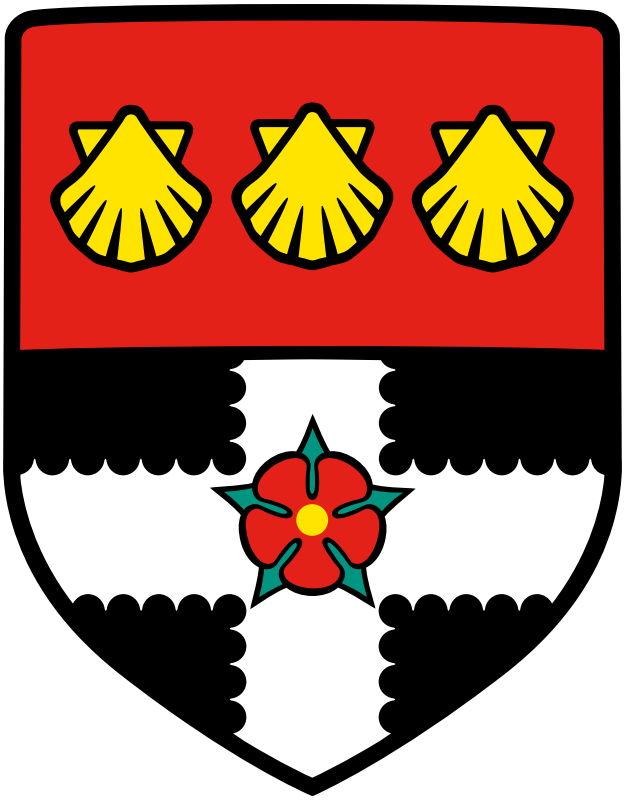
SCHOOL OF MATHEMATICAL, PHYSICAL AND COMPUTATIONAL SCIENCES, UNIVERSITY OF READING

Individual Project – CS3IP16



Find My Android (The use of multiple positioning techniques to ascertain an individual's device location and methods of recovery through remote control)

Oliver M. Bathurst, BSc. Computer Science

Student number: 24025024

Supervisor: Tim Threadgold

Date of submission: 10th April 2018

1. Acknowledgements

I would like to offer my thanks to Tim Threadgold who supervised this project for me and offered valuable suggestions which complemented the project. I would like to thank the University of Reading for accepting me for my BSc in Computer Science.

2. Abstract

The mobility and advancements in the computational power of modern mobile devices has enabled the development of sophisticated location-aware applications for user convenience, navigation, and security purposes. However, common modern-day applications have yet to capture the wide range of location-insightful metrics which are available through the various Android OS APIs. Since smartphones can command a high value, both in price and the value placed on them by the user, nowadays there is a demand for mobile locator applications. In this thesis, a location-aware Android application with a wide array of geolocation methods was developed as a framework for relaying location and other information to the user through different methods of remote control. The aim of the application is to create a complete and comprehensive platform that utilizes all currently available positioning techniques, both short range and long range, with a user-friendly interface. The smartphone application was designed as a location-retrieving, device monitoring tool which will also assist users in performing many different actions to locate, erase and monitor the device remotely though triggers sent via SMS, Cloud Messaging and finally, by email. To enable the execution of actions within the app, the user sets word and phrase triggers through the GUI. Several listeners, namely Cloud Messaging and SMS, were introduced to discreetly and silently handle the processing of messages received, comparing them against the user's triggers. The resulting product from this thesis is an application which performs an array of different actions silently, without user interaction or the app being active, based on messages sent via several techniques and responds remotely with detailed and useful results to the user.

3. Table of Contents

4. Table of Figures

5. Table of Tables

6. Glossary

AP	Access Point
API	Application Programming Interface
BT	Bluetooth
BT LE	Bluetooth Low Energy
Cell-ID	Cell Identification
dBm	Decibels to 1 mill watt
FCM	Firebase Cloud Messaging
GC	Google Cloud
GCM	Google Cloud Messaging
CDMA	Code Division Multiple Access
GPS	Global Positioning System
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HTML	Hyper-Text Markup Language
JRE	Java Runtime Environment
JS	JavaScript
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAC	Location Area Code
LTE	Long Term Evolution

MAC (address)	Media Access Control (Address)
MCC	Mobile Country Code
MNC	Mobile Network Code
OS	Operating System
PHP	Hypertext Processor
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
SMS	Short Messaging Service
SQL	Structured Query Language
SSID	Service Set Identifier
URL	Uniform Resource Locator
WCDMA	Wideband Code Division Multiple Access
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

7. Motivation

Many pre-existing Android applications centred on the geolocation of a device are too simplistic for purpose and remain limited in their methods for the remote transfer of information and locating a lost device indoors. The motivations for this project include the development of numerous methods of indoor geolocation as well as the implementation of the commonly used providers for accurate outdoor location such as GPS. Moreover, this application aims to include support for a wider range of communication channels to the device from an external entity, and to include mobile-phone-specific features such as cellular two-way communication, useful if a device is lost in a remote area without internet connectivity. Existing applications, such as Google's Android Device Manager, lack such features, providing three options as of Feb 2018: lock, erase and locate. This relies exclusively on the lost device having an internet connection and is a pitfall many similar applications suffer from, with few implementing offline solutions such as SMS. A concern to the researcher was the noticeable trend in the options available in these applications, which appear to infer that a missing device is simply lost and not stolen. Subsequently a key focus of this application is remote reconnaissance, using two-way communication channels to relay sensitive information about a stolen device as well as simple one-way trigger messages to erase or alter the device remotely. On top of remote communication, redundancy measures ensure that the instance-based nature of sending messages to the device does not remain the only geolocation method. An online database containing device location data that is updated regularly and automatically by the application mitigates the frustration of a lost connection to the device by providing a "last known location" point of reference for registered users.

8. Introduction

Since its public availability as a utility in the year 2000 [1], GPS has formed the basis of many localization implementations for outdoor applications ranging from car navigation to commercial aviation. Today, GPS capabilities are found within numerous smart devices, such as mobile phones, tablets, and Internet of Things (IoT) devices. A large number of GPS usages within this field originate from applications which rely on the location insights obtained from GPS to deliver certain levels of functionality, such as suggesting local attractions and information that can be directly suggested from the user's location. Whilst this popular means of localization remains dominant, there are many other considerations for alternate means, GPS suffers from the undesirable loss of positional accuracy in indoor environments. This reveals a gap that needs to be filled that has given rise to a new market of devices that can use the location context of a mobile device within a building or room to enhance a particular service to the user. Such devices commonly rely on proximity to other devices such as "smart" appliances within the home.

The aim of this project was to encompass both the conventional methods of localization such as GPS, but also indoor positioning algorithms and methods to facilitate and improve indoor localization that can be applied using technologies an end user is likely to encounter such as Bluetooth and Wi-Fi. The host of these features was decided to be an Android application, accompanied with various other technologies such as an online database and a web interface. The dominant purpose of this application is to permit methods of remote control, surveillance and remote information transmission over different communication channels using a series of unique words and phrases called "triggers". This endeavors to provide a user a large degree of control and information over their misplaced or stolen Android device. Such features extend not only to the fetching of location data and other contextual information, but also a myriad of security-oriented techniques, including the remote erasure of the device.

