KING'S College LONDON

University of London

6CCS3PRJ Final Year
LOST

Final Project Report

Author: Joshua Simpson

Supervisor: Andrew Coles

Student ID: 1225043

April 25, 2015

Abstract

Navigational tools have been developing rapidly over the past couple of decades - at the moment, capable of providing us with a location within a few meters of where we are. Unfortunately, they still have not progressed to a point where a user can use location-based services from within a building to an accurate degree - an advance that would open new avenues of development in contextually aware services

This report showcases an extension of works that use wireless access point data to provide contextually aware location based services at a building - the end result in this being a mobile application package providing location information and route-planning functionality, whilst also crowd-sourcing contextual data. The main aim of this entire package being to demonstrate the value of contextually aware systems to different types of users.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Joshua Simpson

April 25, 2015

Acknowledgements

Dr Andrew Coles - for his patience, expertise, and lulzy memes.

The Informatics Department - for their fantastic support for all my endeavours throughout the year, and for a well-grounded education in recalcitrant platypodes - I am now more than prepared to face the obstinate platypodes of the world. Also, because the bots kept me entertained.

The Hackathon Community - for their amazing support through difficult times.

Contents

Chapter 1

Introduction

Location based services have been developing rapidly over the past few decades, and are used in various aspects of technology - ranging from the route-planning software that is used to figure out a morning commute, to live time-tabling services that use bus or train positions to give accurate estimates on arrival times, to advertising based on the user's current location. It is safe to say that in some way or another, location services are prevalent in modern day life.

However, GPS[1] -the most prominent form of location detection - only works to a certain level; specifically to within a few feet at street level[6]. Whilst achieving this level of precision a feat of engineering, it leaves a large market gap for providing truly localised contextually aware services.

Contextually aware services are another rapidly developing technology, providing users with information or activities specific to their current location or situation, determining the suitability for such events by using sensor data[2].

There are previous studies on using local area wireless access points to provide this sensor data at a level more precise than GPS can currently allow, but they are usually run in controlled environments, deriving a single mathematical equation in order to determine distance from the access point with high precision[5]. Unfortunately, this is impractical to apply as general use outside of a test environment.

[1] Global Positioning System - a system using satellites to provide precise location information
[2] Data such as location, temperature or time

1.1 Project Aims

The main goal of this project is to provide a proof-of-concept mobile application that can provide location and route-planning services whilst also delivering contextually aware services in a specific building - all without using GPS - demonstrating the possibilities that this area of application development holds for developers. Around this main feature, the project will aim to create software suite allowing for the efficent setup of a 'contextually aware building', as well as provide practical functionality to a variety of user groups (by using the application to crowdsource data), to demonstrate the potential in this field. To achieve this, I will be collecting data on user location and wifi failure[3] throughout the building and providing a backend that displays this data in a useful manner.

To demonstrate a successful project, a large building with many potential locations is useful, as it shows the accuracy of of the location-finding and route-planning algorithms. To this end, the building that I am using to test this application is King's College London's Strand Campus[4]. With roughly 17,000m2 of space and 9 floors to utilise, the application would not only be thoroughly tested but also most likely welcomed by King's 25,000 students[12].

1.2 Report Structure

This report will begin with a background on the project - taking a look at location based services and contextually aware services and the applications of both that are already available, along with an analysis of projects or papers that have attempted to solve the problem of location services at building level. This data will then be collated to analyse key problem areas during implementation.

Following this there will be an outline of the requirements and specification for the project - showing both the project itself and the chosen extensions - complete with justifications on decisions made in the specification. Next, the implementation of each component in the project will be described and analysed.

Finally, there will be an evaluation of the final implementation, comparing the result against the expectations set out in the Design chapter. This will be followed by a conclusion containing final thoughts about the project, and detailing future work that would follow as a result of work done in this project.

[3] A common occurence in the chosen building

[4] It has not escaped my attention that this will also help all the new first years find Waterfront

Chapter 2

Project Background and Main Points to Consider

2.1 Location Services

Defined as *"services that integrate a mobile device's location or position with other information so as to provide added value to a user"*[17], the origins of location information services date back to 1973, when the US Department of Defence developed GPS to overcome the limitations imposed by navigation systems in use at the time[14]. This original system has since been developed into an integral part of many mobile applications, including mapping and route planning applications, as well as forming the 'sensor' in a lot of contextually aware applications. There is a well established method that is used in order to ascertain location, called trilateration[1]. Triangulation[2] is another method used to determine location, but is mostly inapplicable in the context of mobile devices[3]. Whilst I will not be developing an application that directly uses existing location technologies such as GPS, it is important to consider the practices used in such technologies when developing this project.

2.1.1 Application: Google Maps

Google Maps is easily the most widely used mapping tool on the planet, with over 54% of smartphone owners using the app on a regular basis in 2013 (for comparison, this was 10% more than Facebook at the time)[15]. Google Maps are a chief provider of location information

[1]The process of finding a location by measuring distances, using geometry
[2]The process of finding a location by estimating the direction in which signals are coming from
[3]It is also commonly mistaken for trilateration

services, with advanced route planning and street-view functionality. They serve this data through their own API[4]. The most interesting aspect of their service (in relation to this paper, at least) is their recent addition of the use of Wireless Access Points for their location services. Whilst they do not rely solely on it for location, they do use the data to increase accuracy and speed through their Maps application[10]. Interestingly, Google collect this WiFi data by crowdsourcing through an 'opt out' scheme in the Google Maps application[19] - this may not be possible in the case of this project (as Google have used WiFi in a more assistive manner), but it provides a good basis from which to approach the problem.

2.1.2 Paper: Indoor Location Using Trilateration Characteristics

[6]

This paper is especially relevant to the problem, considering the use of Wireless Local Area Networks in order to achieve trilateration. This is achieved by measuring the signal strength for at least 3 access points, and using the signal strength to determine the distance from the access point. Once you have three accurate distances you can essentially create a 'Venn Diagram' of where the user is located.

The paper goes on to state that there are inherent inaccuracies with this positioning technique, due to the variable nature of radio signals[6].

To circumvent these inaccuracies, an elegant averaging method was developed, which started returning an accurate 'average signal strength' after 40 readings.

Issues with the trilateration approach

- As stated above, the trilateration approach has difficulties providing accuracies:

"Radio signals are extremely variable, particularly indoors, due to being reflected by obstacles or refracted round corners, known as multipath reflection. Environmental changes can also affect the signals, such as the number of people around. This means that the positioning technique is inherently inaccurate, with positions from raw Wi-Fi signals being in excess of 10m out. "[6]

- Whilst there is a solution designed to solve this problem in the case of the test environment, an application in a building such as Kings would require a number of different formulae in order to provide accurate signal strength averaging for different areas of King's[5]

[4] https://developers.google.com/maps/

[5] Because of the variables such as distance between routers, building material, and the number of users in any area at a given time

- Finally, in cases where the averaging solution is easily applicable, there *should* be concerns regarding the impact on a mobile device's battery life when performing repeated scans and operations on the results of those scans.

2.1.3 Paper: More Stuff on Location - Fuzzy Location?

2.2 Context-Aware Services

Contextually aware services are services that use data that pertains to the user or application's current situation to provide actions or functionality specific to that scenario. It is defined by Abowd as

"any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves." [3]

Contextually aware services use 'sensors' to fetch this information - examples include location data, temperature, even the camera on your phone could be used as a sensor for contextual services[6].

