

- [30] Wikipedia, “Great Circle Distance,” http://en.wikipedia.org/wiki/Great-circle_distance, 2015.

Overview of the Great Circle distance and some algorithms that return it. Explains the use of the formula due to there being no straight lines on a sphere.