This report will concern the implementation and design of the Android application, web interface, database, location algorithms, and other features and functionality. The report will present a balanced view on the efficacy and accuracy of the features developed and attempts to provide an insight into the application of these features. The testing of the finished system will be described and reflected with respect to the initial functional and non-functional requirements to assess shortcomings and successes. Finally, a brief discussion will summarize the outcomes of the project and future work that may enhance the system.

9. Problem Articulation, Functional and Non-Functional Requirements

The problem this project is addressing is the localisation of Android devices in indoor environments, as well as the implementation of conventional outdoor positioning systems and providers such as GPS. Secondly, another large problem being explored within this paper is the remote control of a separated device using specific phases and messages. This is a major factor that enables an array of security features that are commonly present in other applications, such as the locking and erasure of all device storage, as well as full control over device functions that may be useful in close range location-fetching, such as the sounding of an alarm. In addition to this, this application aims to provide efficient and fast mechanisms of sensitive and informative data response and relay using two-way communication channels. This provides plain text reports to an external user whom sent the trigger phase over their chosen communication medium, such as SMS and GCM. These reports include content based on the trigger that was intercepted by the device, such that the device can return many different individual data points of reference based on an individual phrase, including but not limited to: a general location report with device information and coordinates, cell tower information, and Wi-Fi and Bluetooth fingerprints. Another problem that is inherent within relevant applications and literature is the lack of offline redundancy, that is, if the device is not currently associated with a data connection, such as Wi-Fi, the application will cease to be fully operable. To counter this, one key project implementation involves the use of SMS to trigger device actions and receive responses, which as a medium enjoys significantly better coverage over Wi-Fi especially in remote areas. This is coupled with an account and online database solution, which aims to frequently store the latest location of the device at intervals, from which a location history can be observed, useful in cases of theft.

The methods detailed overcome the limitations included in many existing location applications, of which many rely on the default location services accompanied with an internet or data connection and provide a limited amount of control over the lost or stolen device. These applications also suffer from a complete lack of indoor localisation utilities and methods, inefficient in scenarios in which satellite-based systems and cell tower triangulation is unable to provide accurate location prediction, common in buildings of many sizes and complexities. On a technical level, this is due to the device necessitating a clear and unobstructed line of sight to a minimum of three satellites to perform the triangulation, or multilateration in the case of N>3 satellites, to pinpoint the devices location. In an indoor environment, the receiver would not have a direct line of sight; the signals that are transmitted from the satellites would then have to pass through the building structure which will cause attenuation, making it harder for the receiver to get a fix.

This project has endeavoured to implement, in addition to multiple outdoor and other close -range strategies, accurate Wi-Fi and Bluetooth based fingerprinting algorithms and positioning logic, making use of the predominantly static nature of these resources that are

present and available to many, if not all, potential users. Usage of these resources aims to enhance the abilities of Android devices to accurately declare indoor positions.

Several stakeholders were identified during development of this project. The first stakeholder is the developer, the researcher, responsible for the builds and maintenance of this Android project throughout the development lifecycle. Secondly, a silent stakeholder was recognised as being the server technicians who are responsible for the upkeep of the university hosting services utilised by the application as a part of the online database subsystem. The final stakeholder is the end user, anyone that owns an Android device and requires the capabilities provided by the application.

The final solution to be assessed is required to fulfil the following objectives:

- The system will provide user location predictions within an outdoor environment
- The system will provide user location predictions within an indoor environment
- The system will provide a trigger customisation utility
- The system will be able to accept a trigger/phrase over at least one medium and perform the associated action based on the trigger
- The system will provide an indoor localisation method
- The system will be able to respond to select triggers over a two-way communication channel
- The system will make use of all pre-existing Android location methods

With these specifications of the desired system established, the following criterion was used to determine the acceptability and validity of the final implementation:

- The application interface will be aesthetic and user-friendly
- The application should support the two fingerprinting methods previously disclosed
- The application should have a user-friendly and instructional training phase for both fingerprinting methods
- The application should include positioning phases to complement the training phases
- Indoor resolution: the application should correctly report the location to a resolution of a standard room dimension
- Indoor accuracy: the application should correctly report the room the device resides within 90% of the time
- Latency: the application will relay a result/perform an action within an acceptable period based on the task being triggered

With these requirements established, the following potential constraints were considered:

- The network speed (link speed) of the device
- The network connectivity status of the device
- The cellular network status and the connection strength
- The processing speed (performance) of the host device
- The application will be available for the Android OS only

In conclusion, a large array of functional and non-functional objectives was identified for this project. Dominantly, accurate Wi-Fi-based and Bluetooth-based positioning systems should be developed for indoor localisation. This system should integrate a fingerprinting algorithm revolving on the signal strength of those technologies to predict the user's location. The Android application itself would be created to serve primarily as a configuration utility for the end user. The app should have a simple and easily interpreted user interface, and it must enable and satisfy the objectives and core functionalities detailed within the requirements analysis. The user-inputting of data during the calibration (training) phase should be instructional and easy to understand. The associated algorithms for these fingerprinting methods must permit a plain-text response including the prediction results. A frequently-updated database utility should be created as an accompaniment for the application. This will mitigate the user's reliance on the transmission of triggers to their Android device.

10. Literature Review

It is necessary to analyze relevant literature to provide an understanding of the different positioning technologies and other relevant technology and guide implementation decisions for the development of the system. In this section, location services/technologies and Android literature will be analyzed to investigate the features available for system development. Advances in internet connectivity, network technology, mobile computation and the rapid expansion in the quantity of mobile personal devices such as smartphones and tablets have incurred the growth of several Location Based Services (LBS). The concept of Location Based Service, shorted to LBS, is that the service provider gets the location information through the wireless communication networks (GSM and CDMA) or other position methods like satellite positioning so as to provide location service for users [2]. At the center of any location based service is the underlying positioning method involved in tracking the current location of the user.

10.1. *Related Positioning Technology*

10.1.1. *GPS (Global Positioning System)*

The most well-known and widely used satellite positioning system is GPS, operated by the United States Department of Defense. In April 1995, the whole system containing 24 operational satellites at a cost of $12 billion was declared operational and that some features would be available for civilian use [3]. The methodology behind the functioning of this positioning system is that a ground-based receiver measures the time a generated code takes to travel from a satellite to the receiver, this is generally very fast, around 0.1 seconds commonly. From this travel time data, the receiver can estimate the distance to the satellite. The satellite now places the receiver somewhere on the surface of an imagined spherical object centered on the satellite with radius also shown below. The distance of the receiver from a second satellite is then measured, narrowing the potential locations of the receiver to an elliptical ring at the intersection of the two spheres as shown below. By including measurements from a third satellite, called trilateration, the potential locations of the receiver can be further reduced to just two possible points as shown. One of these positions is then eliminated, mostly due to the reading being either too far from the Earth's surface. Readings from a fourth satellite out of the possible 24 can then be taken so that the receiver can now be positioned in three dimensions which are latitude, longitude and altitude. The figure below shows a simplified 2-dimensional model. For most of the applications used in the present day the accuracy of standard GPS is sufficient, but there are some applications that require even greater accuracy. The differential GPS was developed to satisfy this requirement. Differential GPS provides a correction for errors that may have occurred in the satellite signal due to slight delays as the signal passes through the ionosphere and troposphere [3]. Although the accuracy of GPS is widely lauded in outdoor positioning, GPS maintains no indoor service, resulting in poor coverage in urban "canyons" [4]. According to market research from the independent analyst firm Berg Insight, the sales of GPS-enabled GSM/WCDMA handsets was 150 million units in 2009 [5]. With most modern smartphones having GPS capabilities, it is an important feature of any proficient location-aware Android application to utilize GPS for outdoor

coverage. It offers unparalleled outdoor accuracy and no directly comparable methods exist presently, making GPS monopolize this area of localization. Therefore, Find My Android was implemented with GPS-provided location-fetching functionality to take full advantage of its effectiveness. This provides the end user with accurate location reports obtained from an outdoor context.