2.2.1 Estimote

A rapidly accelerating tech start up, Estimote provides 'beacons' using low-power Bluetooth in order to send an ID[7] to an Estimote enabled application - these can then be programmatically associated with locations, events, etc. and calculates its distance from the phone using RSSI (the received signal strength) inside it's own API, meaning that it can be used very effectively to provide location-based services with minimal effort. This is so far the the most prominent product in the market using the iBeacon base.

Currently the major drawback of the Estimote is the price, which limits Estimote beacons / stickers to small buildings âĂŞ covering an area such as KingâĂŹs would be incredibly costly[8], and any structural changes regarding the beacons would require significant effort to represent in an application developed to use Estimote. Another issue is that - as stated above - using RSSI to fetch location information can become very complex in a large building with lots of people in it (which would cause signal attenuation at different levels throughout the day).

[6] Consider how recent smartphones have software that stops the screen coming out of standby if it thinks it is in your pocket

[7] `http://estimote.com/api/getting-started/intro-to-beacons.html`

[8] `http://estimote.com/#jump-to-products`

2.2.2 Google Maps

As part of their vast toolset, Google Maps also has context-sensitive functions in it's API - an example is their 'Explore' feature[9], which will show destinations of a certain type that are in the user's local area. The Explore function even takes into account time of day and weather in order to provide the best possible locations to the user.

An interesting point to note is that the 'Explore' function was predated by a 'Search Nearby' function that was removed[10] - possibly due to privacy concerns[11].

2.2.3 Paper: An Architecture for Data Collection and Processing in Context-Aware Applications

[16]

In this paper, Dario Salvi *et al* pose that whilst Context Aware systems are showing and developing a greater use in general applications, that the field is not "applied in the real world"[16]. The paper then goes on to propse an architecture that access and uses contextual data differently to provide the developer with more possibility to develop what is referred to as 'real world applications'.

Afterwards, the writers go on to set out three classes for collecting data - AtomicData[12], ContextData[13], and Relationships[14][16]. These classes provide an interesting base for the data collection part of this project[15].

Overall the paper presents some very useful points on using contextual data to return usable information.

2.3 Crowd Sourcing

Crowd sourcing is generally defined as using a group of people or community to generate a needed resource. It is mostly thought of in terms of popular crowd-funding sites such as Kickstarter, or Indiegogo, but the term extends to other resources, such as data or ideas[16]. Generally, data crowdsourcing follows a pattern of clients paying for information that they

[9] `http://google-latlong.blogspot.co.uk/2014/07/spend-more-time-exploring-with-google.html`
[10] `http://www.jsonline.com/blogs/news/247119811.html`
[11] Covered later in Ethical Concerns
[12] Singular pieces of information
[13] Compositions of more than one piece of AtomicData
[14] Links between two ContextData
[15] ContextData holds the most interest, as I will need to collect multiple pieces of data
[16] For example, Cracked.com - a satire website - uses its own staff, but also looks for stories contributed by it's readers

need, and users that receive rewards to collect or generate this information.

2.3.1 Captcha

Captcha was originally invented as an identification system to help discern humans from robots when performing tasks online. This started out as just a verification system (which is growing more and more elaborate as robots evolve), but was eventually evolved into an aid to digitizing books[17]. The most interesting point about this is that most of the users of Captcha have no idea that they are crowd-sourcing a huge effort every time they solve a Captcha.

2.3.2 The 90-9-1 Rule for Participation Inequality in Social Media and Online Communities

[13]

This article addresses a major problem with crowdsourcing data which Nielsen refers to as participation inequality, stating that user participation often follows a 90-9-1 rule, with 90% of users being "lurkers or users that will observe but not contribute"[13] . Nielsen also covers how to overcome this participation inequality (after briefly summarizing with âĂŸyou canâĂŹtâĂŹ); one method in particular is of interest when considering the aims of this project and its associated application: 'make participation a side effect' âĂŞ essentially the act of collecting data without the user having to do anything - this is less intrusive to the userâĂŹs day-to-day use of the application, but can also have obvious ethical ramifications if the user is unaware if they are sourcing data.

2.4 Platform

Because of the nature of the application being developed, it makes the most sense to develop for a mobile device - whilst devices like laptops offer more power to process information, the dataset that needs to be taken does not require the extra resources. Alongside this, mobile devices lend themselves better to the data-collection aspect of the project, with background services that can perform operations whilst the phone is inactive in a user's pocket. Finally, it is easier for a user to take out their phone in order to ascertain their location than it would be for a user to take out their laptop to perform the same operation.

[17] http://www.npr.org/2013/10/04/191620023/can-you-crowdsource-without-even-knowing-it

With this in mind, the choice of platform then comes down to which type of mobile device to develop for - Android devices, or iOS devices. Whilst I could develop for both device types simultaneously using a platform such as Phonegap[18], I have concerns about not using the native SDK for a platform - the abstraction provided by platforms such as Phonegap may reduce or restrict functionality that is necessary during development[19]. In the end, I decided to develop this for the Android platform alone.

2.4.1 Reasons for Android as a platform

The first point I considered whilst choosing a platform to use is the popularity of a platform - it makes sense to make the application available to the widest audience possible. In this case, Android dominates the market share, with 76.6% of smartphones using the Android OS, compared to 19.7% using iOS in the last quarter of 2014[11].

Another major concern when deciding upon device was the functionality provided by SDKs for both Android and iOS. When I reviewed the functionality for both devices, I was unsurprised to find that iOS does not allow access to lower-level functionality. What did surprise me is that statistics like wireless signal strength was amongst these lower level functions[2][1][20]. This restriction of access was a major concern[21]

2.5 Ethical Concerns

When developing an application that uses any form of personal information, there are definite privacy concerns. This is exascerbated when considering that there will be location information being returned to a server that is accessible by staff members. In this respect, it is important to not only adhere strictly to Data Protection laws, but also to consider the user's privacy and wellbeing. To this end, I need to make sure to collect the least amount of data I can, and to anonymise this data as much as possible in order to avoid sending any user details that would affect their privacy. Alongside this it would be prudent to secure the repository of any stored data against unauthorised users.

Another ethical point to consider is the code that is used to create this application. I should take due care to create this app using only my own code, except where express permission has

[18] http://phonegap.com/

[19] Phonegap as an example, uses HTML, CSS and Javascript to create an application for both platforms simultaneously

[20] Whilst StackOverflow is not preferable as a citable source, finding an admission from Apple on this point is very difficult

[21] Unfortunately, later modifications to the application to circumvent issues with using signal strength meant this was no longer an issue

been granted. In the same vein, I must make sure to adhere to all licenses for any libraries which I use in development.

Chapter 3

Requirements

The client end of this project consists of two parts - an Android application that can determine user location (LOST) and provide route-planning inside the building, whilst feeding data to the backend, which will store the data and provide functions to view it as information. During implementation, I later developed a secondary Android application (FOUND) so that I could easily collect wireless data to test the application with. FOUND will not be available for users of the resulting software package for this project, however when considering possible extensions after the BSc report, FOUND would be an active component to allow for the easy mapping of a building in a bespoke software package. Because of the late addition of FOUND to the project, I will cover it briefly in Design, but give an in-depth analysis of the application in the Implementation section of this report.

3.1 LOST - Android Application

3.1.1 Functional Requirements

F1 - The application will be able to retrieve the most recent SQLite database containing wireless access point location information upon startup.

F2 - The application will be able to accurately discern the user's location, using data provided by an SQLite database.

F3 - The application will be able to provide the user with route-planning data, allowing the user to select a start and an end point, and provide the shortest path between the two points.

F4 - The application will be able to discern whether the device is currently in the correct building for usage.

F5 - The application will send anonymous information on the user's location to the backend periodically, if it is in the correct building

F5.1 - The user will be able to turn off this functionality

F6 - The application will periodically check if it has a functioning internet connection - if it is in the correct building. If the internet connection is malfunctioning, it will save this data.

F6.1 - Once the application regains internet access, it will send location data, and the MAC address of malfunctioning router to the backend.

F6.2 - The user will be able to turn off this functionality.

3.1.2 Non-functional Requirements

NF1 - Application will not cause an 'ANR' (app not responding) error on any 'low-end' smartphone (at time of writing, low end specifications are considered to be 1Ghz dual core processor and 768mb RAM[1]).

NF2 - Application will provide location accurate to within 20 feet.

NF3 - Application will use as little battery life as possible, using less than 5% battery life within an hour.

NF4 - Application will minimise data usage, making sure not to post data back using mobile data.

NF5 - Application will not send data that can be used to identify the user, or any information that may be personal to the user / a breach of Data Protection laws.

NF6 - Application will be usable on android versions 4.0.0 upwards, covering more than 90% of android devices[7].

[1]Figures created by comparing phone specifications at `http://www.phonearena.com/news/Affordable-not-cheap-15-best-low-cost-smartphones_id58696`

3.2 Backend

3.2.1 Functional Requirements

F7 - Backend will operate a RESTful API that will allow the mobile application to post data to the server

F8 - Backend will store POSTed data to an SQL database

F9 - Backend will be retrieve data from SQL database, and display to user as information

F10 - Backend will require user authentication to allow access to displayed data

F11 - Backend will be able to store wifi location database, and serve it to a mobile device

3.2.2 Non-functional Requirements

NF7 - Backend will provide an easily navigable user interface

NF8 - Backend will show data in an easily readable manner

3.3 FOUND - Android Application

3.3.1 Functional Requirements

F12 - Application will scan local wifi access points

F13 - Application will insert the top access point into an SQLite database as a node, along with an edge connecting it to another node in the directed graph

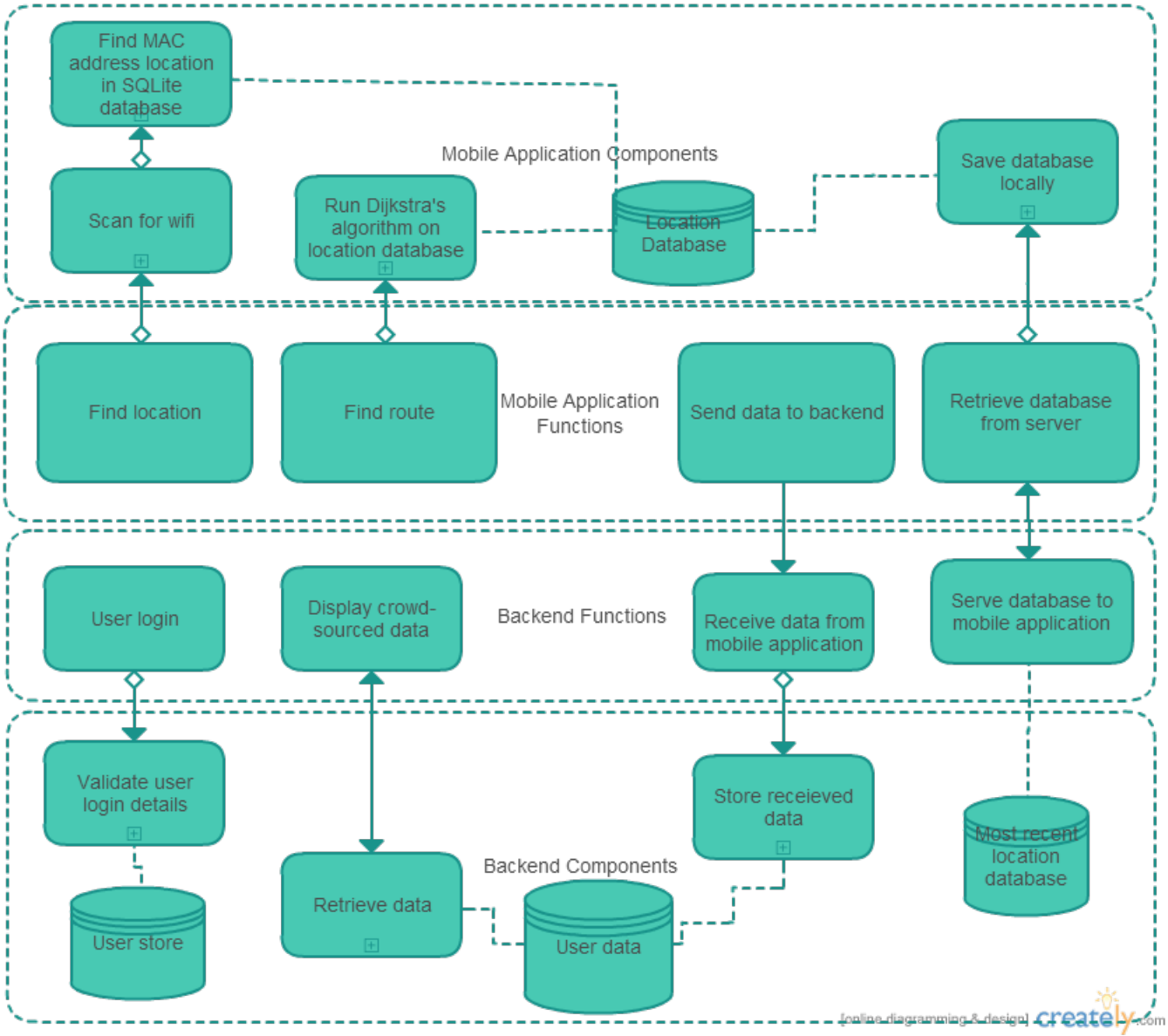
Chapter 4

Design

The application is split into two parts - therefore it will be detailed as a whole product, before the design of each component is analysed individually. Generally, the backend and the frontend application work as their own components, with the only interaction between the two being fetching the database, and sending data from the application to the database; because of this, an agile development cycle will be used.

Further to this, a basic version of the Android application will be developed first, followed by an inital deployment of the backend, with the ability to receive data from the mobile application after that - from then, incremental additions will be made to either component, building for functionality first, then returning later for optimisation.

4.1 Component Diagram



4.2 LOST

The 'front end' of the project, the LOST application provides most of the functionality detailed in the earlier chapters of this report. In this section, I will go into more detail as to what each function will need to do in order to provide the outlined functionality. You will find a Use Case Diagram, along with wireframes for each aspect in the Appendix. For clarification, each 'view' in an Android Application is called an Activity.