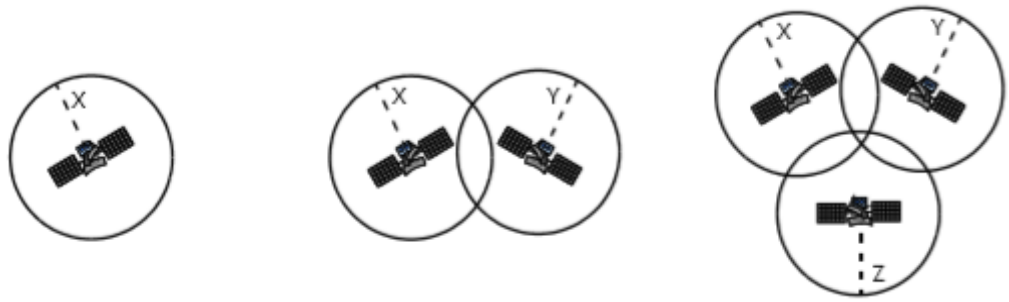
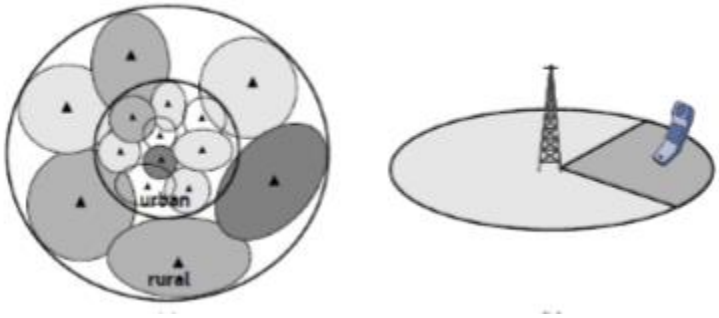


Figure 1. Varying arrangements of satellites providing differing levels of locational accuracy [3]

10.1.2. *Cell ID*

Cell ID [3] is the most basic and oldest form of cellular location and works simply by detecting the base transceiver station (BTS) with which the cell phone is registered to for reception. The mobile phone (Mobile Station) is programmed to always seek a registration to a BTS to maintain cell reception. This is usually the nearest BTS for maximum fidelity, but may also be the BTS of a neighbouring cell tower due to complications owing to terrain irregularities, cell overlap or excessive congestion at the nearest BTS. The size of a cell depends on the terrain of the surrounding area and also on the possible number of users. Hence, the size of cells is much smaller in city centers than in rural areas in which higher ranges and strength are commanded for cell phone reception. The accuracy of a position fix is entirely dependent on the size of the cell due to the location reported through this method being simply the location of the BTS. Consequently, using the BTS location as the only metric, the MS may be anywhere within the boundary or radius of the cell. In a typical scenario the extent of error in urban locations may be approximately 500m, but in rural locations this can extend greatly, up to about 15 km. Each base-station will have multiple antennas, each covering a sector of the cell. So, a BTS with three antennas will produce a cell with three 120 degrees sectors for complete radial coverage. By detecting the antenna with which the MS is registered, the location of the MS can be narrowed down to somewhere within a sector of the cell with the BTS at its apex as shown below. Such techniques require a database of cell tower locations to provide useful coordinates, though coarse grained localization methods [6]. Although network coverage is less accurate than GPS, it provides a location solution for devices that only have cellular capabilities and don't have a GPS receiver to utilize for outdoor localization. It is then important, in order to encompass the widest range of potential devices and thus end users, to provide this functionality to users within the Find My Android application. This method could also prove effective in areas in which GPS coverage is limited and cellular reception is available, however, this is likely to be experienced in

sparse environments in which pinpoint accuracy is paramount. Within the application, CellID is explored accompanying a range of interchangeable location providers such as GPS and Wi-Fi, among indoor localization methods developed as a result of the project.



(a) (b)

Figure 2. (a), an example distribution of cell towers "cells" and (b), a cellular device determined to be within a BTS sector [3]

10.1.3. *Wi-Fi Fingerprinting*

Wi-Fi has been widely used for indoor positioning. Wi-Fi provides local wireless access to a fixed network that is low cost, widely deployed and whose indoor coverage is still rapidly increasing, using existing and prevalent infrastructure for positioning is a very attractive option [7].

As Wi-Fi access points often have fixed locations in indoor environment, received signal strength from these Wi-Fi access points on mobile devices can indicate the device's location by estimating the distance from itself to the access points calculated through an indoor path loss function [8]. Received Signal Strength Indicator (RSSI) provides the measurement of the power of a received signal, as this metric varies over distance, it is frequently used to estimate distance from a given access point (AP). In fingerprinting, one reference point (RP) consists of the available access points (APs) and their corresponding RSS [9]. Several smartphone APIs, such as the ones present with the Android OS, provide access to the measured signal strength from the mobile device to the access point [10]. Wi-Fi fingerprint indoor localization schemes commonly have two key phases. In the training phase Wi-Fi fingerprints (received signal strengths) of surrounding APs are collected from each reference point of a target area. This creates a uniquely identifiable point of reference which can be thought of as a model within a classification context. Obtained RSS data is accumulated in the RSS database (storage medium) to train and generate a RSS model for each reference point within the fingerprint. In the testing phase, the current location is determined by comparing measured RSS values at an instance in time with the RSS models currently saved within the system [11]. The location can then be relayed from the system being located to an external entity. As Wi-Fi access points are both extremely prevalent and predominantly static devices, it is a very promising method of indoor classification and

localization that can be employed by many end users in the proximity of such devices. As such, Find My Android was equipped with a proprietary fingerprinting algorithm, devised for the purposes of the application, to test the effectiveness of the method and to build a pseudo-classification model to accurately classify readings, in the form of obtained RSS, to a resolution of a room, useful indoors.

MAC_AP	Delta(dBm)	RSS_Min(dBm)	RSS_Max(dBm)
00:11:20:06:a0:e0	36	-78	-42
00:11:20:68:b1:b0	31	-71	-40
00:11:20:68:b2:50	41	-81	-40
00:14:7c:af:52:72	40	-91	-51
00:22:3f:19:20:0c	37	-88	-51

Figure 3. A sample data table of APs and associated metrics used for localization [12]

10.2. *Important Android Concepts*

This section covers the various Android classes, APIs and concepts which are explored and implemented within the accompanying Android application to this paper.