4.2.1 Finding Location

This is one of the most critical aspects of the application, and also one of the most challenging as a great deal of consideration needs to be paid towards efficiency, accuracy, and precision - it will generally be more expensive in terms of resources, and in fact battery power, to increase the precision.

In order to ascertain location, the application will need to perform a scan of nearby wireless access points. The MAC address and signal strength of each access point will be collected, and put in descending[1] order of strength. The original intention[2] at this point was to use actual signal strength figures to get the most precise location information, as done by Cook *et al*[6].

To get this location information, the top three MAC addresses in the list would be used to filter the location database down to a single node, representing that location[3]. This location would then be displayed to the user, along with an option to refresh this (performing another scan, incase the user doesn't think they are in the right location). There will also be a button that allows the user to use their given location as the start or end point for route planning.

4.2.2 Route Planning

This function ties in with the location finding function of the application - the user will either be able to load it normally, or load it from the 'Location Finder' part of the application - if this is the case then either the 'to' or 'from' sections of the application will contain the user's location.

Regardless of how the activity is accessed, there will be two drop down lists, allowing the user to select the start and end points. The application will translate these selections into nodes in the directed graph created from the location database, and perform Dijkstra's algorithm to

[1] Or considering how signal strengths are measure, ascending. Signal strength is measured in DBm and measure as negative figures. The 'quieter' a signal is, the weaker it is - therefore a signal strength of -10 is much more powerful than a signal strength of -80

[2] Results of this will be detailed in the 'Implementation' section of this report.

[3] Discussed further in the Databases section

find the shortest path between the two locations. The nodes traversed in this path are displayed in order on the next Activity to show the user the path they need to take as a step-by-step route. A possible extension on this once functionality has been implemented would be to have each step highlight green when the user reaches the associated location, however this would most likely be done through repetitive wireless scanning, so steps would need to be taken to ensure that this doesn't rapidly drain battery.

Choosing an algorithm

Because of the limited power available to a mobile device, alongside the short attention span of users[4], it was important to select the most efficient algorithm for the problem being faced. Another algorithm which could have been used in this instance would have been Bellman-Ford's algorithm. After research into, and a comparison of, the two algorithms it is shown that Dijkstra's algorithm is much more efficient than Bellman-Ford's[18], with the latter being used only in the case of directed graphs with negative edge weights (a situation that Dijkstra's algorithm is not equipped to deal with) - as there are no negative edge weights when travelling from one area to another, Dijkstra's algorithm is clearly the best choice.

4.2.3 Recognising access point faults

One of the chosen forms of data to upload is faults with access points, as this shows a useful application of the concepts that form the foundation of this project. In order to retrieve this data, the mobile application needs to be able to recognise when it is in Kings, when it is using a Kings wireless access point, and when that access point is having issue. However, there is an obvious issue when trying to upload data about malfunctioning access points - the lack of internet. To solve this, we need to add functionality for caching data.

Caching Data

Because there will be times when collected data can not be sent back to the server (either through malfunction or lack of connection), it is important to keep the data - making sure it is properly timestamped for when it is uploaded - and then upload it when a connection becomes available again. An important problem to consider here is that data may become irrelevant if it is uploaded after too long a period of time, so it is worth looking into functions that clear items of a certain age out of the cache.

[4] And indeed Google's limit before declaring an application unresponsive

4.2.4 Data Upload

This function will run at regular intervals in the background, regardless of whether the app is open, as the chances of the user having the app open all the time are very slim - studies show that an average app sessions lasts barely more than a minute[4]. Because this will be a repeating task running in the background at any time, it is imperative that the task is threaded properly, so as to avoid blocking the main thread with what could potentially be a long task.

Also important is to consider how regularly the information is sent - if the information is sent too frequently it will be a needless waste of resources, and will leave the resulting data set far larger than it needs to be, but too little and the resulting data set will be insubstantial. It was decided that an interval of 15 minutes is appropriate, as it is frequent enough that people may have moved areas, but infrequent enough that it won't use too many resources.

4.3 Backend

The backend of the project, the web server will receive and present crowd-sourced data to authorised individuals. As a secondary function, it will also serve the locations database up to the mobile application - allowing for the storage and retrieval of the latest version incase the architecture of the building changes; either physically (which is unlikely), or logically (where wireless access points are added, changed, or removed). A critical aspect, in order to maintain trust, and compliance with Data Protection laws, along with the BCS Code of Conduct, is to limit the access to the data to only those who need it.

4.3.1 Public API - Receiving data, and serving the location database

The backend will need to present a RESTful API, that the mobile application can POST data to. Once data has been POST'ed to these API endpoints, the server will need to validate the data before finally storing it. It is important that the API isn't subject to blocking - whilst efforts will be made to stagger uploads from the user end [5], it is important to make sure that the server deals with them all in a timely manner, and can handle lots of operations simultaneously.

Another function of the API will be to receive requests for the locations database, and serve that database to the mobile application - usually for the first startup of the application, but also if any data in the database changes.

[5]Spreading operations instead of having the server deal with all of them at once

4.3.2 Data Presentation

The server also needs to be able to retrieve the data it has stored and present it to an authorised user in a way that converts it from *data* into *information* - this means presenting it in a manner that is easy to read, and provides some function to the user. Originally, it should be able to just display the data to the user in a list (where an access point has a fault / when people are in a location) - but this should be expanded on, providing an analysis of the data and providing it in different forms, such as charts showing the usage of an area over a particular amount of time. An important extension to this would be to provide the user with a way to export the dataset out, so that they can generate information suited to their own needs.

4.3.3 Security

It is important that all the data on the server is only accessible by authorised users - this means that a login should be required to access the site provided by the backend, along with any databases / data sources that it utilises. Further to this, any data that is sent in should be completely anonymised - whilst this is mostly a function of the mobile application, it would be pertinent to maintain this vigilance in all areas of the project. An example of how to anonymise the data at the backend would be to make sure that no data regarding the origin of the data is stored beyond that which is expected. If the aforementioned data does need to be stored, then the same standard of security applied to the rest of the backend should also be applied to this data.

4.4 Databases

At the end of this product, there will only be two databases in use - the locations database served by the backend and used by the mobile application, and the database that will store the data uploaded by the mobile application. The structure of these databases needs to be clearly defined as early as possible, so as to avoid having to make structural changes during development.

4.4.1 Locations Database

The locations database will comprise of two tables - Nodes, and Edges, and will allow for the building of a directed graph in-application that can be used to find the shortest path to a location.

Nodes

The nodes table will contain nodes that represent each location that has been mapped in Kings. Its fields will be:

ID - ID of node, stored as an integer

Name - Name of location, stored as a string

MAC - MAC address of the corresponding wireless access point[6] - stored as a string

Edges

The edges table will contain edges that represent connections between nodes. Its fields will be:

ID - ID of edge, stored as an integer

StartNode - The ID of the node that the edge begins at, stored as an integer

EndNode - The ID of the node that the edge ends at, stored as an integer

Weight - The amount of time taken to get from the StartNode to the EndNode, stored as an integer.

Method - Not definite, as could be implied by the weight, but the method used to get from one node to another (either walking or taking the lift in the case of KCL). Stored as a string.

4.4.2 User Data

This database will contain the data that has been uploaded as people have used the mobile app - the data contained in this database will vary from table to table, depending on the sort of data that we want to collect from the user. Because of this, we will define a couple of example databases, as the other possibilities for collected data are endless.