10.2.1. *Activities*

An Activity is an application component providing a view users can interact with [13]. An application typically comprises of multiple different activities that display different interfaces. A launcher activity or "main activity", declared with the *android.intent.category.LAUNCHER* category within the manifest, is required for an interactive application and is presented to the user when the application is launched. In this paper, the application opens to a login activity for added security, upon authentication the main activity is launched, which commands control of three fragments, of which each can start their own activities.

As previously mentioned, each activity can start another activity to facilitate different actions and functionality or present different interfaces. Each time a new activity starts the previous activity maintains a "stopped" state and the operating system preserves each activity in a stack. Once a new activity starts, it is pushed onto the stack and takes primary user focus. When the user has finished with the activity and either chooses to return to kill the application, it is popped from the stack and destroyed, causing the previous activity on the stack to resume. For an Activity to be usable, it must be declared in the manifest file to be accessible by the Android operating system.

To minimize the potential for errors and preservation of application consistency, it is imperative to manage the application's functionality with respect to the lifecycle of activities.

An Activity has three states:

- *Resumed*: The activity is in the foreground of the screen and has user focus.

- *Paused*: Another activity is in the foreground (visible to the user) and has focus. Paused activities are still alive but can be killed by the system in low memory situations.
- *Stopped*: The activity is running in the background and is no longer visible by the user. Stopped activities are also alive and can also be killed by the system.

To manage the lifecycle of an activity effectively, necessary call-back methods require implementation. The default call-back methods can be overridden by the programmer to perform appropriate functions when the state of the activity changes. The most relevant call-back methods are:

- *onCreate(),* called when the activity is created for the first time.
- *onResume(),* is called just before the activity returns from a paused state and starts interacting with the user.
- *onPause(),* called when the system is to start resuming another activity.
- *onDestroy(),* called before the activity is destroyed.

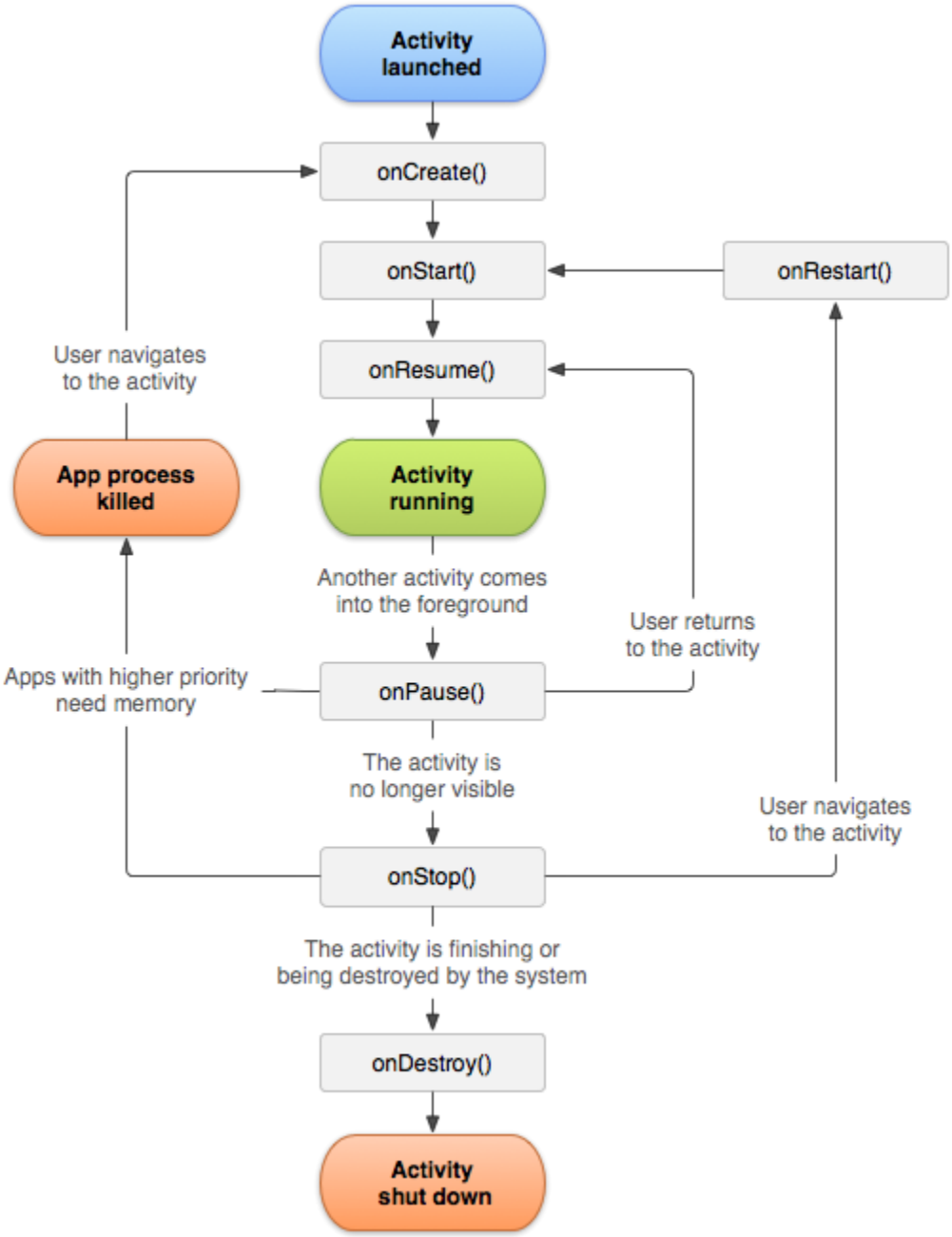


Figure 4. Activity lifecycle flow diagram [13]

10.2.2. *Fragments*

A Fragment is a section of an application's user interface or behaviour that can be placed in an Activity. Interaction with fragments is done through *FragmentManager*, which can be obtained via *Activity.getFragmentManager()* and *Fragment.getFragmentManager()* [13].

The *Fragment* class can be manipulated to achieve an array of aesthetic designs to a user's specification. Fundamentally, a fragment represents an embedded action or interface that is running within a larger Activity. A *Fragment* is used exclusively in conjunction with an associated activity it resides within and cannot be used as a standalone UI entity. Although Fragments define their own lifecycle, that lifecycle is intrinsically determinant on its activity, that is, if the activity is stopped, no fragments inside of it can be re/started; when the activity is destroyed by the system or user, all embedded fragments will be destroyed.

This application utilizes the abilities of fragments to be added, removed, and replaced within an activity in response to user interaction at run-time, such that navigation buttons can be used to replace the content displayed to the user without the requirement of switching activities.

Every set of one or more changes that are committed to the base activity are called transactions and are performed using APIs method calls from *FragmentTransaction*. Like activities, each *Fragment* has a similar underlying lifecycle and stack, each transaction can be programmatically saved to a back stack, of which is managed by the activity, allowing the user to reverse backward through the stack and fragment changes.

All the desired changes to perform in a transaction are collated using method calls such as *add(), remove(),* and *replace()*. The pending changes are then applied to the activity via a *commit()* call.