[6] Originally three MAC addresses for precision, but reduced to one after encountering issues as mentioned in 'Implementation'

Location Reporting

An example database showing the locations / facilities in Kings that are being used in K-ings, demonstrating the usability of both crowd-sourcing data and contextually aware mobile applications.

ID - ID of information upload, mostly for indexing purposes. Stored as an integer.

Location - A string showing where the user was when the data was uploaded. Stored as a string.

Time - Date / Time that data is recorded (which may well be different from the time of upload), stored as a string.

Access Point Recording

ID - ID of information upload, mostly for indexing purposes. Stored as an integer.

MAC - MAC address of reported access point. Stored as a string.

Location - Location of issue. Stored as a string.

Time - Date / Time that data is recorded. Stored as a string.

Notes - Notes about the issue - to be used by users of the backend. Stored as a string.

Time uploaded - The time that the data was pushed to the server. Stored as a string.

Chapter 5

Implementation

In this section I will cover the implementation of the various parts of the project in greater detail, explaining the various components that make up these parts, and the choices behind them.

5.1 LOST

5.1.1 Toolkit

Languages Used

As an Android application, LOST was written almost completely in Java, using XML for layout files and SQL to manage the database. Whilst there is the option to use C / C++ to write directly for the Linux operating system that Android is based on, it is not recommended by Google.

Development Environment

When I started developing LOST, I was using Eclipse - however about a month in I switched from Eclipse to IntelliJ for several reasons. Eclipse seemed to be inherently bug-ridden, requiring a restart frequently which was disruptive to work-flow. Alongside this, IntelliJ made much better use of available plugins - including plugins that provided great Git integration, which I used for version control.

Whilst developing, each build was tested on two Android phones - one older, now low-end phone, and one much more recent high end phone, providing a good analysis of application performance over different devices.

Libraries / Frameworks Used

Whilst developing LOST, I used three different libraries - two of which were critical to the implementation of the application, and a third which provided ease of use whilst developing.

The first library used was the Google Play App Compatibility support library[9] - this library allows developers to use Android components on older devices that have been introduced in newer implementations of the Android operating system, providing greater coverage and compatibility.

The second library I used was the Apache Commons library, which provided functionality to perform the network operations necessary to upload data to the server - most importantly it allowed for the crafting and delivery of HTTP POST requests.

The third library was the SQLiteAssetHelper library, which provided an extension to the SQLite functionality already included in the Android API. It searches for database files in all the directories that assets would normally be put in, and allowed easy manipulation of databases across different devices.

5.1.2 Component Analysis

In this section I will provide an analysis of each class in the application, along with the functions that are performed within each activity.

Activity: Main Menu

The first activity a user will see upon opening the application - at first glance this activity just stands as a gateway to the rest of the application, but as soon as the application has started up, it retrieves the latest version of the locations database, and then opens that database to build a directed graph, using the DiGraph class. Finally, it kickstarts the background service that pushes data to the backend.

It was important to make sure that this operation in the background ran in a seperate thread to the GUI, or there would have been issues with blocking whilst the operation was carried out (problems that would become more prevalent the larger the database became).

Class: DiGraph

This class is critical to the function of the application - with all of the location based functions in the application using stored in an object of this class. The DiGraph class uses custom made Node and Edge classes to create a directed graph within the application, allowing for the later

implementation of Dijkstra's algorithm. Each Node represents a location within Kings, with each Edge representing a connection between those two locations.

One of the most critical aspects of the class is the global variable that forms its adjacency list, details each Node's adjacent Nodes, and the Edges that connect them. Finding a data structure in which to store these three pieces of information, whilst also making them easily retrievable was difficult - eventually a HashMap within a HashMap was used.

In order to allow for the easy retrieval of nodes using different identifying factors, it was necessary to implement a rough type of custom indexing, which involved taking the Nodes that were in the adjacency list and creating Maps for them for any data type that would be needed to call a Node. An example would be:

```
public static Map<Integer, Node> nodeIDList;
public static Map<String, Node> nodeLocationList;
public static Map<String, Node> nodeMAC;
```

Each map in the above code stores a seperate copy of the list of Nodes, with whichever data is used to find it using the .get() method. For example, using

```
nodeIDList.get(0);
```

would be the same as

```
nodeMAC.get("MAC address of the first Node");
```

The problem with this approach is that when originally adding each Node to the DiGraph, the operation is now much more expensive to perform, and there is exponentially more space used per Node - however the ability to easily retrieve each node offsets this initally expensive operation, and the memory usage is still negligible in terms of the memory available to the device.

Activity: Location Finder

When this activity is opened, a wifi scan is immediately performed, and a loading screen shown until the Activity has received and processed the results. The completion of a wifi scan triggers a WIFI_SCAN_RESULTS_AVAILABLE event which is listened for by the BroadcastReceiver class. When the BroadcastReceiver receives this action, it triggers an onReceive method where the results can be fetched and operations run as a response to this action.

The results are returned in a ¡ScanResults¿ List, which is then ordered using a comparator, according to signal strength (from the best quality to the lowest quality). In the best case, the top result is taken, and its MAC address used to find the Node relevant to that user's location in the DiGraph. Because of the issues with signal variance that have been described throughout this report, this is not always the case. To cover this occurence, the implementation will loop through the results list until it hits the first recognisable MAC address.

```
for(int i = 0 ; i < results.size() ; i++ )
{
    if(!MainActivity.myGraph.getLocFromMac(results.get(i).BSSID)
    .equals("Location not found"))
    {
        locationString = MainActivity.myGraph.getLocFromMac
        (results.get(i).BSSID);
        break;
    }
}
resultString = "You're at: \n \n " + locationString;
```

Finally, the user's location is extrapolated by getting the 'location' variable from the Node.

The original plan for this class involved using the top three access points to get a much more precise estimate of the user's location using trilateration. Unfortunately, because of the variation of signal strengths at any given point (a figure that can even be affected by another person's body), the top three MAC addresses in any given place can change constantly. To get an accurate result would require having populated the database with every possible variation for every area, resulting in an exponential increase in the amount of data that needs to be taken (and therefore Nodes required).

Once this activity has pinpointed your location, the loading screen will be replaced by that location, along with a button to refresh the location (which will start another scan), and a button that will use the current location as the start point in the route planning activity.

An interesting feature provided by the BroadcastReceiver is that it will listen for *any* trigger of the SCAN_RESULTS_AVAILABLE action, meaning that when the system performs a wifi scan seperate from the application, the application will perform the operation defined in it's onReceive anyway - this means that the location will auto-update without having to trigger

multiple scans, ultimately meaning a more economical use of resources.

Activity: Route Planning

This activity provides two drop down lists - each populated with all of the locations that are currently present in the directed graph. The user can select a start point, and an end point and once the 'Find Route' button has been pressed, the activity fetches the relevant nodes for both locations (by using their position in the drop down lists as the node ID, and fetching from the nodeIDList). It then applies Dijkstra's algorithm to the start node to find the shortest distance to all of the nodes in the directed graph, before using the second node as an argument to find a path between the two.

The resulting path is then displayed in the next activity - each step in the path is used to create an 'inflatable' - inflatables allow for the dynamic creation of activities by using the same small section of the layout and filling it with changeable content.

Class: Dijkstra

Dijkstra's algorithm is the chosen shortest path algorithm for this project, and provides all of the functionality behind the route planning activity. The implementation of Dijkstra's algorithm written for this project will take a directed graph object upon instantiation of a 'Dijkstra' object:

```
DiGraph dg = MainActivity.myGraph;
Dijkstra pathFinder = new Dijkstra(MainActivity.myGraph);
```

To run the algorithm, it is passed a source node:

```
pathFinder.execute(dg.getNodeFromID(start.getSelectedItemPosition() + 1));
```

This finds the shortest path for all nodes from that source node, filling out the 'predecessors' map variable with pairs containing the node, then the node preceding it. To retrieve a path, a LinkedList is generated by stepping backwards from a given end node:

```
LinkedList<Node> pathList = pathFinder.getPath(dg.getNodeFromID
        (end.getSelectedItemPosition() + 1));
```

```
public LinkedList<Node> getPath(Node target) {
        LinkedList<Node> path = new LinkedList<Node>();
```

```
        Node step = target;
        // Check if a path exists
        if (predecessors.get(step) == null) {
            return path;
        }
        path.add(step);
        while (predecessors.get(step) != null) {
            step = predecessors.get(step);
            path.add(step);
        }
        // Put it into the correct order
        Collections.reverse(path);
        return path;
    }
```

Service: LocationPush

Services are components within Android applications that can perform operations in the background that don't require a graphical user interface[8] - therefore it serves as the perfect platform from which to routinely push data without requiring user input.

Firstly, it is worth noting that a Service does run in the main thread of it's hosting process - therefore any long running operations should be be run within a new thread so as to avoid blocking on the main.

The first thing the service does is trigger a wifiscan in much the same manner as the Location Finder activity does - the main difference being that when the BroadcastReceiver is triggered the first time, it is then unregistered to prevent the repeated triggering of the operation defined in it's onReceive.

After this first trigger, the Service will create a new thread, and attempt to connect to google - if it cannot connect it will check the status of the internet connection and then create a 'fault' record in an SQLite database, noting the time and any details that can be provided as to where the failure occurred. If it can connect, the Service will POST the location data in JSON format, before checking to see if there are any faults in the aforementioned database - if there are it will POST each fault individually in JSON format[1].

[1] Expensive, no?

Finally, the thread will sleep for 15 minutes - upon waking, it will register the BroadcastReceiver again so that it can listen for the next SCAN_RESULTS_AVAILABLE action that the device provides. This again allows for an economical use of resources whilst running in the background, instead of triggering a wifiscan itself every 15 minutes:

```
public void onReceive(Context c, Intent intent) {
            unregisterReceiver(this);
            new Thread(new Runnable() {
                    ....
SystemClock.sleep(900000);
registerReceiver(scanReceiver,
        new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
}.start();
```

Activity: Settings

This Activity provides very little towards functionality, instead providing utility. Firstly, it provides a warning to the user that the app will collect data at regular intervals, and provides checkboxes that allow the user to opt out of the data collection. During development, another feature was also added to provide more utility to the user - a button that will open a dialog for the user to enter their King's username and password.

This data is then used to create correctly set up wireless profiles for both AccessKings and Eduroam. This idea is meant to provide more functionality to the user from the application, and also helps to make sure that the user doesn't have connection issues due to authentication when in Kings.

Unfortunately, during implementation there was a major issue discovered - the functionality to programatically add a wireless profile for a WPA-Enterprise network was only added in Android version 4.3, and so far is not available for earlier Android versions. An implementation using this would violate non-functional requirement NF6, so the application will make sure that the device it's running on is at the required Android version, else it will not display that functionality - the rest of the application will continue to function as normal.

5.1.3 Difficulties during implementation

Whilst developing the mobile application I encountered the majority of the difficulties for this project - mostly due to having to implement complex methods such as a functioning Dijkstra's algorithm, but mostly trying to develop a method to determine the user's location that was accurate, with as much precision as possible. After extensive testing, it became clear that the originally planned '3 MAC' method, where I would use a combination of trilateration and fuzzy location determination[2] to determine a precise[3] location, however the rate of success for accurately finding the user's location was too low - most of the time the application returning that it didn't recognise the area - this is mostly due to the original implementation.

This original implementation took the top three MAC address and concatenated them, meaning that to find a match the user had to be in the location where that exact combination happens again - however, as covered by Cook et al, radio signals are extremely variable[6], and this had a major impact. An example of this is shown in the Appendix - two subsequent screenshots showing a refresh with a colleague next to me and with that colleague three feet away, showing two completely different results. This was a repeatable occurrence.

Because of the limited amount of time available, it was necessary to establish a working, accurate proof-of-concept before trying to develop a higher level of precision, and so the current implementation is almost guaranteed to accurately discern your location within Kings.

5.2 Backend

5.2.1 Toolkit

Languages Used

To develop the backend I looked into the functionality provided by a few different languages - with very little grounding in web server development it was important to pick a language that wouldn't turn out to lack functionality that I required later on.

Eventually, Ruby was chosen as the language, using the Ruby on Rails web framework to develop the backend. Ruby on Rails allows for rapid prototyping, so I could focus on implementing functionality quickly and effectively. As a proof-of-concept project, this was an important factor in choice[4].

[2] A method for roughly determining location
[3] To within 5-10 feet
[4] Admittedly another was that Ruby on Rails is 'optimised for programmer happiness'

Development Environment

Setting up an optimal development environment for the backend was much tricker than for the Android application, but there were several tools that streamlined the process. Firstly, a Digital Ocean server was rented for the duration of the project - Digital Ocean provide 'ready to deploy' droplets - essentially allowing me to rent a server that came with a version of Ubuntu and any web server stack already installed - after this I just had to familiarise myself with the server and the language itself.

Outside of this server, I used Sublime text editor, along with the SFTP plugin to edit files locally and push them back to the server - this was mostly when I was implementing larger functions - for smaller functions I would use SSH and Vim. Also useful was the Rails console, which provided a shell to test code in before implementing it in the application proper.

Libraries / Frameworks Used

Libraries in Rails are given the name 'Gems' - implementing them in a Ruby on Rails (RoR) server is fairly easy to perform and they can provide some great additional functionality that would be difficult / time-consuming to write from scratch. To this end I used two Gems that aren't automatically packed with RoR.

Devise was the first Gem - Devise provides easily implementable user and session management, allowing for the quick development and continued rapid prototyping of a strong authentication system. To write my own user authenticaton with a Gem like Devise available would have been a waste of time, and almost definitely would not have provided the completeness of security that Devise does.

The other Gem that I used was the Bootstrap Gem. Bootstrap, developed by Twitter, provides a front-end framework for easy deployment of intuitive, good looking sites. It also focuses on making any front end-code written responsive - meaning that it will run effectively on mobile phones. Whilst this wasn't critical, the overall package allowed for rapid-prototyping much easier by alleviating any concerns as to the front-end. As a developer that works mostly with back-end / functional code, this was a highly valuable Gem.

I also used one Javascript API called Highcharts - Highcharts provides an API for charting data, allowing the development of good looking data visualisation views. This was crucial for turning the stored data into information, and whilst it took some work to get Highcharts working well with RoR, it was time well spent and Highcharts helped to develop professional looking charts that would not have been possible had I wrote the function myself.