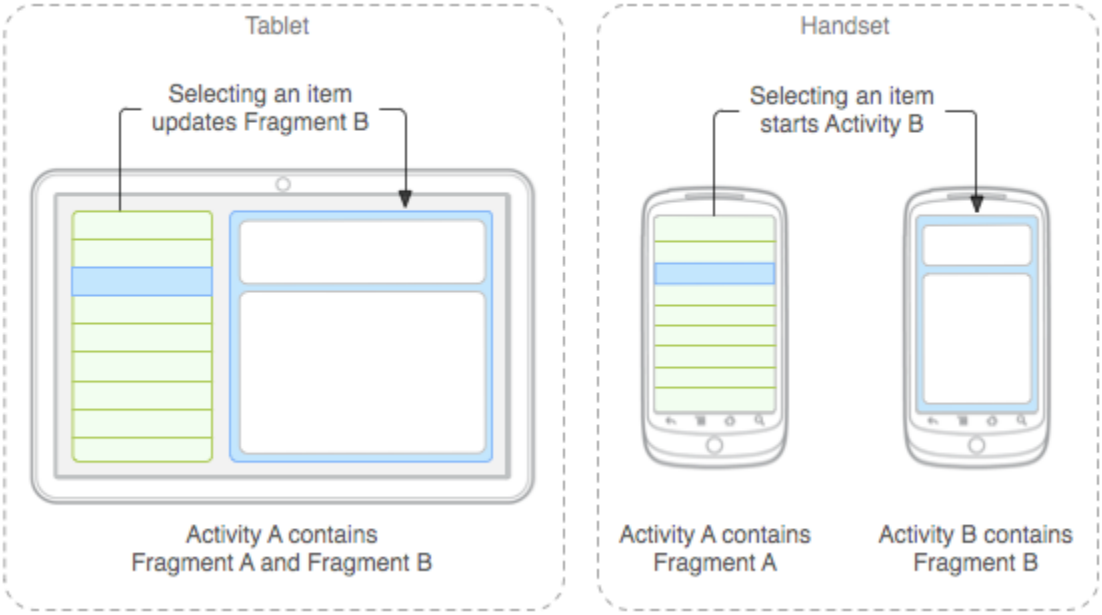


Figure 5. Example usage of fragments [14]

10.2.3. *The Android Manifest (AndroidManifest.xml)*

Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest file provides essential information about the application to the Android system, which the system must have before it can run any of the app's code [15]. The XML document contains many elements required for proper compilation, such as <activity> for all activity declarations, <*intent-filter*>, <*action*>, <*category*>, and <*data*> for BroadcastReceiver declarations. The <*permission*> tag indicates a required permission for the application, API calls which attempt to perform actions without the required permissions may fail, and it's the duty of the programmer to check for these permissions before executing certain actions at run-time. Additional elements include <*receiver*> and <*service*> for declaring and indicating a BroadcastReceiver and a Service respectively.

10.2.4. *Intents*

An intent is an abstract description of an operation to be performed. It can be used with *startActivity* to launch an Activity, *broadcastIntent* to send it to any interested BroadcastReceiver components, and *startService(Intent)* or *bindService(Intent, ServiceConnection, int)* to communicate with a background Service [16]. An Intent conventionally exists in two parts:

- Action -- The general action to be performed, such as *ACTION_VIEW, ACTION_EDIT*, *ACTION_MAIN*, etc.
- Data -- The data to operate on, such as a person record in the contacts database, expressed as an Uri.

Explored in this paper, Intents can also be supplied with addition information within an Intent object; this is called “extras”, which applies additional information to the Intent in the form of a Bundle object. This is utilized to provide extended information to the recipient of the Intent. For example, if an Intent’s action is to send an e-mail message, the Intent could also include extra complementary pieces of data to supply a subject, body, etc. Intents that are broadcasted by the OS are received by “interested” BroadcastReceivers, receivers that maintain one or more matching attributes of the Intent inside their registered *IntentFilter*, such as the same category or data of the Intent. Apart from standard actions, applications can also define their own, customizing the Intent’s action such that they are only meaningful to *BroadcastReceiver* classes of the same application and perform application-specific processes when received by the application itself, of which the author chooses.

10.2.5. *IntentFilters*

IntentFilters are structured descriptions of Intent values to be matched. An *IntentFilter* can match against actions, categories, and data (either via its type, scheme, and/or path) in an Intent. It also includes a "priority" value which is used to order multiple matching filters. *IntentFilter* elements are often created in XML as part of an application's *AndroidManifest.xml* file, using *intent-filter* tags [17]. The purpose of intent filters is to customize and define the type of Intents that are to be passed to certain *BroadcastReceivers*. The Android OS uses the priority attribute of an *IntentFilter* as a factor in determining which application/receiver should receive the matching Intent first. Priority is a singular integer value, ranged between -1000 inclusively and 1000 exclusively *(-1000≤ priority <1000*), with the highest priority being 999. This priority value is used within the application in this paper to intercept message-related Intent broadcasts and to consolidate their contents for triggers before they are accessed by other applications. This enhances the speed at which the application can respond to meaningful events such as triggers and turnaround time for relaying information.

10.2.6. *BroadcastReceivers*

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to *sendBroadcast()* or *sendOrderedBroadcast()*. *BroadcastReceivers* that have registered Intents within their IntentFilter in the manifest will be recipients of such Intents. For example, if a BroadcastReceiver is registered in the manifest with the IntentFilter:

<intent-filter>
<action android:name="android.intent.action.BATTERY_CHANGED" />
</intent-filter>

This filter will cause the receiver to intercept the above intent when broadcast by the operating system, such as if the level of device charge drops or rises. Receivers are denoted within the manifest via a *<receiver>* element along with the name/location of the BroadcastReceiver class. The focus on Broadcasters within the scope of this paper centers on their ability of receive or "intercept" communication-related intents such as those associated with the receipt of SMS and GC messages. Their usage allows the background receipt of trigger messages, execution of the desired actions, and sending of plain text responses, all without the application needing to be active or the user of the phone being knowledgeable of the operations.

10.2.7. *AsyncTasks*

AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers [18]. AsyncTask is necessitated for networking as performing a task of this type on the main thread of an application will cause a run-time exception to be thrown. This class is used extensively throughout this project to perform background networking tasks, such as contacting online resources or updating the user's entry on the database. The three generic types used by asynchronous execution are:

1. Params, the type of the parameters sent to the task upon execution.

2. Progress, the type of the progress units published during the background ` computation.

3. Result, the type of the result of the background computation. [18]

These types are used in order when inheriting the AsyncTask class for asynchronous execution such that a class which inherits the *AsyncTask* class with three string parameters:

AsyncTask<String, String, String> { ... }

Uses the String data type for the execution parameters, progress units, and the result type.

10.2.8. *SharedPreferences*

The *SharedPreferences* class available for Android provides a data storage framework that enables the retrieving and saving of persistent key-value pairs of primitive data types. *SharedPreferences* can save any primitive data the user requires persistency for: Booleans, floats, Integers, and Strings [19]. This data will persist across user sessions of the application, and if the application is closed or killed. A static *SharedPreferences* object is created via a call to *getDefaultSharedPreferences(), e.g.*

SharedPreferences shared = getDefaultSharedPreferences(Context context);

This object holds all key-value pairs associated with the application for read access.