5.2.2 Component Analysis: API

In this section I will run through each endpoint of the API, showing the data the JSON that it takes in, and the action performed when each endpoint is consumed.

POST: /locations

When a JSON of the form :

```
{"loc" : location, "time" : time-of-record}
```

is POSTed to the above endpoint, the server will create a new location record, with the value of 'location' as it's location, and using 'time' as the time recorded. This is one of the most used endpoints, as it is the endpoint that the mobile application posts to every 15 minutes.

POST: /accesspoints

When a JSON of the form :

```
{ "mac" : mac, "location" : location, "time" : time, "notes" : notes }
```

is POSTed to the above endpoint, the server will create a new access point fault record, using each of the key-value pairs in the JSON post for the relevant field in the database.

POST: /dbs

This endpoint is reserved for the secondary mobile application, designed to allow for the easy sharing of a locations database created within the application. When a file is posted to this address, it is stored on the server, under the path /public/dbs/.

GET: /dbs/get

This endpoint allows the main android application to fetch a created locations database from the server - a GET request will return the database file.

5.2.3 Component Analysis: Site

The frontend of the server presents the data that has been collected through the mobile application to an authorised user - firstly as raw data, but with the option to view the data in graph format.

Login

Before any of the pages on the site can be accessed, the user will need to log in. If they try to access any of the views available from the site wihout having logged in first they will be redirected to a login dialog. This functionality is thanks to the Devise Gem, allowing for session-authentication. This is implemented using one line of code:

```
before_filter :authenticate_user!
```

Which checks to see if the user currently has a session.

Index

The root of the site provides a fragment view of the data that a user could see by clicking on the relevant section of the site - it displays a small portion of the current Access Point Fault list, along with a chart showing total location usage over the past 7 days.

Locations

The locations list shows each record that has been created by a POST from the mobile application, with the details of which location the record was created at, and the timestamp for the record. There are links on each row to either view or delete the record. When the record is viewed, the user is taken to a page showing the usage of that location over the past 7, 14 and 28 days.

Additionally, from the main locations list, the user can click a button at the bottom which will download the entire list in .csv format, allowing the user to load the dataset into their own program and create their own information sets from there.

Access Point Faults

The access point faults list functions in much the same way as the locations list, showing the MAC address of the malfunctioning access point, along with it's location, the timestamp for when the problem was discovered, and any notes pertaining to the issue. The options for each of these records are either 'Show', 'Delete', or 'Edit'. Show will take the user to a page that shows only that record's details. Delete will delete the record, and Edit allows the user to modify the record - allowing for the updating of the Notes field.

5.2.4 Difficulties during implementation

Whilst RoR makes a point of being easy to rapidly deploy with, there were still some issues, not least because of the learning curve of web development, which was added to by having to develop an API alongside the user website.

The biggest issue I faced whilst implementing the backend was using the Highcharts API - whilst it massively sped up the process of creating sleek charts, it was a challenge to pull the correct data from Ruby's Active Record to use with Highcharts. Eventually, a method was created in the model manager for the locations:

```
def self.get_location_count(location)
        summary = Location.where("loc == ? AND time > ?", location,
        (DateTime.now - 7.days)).group(:time).count
        past_week = ((Date.today - 7).. Date.today)

        past_week = past_week.map {|d| d.strftime "%y-%m-%d" }

        summary = Hash[summary.map{ |k, v| [k.strftime("%y-%m-%d"), v]}]

        final_result =Hash[*past_week.map(&:to_s).product([0]).flatten].merge
        Hash[*summary.flatten]

        final_result.to_a
end
```

The issue with the above code is that it only works for one part of the Highcharts API, with different types of graph obviously requiring data output in different manners.

5.3 FOUND

FOUND was never meant to be a part of the original application - rather it became a necessity after a short amount of time spent attempting to gather wireless location data. FOUND is a mobile application designed to build a database that the main application (LOST) can transform into a directed graph. The toolkit used is the same as was used to develop the LOST application.

5.3.1 Activity: Main Function

The application has only one activity, dedicated to it's one function. Because it an application designed for my ease of use, and never actually meant to be seen by the user, it would be appropriate to consider it as a kludge[5] when analysing it in terms of UX, usability and functionality.

The application will take a scan of local wireless points, and create a database record that will be readable as a Node object in the main LOST application. If this is the first Node in the database then it will leave it at that. If it is not then it will also create an Edge connecting the newly created Node to the last created Node (or a user-selected Node). There is also option to create more than one edge from the newly created Node, attaching to to multiple other Nodes in the case of locations being connected in multiple ways.

Finally, there is a function which POSTs the database to the backend, so that it can be retrieved by all versions of the main application with ease.

5.4 Testing

During the development of this application, testing was important after the implementation of each new feature. Unfortunately automating this through unit testing was in almost every case, not possible. Because of the nature of this project requiring actual location information taken from the environment it is to be used in, it was much easier for any testing of functionality, and indeed any regression testing, to be performed by testing on the two development phones, making sure that everything that happened was logged using logcat.

5.4.1 Monkey

Whilst not unit testing itself, one tool I did use was Monkey. Monkey simulates a user-defined number of completely random events against the app to see whether any combination of events would cause a crash. This was useful, and performed at the introduction and completion of any new functions.

Seeding

An interesting point to note is that whilst Monkey will bombard the phone with random events until it causes a bug, it can be difficult to get Monkey to replicate that same sequence of events.

[5] http://dictionary.reference.com/browse/kludge

By specifying the seed for the random event generator, the sequence of events can be repeated - this allowed for an easier analysis of any implemented bugfixes.

Chapter 6

Evaluation

In this chapter I will evaluate each componeont that has been created as a result of this project, both against the requirements laid out in Chapter 3, and critically as a component of the project.

6.1 LOST

All of the requirements originally laid out for the LOST module of the project were successfully achieved - the application will succesfully give the user their location in any mapped location, the successful implementation of Dijkstra's algorithm means that the user *will* be provided with the shortest location to their destination, and the application returns useful data to a backend. The application does this all with currently discovered bugs or crashes, and the implementation does a lot of work in the background to make sure that the entire experience flows well. However there do remain some comments to be made on the final implementation.

Firstly and most importantly is the location determination functionality - whilst the requirement of accurately discerning the user's location has been achieved, the accuracy and precision of the implementation is not ideal. The application will give the user a location 99% of the time, but this may not always be accurate, as touched upon briefly in the Implementation chapter. Such a problem was the signal variation that there were cases where the closest router in a room would come at a lower signal strength than a router in the adjourning corridor, and therefore the user whilst the user would receive a roughly accurate location, it wouldn't be the textitmost accurate location. Obviously this meant that the precision of the application was also thrown off.

In order to combat this, when creating the locations, many nodes would be created for a

single location (to try and cover all possible scenarios) - this caused it's own problems - for starters any duplicate nodes would also be shown in the Route Finder part of the application, as the user needed to be able to navigate from any one of these nodes.

Finally, the user interface, whilst simple to use, is fairly bland - whilst the aim of the project was to provide a proof-of-concept application, the application was designed with a large user element in mind, and this does not show through as well as it could do in the UI.

6.2 Backend

Again, all of the requirements have been satisfied. There is a secured backend that takes information from the mobile application in the form of an API and provides that data as easily accessible, useful information to a user.