For example, if the user wants to read a key's value from *SharedPreferences*, the method call demands the key name and a default value as arguments, such that if the key does not exist, the user-provided default value (of the same datatype as the primitive type being returned) will be returned, allowing useful decisions to be made by the programmer.

(SharedPreferencesObject).getString("key name", "default value");

For example, to read a stored String of key "*str*" and assign to a string class member *s*:

String s = shared.getString("str", null);

The above method call will attempt to find and return the string value associated with that key and it is subsequently assigned to the string object *s*, otherwise the method's default value null will be returned and assigned, a useful distinction if the user wants to determine a key's existence as a decision-making exercise. It can then be usefully asserted after the method call that if the string object has a null pointer, the key's value does not exist;

if(s == null) //key does not exist

Other method calls to return various datatypes include *getFloat()* for floating-point data and *getBoolean()* for simple true/false values.

To insert data into SharedPreferences, a similar syntax is used, however a call to *edit()* performed upon the *SharedPreference* object is used to create an editor.

SharedPreferences.Editor sharedEditor = getDefaultSharedPreferences(Context context).edit();

Values are added with methods such as *putBoolean()* or *putString()*. The first parameter being the key, and the second the value.

sharedEditor.putString("stringkey", "test");

To commit the key-value pair to storage, the Android API provides two method calls that can be performed on the editor object to complete all pending "puts" and write to storage, *commit()* and *apply(),* that produce two different use cases. *Commit()* returns a Boolean value determinant on whether the operation was successfully completed, true or false, if the programmer does not require a return value *apply()* is then preferable, as it does not perform synchronous writing alike *commit()* that blocks the main thread, and completes asynchronously in the background.

sharedEditor.apply();

10.2.9. *Serialization and Deserialization of Java Objects*

In the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment) [20]. The resulting serialised array of bits can be reread according to, and with knowledge of, the original serialization format. It can then be used to create an identical representation of the original object, such a process is called deserialization. Serialization, and subsequently deserialization, are methods used within this paper to serialise and deserialize irregular data structures into, and out of, primitive data type objects, for persistent storage within Android's *SharedPreferences* This is preferable to a file IO implementation, both in terms of space and code efficiency, storing a serialised data structure under a single key. For this, Google's Gson JSON library was used. Gson is a Java serialization/deserialization library to convert Java Objects into JSON and back [21], within Android, JSON objects can be treated as the primitive data type "String" and therefore the serialised object can be placed into the preferences for the application via the aforementioned *putString() SharedPreferences* API call.

10.3. *Android Location Providers*

The Android OS has three key location providers, each presenting different nuances and methods of obtaining a location fix. Each device is commonly equipped with three providers: GPS, network and passive, although not all devices possess a GPS chip, eliminating the former method. Due to the procedures of these providers, each can be characterised in terms of accuracy and coverage, whereas one may be suited to urban environments, another may be the only viable solution in remote locations. It is then recommended to choose the provider that has the highest likelihood of procuring the most accurate location fix of the device in the environment it is likely to reside within. This chapter presents the methods of the three providers for geo location, as well as the latency, coverage, and accuracy of each.

10.3.1. *GPS*

GPS (Global Positioning System) is defined as a constellation of at least 24 satellites, the navigation payloads which produce the GPS signals, ground stations, data links, and associated command and control facilities which are operated and maintained by the Department of Defense (DoD) in the United States [22]. GPS permits a variety of land, sea, and airborne users to deduce their three-dimensional position, velocity, and time, at any time of day in all-weather scenarios, anywhere in the world. The GPS receiver contained within select Android devices, prevalent among mobiles, can calculate this time difference between the broadcast and the time of signal arrival. When the device understands its distance from a minimum of four different satellite signals, it can calculate a geographical position.

10.3.1.1. *Latency*

The GPS receiver exists as an independent component in the Android platform with the singular purpose of inferring location. A GPS location fix requires at least four, but commonly seven satellites to obtain a location fix, as well as having a relatively unobstructed line of sight to those satellites. Such prerequisites contribute a large amount of time delay between requesting and receiving GPS information, consequently, this method consumes a lot of power if used on a regular basis.

10.3.1.2. *Coverage*

GPS coverage is dominant over all available Android sources; however, this only extends to outdoor scenarios where satellite visibility is achieved. Due to this limitation, GPS is infamously inaccurate in indoor locations, in which this method may provide some location data in some scenarios if there is a degree of visibility of satellites, but with notably poor performance. The obstruction factor may propagate even to outdoor environments, GPS may suffer from poor performance in built-up areas, aggravated in cities due to objects such as buildings, trees and other obstacles can obscure a satellites line of sight, decreasing its viability. Therefore, GPS coverage can be limited compared to other sources, but remains a reliable solution for outdoor positioning.

10.3.1.3. *Accuracy*

In outdoor scenarios, with good satellite coverage and line-of-sight, GPS is the recommended positioning system, providing the most accuracy over all other sources. In optimal conditions, including those conditions previously discussed in addition to favourable weather, GPS can achieve a precision higher than three metres, but in general, the margin of error ranges between three and ten meters. In bad conditions with lower visibility of the satellites, the GPS accuracy degrades to between thirty and one hundred meters. GPS also provides the capabilities to deduce device altitude, useful in areas of varying elevation, providing more location context.

10.3.2. *Network*

Android's "*NETWORK*" provider utilises a specialised lookup service, maintained by Google, determining a device's location based on the availability of cell tower and Wi-Fi access points sent via a query. Results are retrieved by means of a network lookup [23]. This method of geolocation creates and sends a location request to Google servers containing a list of currently obtainable MAC (Media-Access-Control) addresses that are visible to the Android device at that instant. The location server then performs a lookup and comparison operation, cross-referencing the given list with a separate list of known MAC addresses of the device, and identifies associated locations. The server then uses these locations to multi-laterate the approximate location of the user. However, unlike GPS, this method relies on the participation of Android users sharing their location and associated, insightful metadata.

10.3.2.1. *Latency*

As this method does not rely on GPS chip activity, location locks tend to be fast, requiring no additional power. Therefore, location reports, over the internet or otherwise, will suffer most or all latency as a result of the connection strength of the Android device.

10.3.2.2. *Coverage*

Although the coverage of the network provider is less extensive than GPS, it provides a more versatile solution for built up areas in which GPS coverage is limited. Because this method relies on the device being in range of a wireless AP, and thus maintaining a signal to receive location reports, it can produce accurate performance in both outdoor and indoor scenarios, but at the expense of some accuracy. This solution is then a viable option for areas of large AP density, such as public Wi-Fi within city centres.

10.3.2.3. *Accuracy*

Indoors, the network provider can achieve much better positional accuracy than other providers, commonly ranging between three and ten metres. Outdoors, the accuracy degrades significantly to around 25-150m, depending largely on the availability of access points. In places of little to no network availability, the GPS provider is more suitable.

10.3.3. *Passive*

Passive is unalike the other providers and does not in itself attain a location fix. This provider can be used to passively, not actively, receive location updates when other applications and services request them without the provider itself requesting the locations. This provider also returns location data generated by other Android providers. If the Android device does not have GPS capabilities or the locational accuracy is set to ‘Low Accuracy’ within settings, this provider will only return coarse, inaccurate location fixes. As this method is simply returning the last known location produced by other applications, its power consumption is negligible, and the location fix immediate. However, due to the inherent methodology applied, it may not return the current location of the device, and the age of the location fix may be old and irrelevant.