Ideally, the backend would display data more thoroughly, and in more ways; whilst an attempt to provide the widest range of data visualisations has been provided through the exporting of available data to a .csv file, it does not completely make up for this lack of functionality being directly available.

6.3 FOUND

Whilst this tool was never designed to be part of the project whole, it provided an incredibly useful function - albeit not perfectly. It allowed for the building of a directed graph in database format whilst moving using only a mobile phone, and was not capable of creating an invalid directed graph - but when testing the location database around the university, it showed that the data taken for each location was not always accurate, and so required multiple nodes to be created for a lot of locations.

This could again be attributed to the signal variance discussed throughout the report, but some fault also lies in the data structure originally designed to store this information.

Chapter 7

Conclusion

7.1 Final Thoughts

When this project started, I found myself considering the problem environment with too much attention paid to the real-world environment, instead of abstracting the problem out to a more logical problem - it is a mindset that I was forced to take during the course of the project and that will be invaluable to me as a developer in future projects.

An important lesson learned was to avoid 'feature-creeping' - during implementation I constantly found myself digressing from the development of a feature to add in a function that wasn't necessarily needed in the first place - whilst some of them have been useful (such as the FOUND application), others could have taken a much lower priority (such as the semi-automated wifi configuration setup functionality in LOST).

As a proof of concept project, I believe that this project has succeeded - the result has been a working system that provides a location based contextually aware environment, without relying on GPS - this proves that location technology can develop beyond the current scale that GPS provides. Overall though, I feel as if each component in the project could provide more functionality than it already does. There is always more that can be done to develop a system, and in the case of this project, this is especially true.

7.2 Future Work

This project has provided some fascinating experience, and there are many avenues to consider for future development.

7.2.1 Location Determination

The current application provides a location, but not necessarily the most accurate. Once modifications have been made to the underlying data structure as described below, the accuracy can be increased by creating locations using multiple MAC addresses - as originally intended - just by collecting the two highest strength MAC addresses per scenario instead of one, the accuracy and precision can be dramatically increased, with each additional MAC address acting to reduce the field around the first that the user could be in.

7.2.2 LOST

Unfortunately, most future work for the LOST application is more aimed at the user experience, and less at providing meaningful contributions to the original problem. This covers additions like providing social integration so that users can provide low-level location information and route-planning information to others. Whilst useful when considering it's application in the context of the environment it is currently being applied in, it does not provide any real significance, and would infact be rather gimmicky.

7.2.3 FOUND

The most interesting piece of future work I would like to pursue would be to redesign the FOUND application outside of it's original parameters as my own utility. With careful attention, and modifications made to the underlying data structure for recording locations, the FOUND application could potentially become a useful development tool for mapping large areas with little expense[1]. The eventual result of this would be easily developable and deployable bespoke contextually aware systems.

To this end, the structure for storing locations would need to be changed - instead making sure that only one node is stored for an area, but with multiple MAC addresses to cover all potential signal variation scenarios. This would also tie in with another major point - the location determination method itself.

[1] Obviously with the provision that there are available wireless access points

Chapter 8

Bibliography

References

[1] Stack overflow - iphone signal strength. http://stackoverflow.com/questions/2959567/iphone-signal-strength.

[2] Stack overflow - measuring signal strength. http://stackoverflow.com/questions/21715690/measuring-signal-strength-from-wifi-to-iphone-ipad.

[3] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer Berlin Heidelberg, 1999.

[4] Matthias Bohmer, Brent Hecht, Johannes Schoning, Antonio Kruger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 47–56, New York, NY, USA, 2011. ACM.

[5] Yongguang Chen and Hisashi Kobayashi. Signal strength based indoor geolocation. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 1, pages 436–439, 2002.

[6] B Cook, G Buckberry, I Scowcroft, J Mitchell, and T Allen. Indoor location using trilateration characteristics. In *Proc. London Communications Symposium*, pages 147–150, 2005.

[7] Google. Android platform version statistics. https://developer.android.com/about/dashboards/index.html?utm_source=suzunone.

[8] Google. Android services. http://developer.android.com/guide/components/services.html.

[9] Google. Android support libraries. http://developer.android.com/tools/support-library/index.html.

[10] Google. Configure access points with google location service. https://support.google.com/nexus/answer/1725632?hl=en.

[11] IDC. Smartphone os market share, q4 2014. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2014.

[12] King's College London. King's college london headcount summary 2012-2013. http://www.kcl.ac.uk/aboutkings/facts/ida/stats/HeadcountSummary2012.pdf, 2012.

[13] Jabok Nielsen. The 90-9-1 rule for participation inequality in social media and online communities. http://www.nngroup.com/articles/participation-inequality/, 2006.

[14] National Research Council (U.S.). Committee on the Future of the Global Positioning System, National Academy of Public Administration, Commission on Engineering, and Technical Systems. *The Global Positioning System: A Shared National Asset.* Romantic Reassessment; 118. National Academies Press, 1995.

[15] Felix Richter. 15 years of google. http://www.statista.com/chart/1502/15-years-of-google/, 2013.

[16] Dario Salvi, Manuel Ottaviano, Ignacio Peinado, and Maria Teresa Arredondo. An architecture for data collection and processing in context-aware applications. In *Adjunct Proceedings of the 3rd European Conference on Ambient Intelligence, AmI09, Salzburg, Austria.*

[17] J. Schiller and A. Voisard. *Location-Based Services.* The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2004.

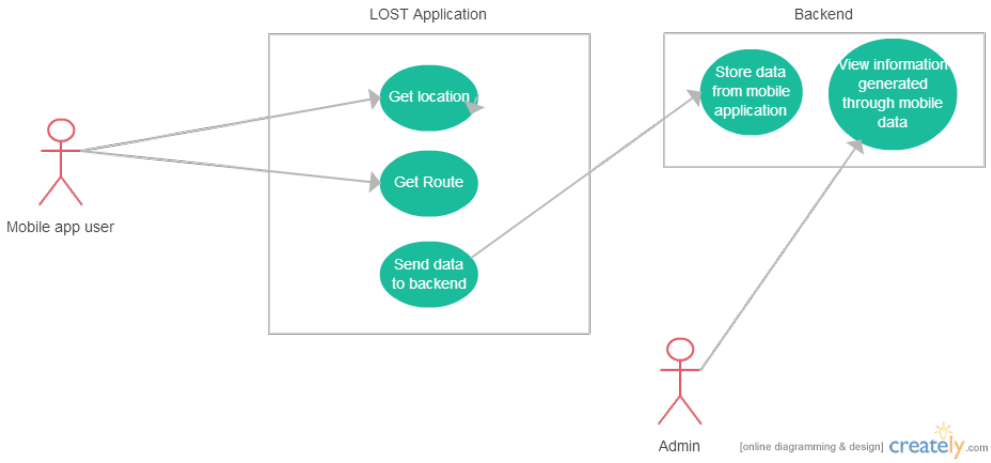
[18] K Thippeswamy, J Hanumanthappa, and DH Manjaiah. A study on contrast and comparison between bellman-ford algorithm and dijkstraâĂŹs algorithm.

[19] Steven J. Vaughan-Nichols. How google–and everyone else–gets wi-fi location data. http://www.zdnet.com/article/how-google-and-everyone-else-gets-wi-fi-location-data/, 2011.

Appendix A

Extra Information

A.1 Use Case Diagram



A.2 Component Diagram