10.3.3.1. *Latency*

Due to the provider simply returning the last known location of the device produced by other providers and applications, it is an immediate process.

10.3.3.2. *Coverage*

The coverage level maintained by passively obtaining a location is determined by the underlying providers used by other applications. It has the potential to utilise all Android providers for maximised coverage but is reliant on the one provider most recently used, so the coverage is

limited to that of one provider at that instance in time. Such usage of other sources guarantees that the coverage level is never consistent (unless all services and applications use the same provider) and if one provider is used exclusively by the system, the coverage of passive is therefore equal to that of the provider.

10.3.3.3. *Accuracy*

The accuracy of passive location fetching is, by definition, directly proportional to the accuracy of the provider used by the last application or service of which the passive provider obtains its location data from. That is, if the last known location was declared by GPS, the passive provider using that fix will subsequently inherit the accuracy of that provider. Thus, the accuracy is extremely varied up to 1 mile. However, if the device were positioned in areas of large AP density or GPS coverage, this would ensure other applications that request location updates using GPS and Wi-Fi providers can do so, increasing the accuracy of this provider when used.

11. Background

11.1. *Why Android?*

The Android OS was announced over a decade ago, on November 5, 2007. Within this short period of time Android has progressed extensively throughout its various flavours and early prototypes, to be the most prolific mobile OS in the smartphone market. Android now commands a reported 81.7% of the smartphone market, making the Android platform and application development an attractive and lucrative proposition. The success of a proposed application should always be quantified with respect towards reachability, the largeness of the user base; this is also displayed in the apparent increased targeting of the Windows platform within the desktop OS market. The proclivity of developers developing on the Android platform resulted in an enormous incline in the number of apps available from it's infancy in December 2009 (30,000) to December 2017 (3.5m) [24].

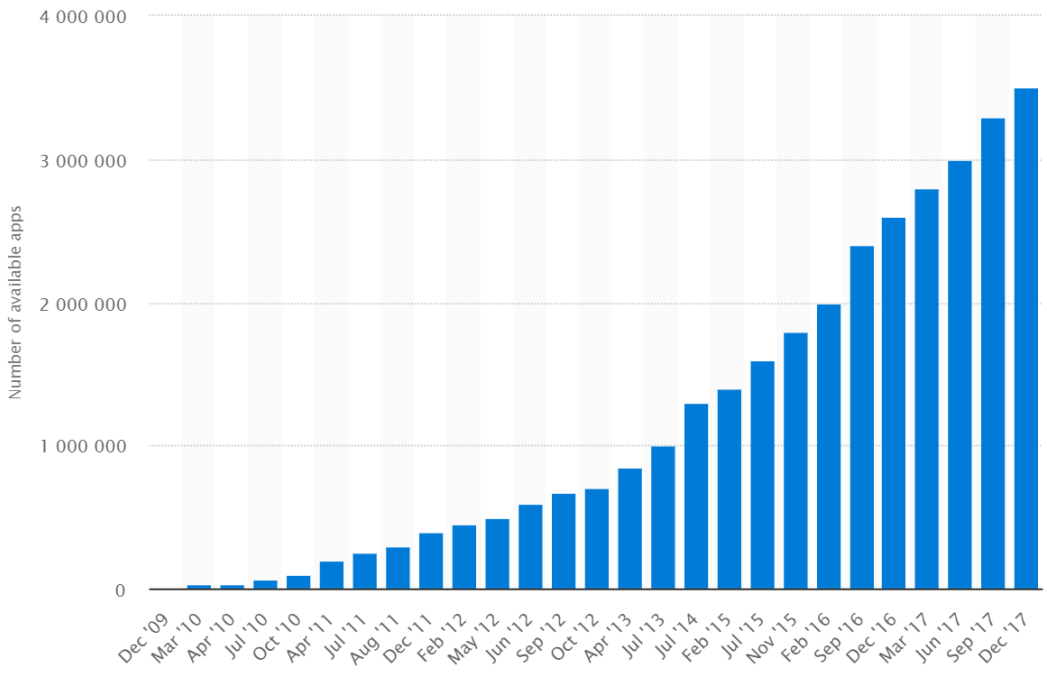


Figure 6. Number of available applications in the Google Play Store from December 2009 to December 2017 [24]

A global leader in year-to-year growth, Android still maintains its lead and potential for prospective app developers, with great projections for future increases in sales and downloads. Table X shows the data collected in Q4 2015 compared with 2016. Despite a small one per cent

increase in Android's overall market share, other mobile OSs, such as Apple's iOS, Android's main competitor in the mobile space, pale in comparison, with a meagre 0.2% increase over the years reported. Android's market share dominance also translates well with units sold, collecting 352,669.9 out of the reported 431,539.3 units sold within Q4 2016.

Operating System	**4Q16 Units**	**4Q16 Market Share (%)**	**4Q15 Units**	**4Q15 Market Share (%)**
Android	352,669.9	81.7	325,394.4	80.7
Ios	77,038.9	17.9	71,525.9	17.7
Windows	1,092.2	0.3	4,395.0	1.1
BlackBerry	207.9	0.0	906.9	0.2
Other OS	530.4	0.1	887.3	0.2
Total	**431,539.3**	**100.0**	**403,109.4**	**100.0**

Table 1. Worldwide Smartphone Sales to End Users by Operating System in 4Q16 (Thousands of Units) [25]

Due to the wide variety of Android devices available with differing hardware specifications, many devices employ modified versions of an Android flavour. As of 2018, the Android platform exists in eight versions or flavours, with newer entrants enjoying more frequent updates. Spanning 17 APIs, newer versions understandably exist with more features and enhanced level of available API methods for developer use. However, these recent APIs are not necessarily cohesive with their predecessors. Such distinctions present a balancing act, shaping a developer's decision as to what API level they must use to implement their vital application features, but also whether usage of that API level will exclude a large proportion of users whose Android versions have a lower API level. Fortunately, as of Feb. 5th, 2018, a large 82.3% of reported Android devices belong to later the group of Android versions (Version => 5.0), with the largest distribution of devices running the Marshmallow version 6.0 operating system. [26]

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%

Table 2. The distribution of Android versions/APIs over all reported devices (Feb 5th, 2018) [26]

The recently released Android 8.0 and 8.1 (Oreo) hardly visible on the table with a combined 1.1%, it highlights the wide distribution of devices. With 99.3% of Android devices

now running 2.x versions or newer. These statistics are indicative of devices which have made a connection to the Android Market. Therefore, it is not entirely representative; however, trends can be inferred in relation to previously harvested data, showing the deployment of the successive Android updates over time. As Android is a software stack and thus can be installed on any device that satisfies a minimal set of requirements and is meant to support a variety of resolutions or even screen orientations, guarantees can be made such that any vendor's device installed with an Android version (and subsequently API level) includes all API-specific method calls and additional functionality. The open-source nature of Android has attracted attention and business from numerous vendors and large companies that build and maintain their own flavours, such as Samsung, Sony and Huawei. This ensures that updates are released semi-regularly over a single vendor system such as Apple, ensuring that Android devices receive important updates frequently, and hence application developers can utilise new capabilities faster and on a wider set of devices. Oppositely, the propriety existence of Apple devices such as the famous iPhone constricts the availability of the operating system to a single vendor, hence the consistency of updates over all devices is ultimately decided, and the responsibility of, a single entity. However, the relatively limited number of iOS devices, representative of units sold as previously discussed, reflects a longer lifecycle of maintenance and upkeep.

Due to the distributions previously noted in Table x3, the Android application for this project was developed to concede with the widest distribution and as such encompassed a minimum API/SDK level of 14, such that its projected availability reached 99.3% of Android devices at the date of completion. However, although the application is functional at this API level, it is not conducive to the full functionality of the application's components. Hence, certain features that require later APIs, such as cell tower identification, require version checks at runtime before the user is permitted access. For full functionality, the calls required for components were carefully selected to demand the lowest builds of Android. Consequently, Jelly Bean (4.3) exists as the latest API requested within the application and represents the earliest build of Android needed for all locational features, still targeting a comprehensive 94.3% of the Android market [26].

11.2. *Project Technologies*

11.2.1. *Java*

Java was developed by a team lead by James Gosling at Sun Microsystems, a company later purchased by Oracle [27]. Java is a high performance, concurrent, architecture-neutral, and object-oriented language. A C-language derivative, its syntax rules look much alike C's. Structurally, Java software is typically segmented into packages. A package is Java's namespace, containing different elements. Within packages are the classes, each class may contain any number of methods, variables, constants, and more. Java has a simpler object model compared to C and C++, with its intermediate byte code which can be run on any Java Virtual Machine (JVM), paying no heed to the architecture of the system it is run on. This portability ("write once, run anywhere") saw Java rise from being a language designed for embedded chips and web applications to a general-purpose language used to develop standalone software across many

different platforms, such as servers, desktop computers, and mobile devices. Within this project Java is used exclusively for the Android application.

11.2.2. *PHP*

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML [28]. The distinguishing factor between PHP and a client-side scripting language such as JavaScript is that the code is executed on the PHP server, generating HTML code which is then sent to the client resolving the script. The client would then receive the output/results of running that script but would not be knowledgeable of the code used to produce said results. Careful configuration of a web server can cause exclusive processing of HTML files with PHP so there is no way for a client to determine any underlying implementation. This encapsulation makes PHP an attractive scripting language for database authentication and interaction, with the user resolving the script URL incapable of determining the underlying credentials used. The PHP scripts used within the project involve the creation of user accounts on the database, as well and the fetching and updating of user information.

11.2.3. *SQL*

Structured Query Language (SQL) is a commonly-used language for manipulating relation and standard databases, with operations including the storing and retrieving of data. SQL offers two key language advantages: it introduced the concept of accessing and manipulating multiple records with a singular command; and secondly, it eliminates the prerequisite of specifying how to access a matching record such as with an index. Plain text SQL queries are used to perform this manipulation. These queries can be executed by a variety of implementations, such as PHP scripts. The standard SQL commands to interact with relational databases are `CREATE`, to create a new table, view of a table, or any other object in the database e.g. `CREATE DATABASE DatabaseName;`. `SELECT`, which retrieves records from (N>=1) tables e.g. `SELECT column1, FROM table_name;`. `INSERT`, which creates a new record e.g. `INSERT INTO TABLE_NAME (column1) VALUES (value1);`. `UPDATE`, which modifies a record within the database e.g. `UPDATE table_name SET column1 = value1 WHERE condition;`. `DELETE`, which simply deletes a record e.g. `DELETE FROM table_name WHERE condition;`. Finally, `DROP`, used to delete an entire table, a view of a table or other objects in the database e.g. `DROP TABLE table_name;`. The project's SQL queries are performed on the application's associated user database via hosted PHP scripts.

11.2.4. *HTML*

"HTML is an acronym for HyperText Markup Language. HTML documents, the foundation of all content appearing on the World Wide Web" [29].HTML documents conventionally consist of two related pieces: the information to be displayed on a web page, and the specific instructions that tells the interpreter of the document how to display the information. The instructions are called "mark-up", HTML in itself is a document alike XML and requires an application to

interpret the instructions contained within and to display the content. HTML documents are most commonly parsed by a web application, such as a web browser, and should be displayed similarly or identically over all applications but is also dependent on the web implementation. A basic HTML document comprises of a minimum of four elements that detail behaviour and appearance, defined by a start and end tag in the format <…> and </…> respectively. The required elements of any document are: `<html>`, which encompasses all other elements and defines document bounds, `<head>`, which acts as a container for document metadata `<title>`, which describes the document via a title and `<body>`, which defines the main visible content of the document. HTML is the primary language of the project's web interface.

11.2.5. *JSON*

JSON or JavaScript Object Notation is a text-based data interchange format for exchanging data between platforms [30]. Derived from the JavaScript scripting language, many programming and scripting languages have libraries, APIs, and implementations to read, parse, and write JSON-formatted data. Requiring much less formatting, JSON offers a good alternative to XML, producing a very readable and lightweight syntax, preferable when communicating data over an internet connection. A JSON object is represented as a key-value data format that is typically encased within curly braces. Although JSON normally exists in a .json file, they can also exist as JSON objects or strings within the context of a program, creation and manipulation of such JSON objects/strings are supported by various APIs, such as those available for the Java programming language. The key-value pairs within JSON are delimited by a colon as in "key : value". Each intermediate non-final key-value pair is separated by a comma, such that the innards of a JSON are in the format: "key : value,", with the last declared JSON pair identified by its lack of a comma i.e. "key : value". Whilst a key is restricted to a text format, JSON values, found to the right of the colon, take form as any of six data types: strings, numbers, objects, arrays. Booleans, or the empty value "null". JSON is used as the primary serialisation format for Java objects within this project, including cloud messages.

11.2.6. *XML*

"XML, the Extensible Markup Language, is a W3C-endorsed standard for document mark up. It defines a generic syntax used to mark up data with simple, human-readable tags." [31] XML is a flexible data storage medium that can be customised for any medium such as web pages, object serialisation and vector graphics. XML documents alone do not provide any functionality, requiring a software implementation to parse and manipulate the data stored within the document. XML documents typically contain three components; tags, elements and attributes. A tag is a construct initialised with a '<' character and terminated via a '>' character, three types of these constructs exist; a start tag `<tag>`, an end tag `</section>`, and an empty element tag `<empty />`. Secondly, an element is created by the introduction of a start and accompanying end tag, the text content between the tags represents the element's contents e.g. `<element> text content <element>`. Finally, an attribute is a construct with a name and relevant value, existing as a pair in a start or empty-element tag e.g. `<img src="pic.jpg" alt="Nice picture" />`.

11.3. *Initial Designs*

The initial design of the Android application was centred on aesthetics and ease of use, but also being minimalist. The core functionality of the application is as a configuration utility in which the user modifies settings that affect how the system framework responds to trigger phases and messages. For user-friendliness, a bottom navigation drawer was seen to the best choice for navigation from observing other applications; this would be used to switch between different fragment settings/option screens that the user can manipulate to their liking. The wireframe devised shows three navigation options which changes the fragment displayed to the user.

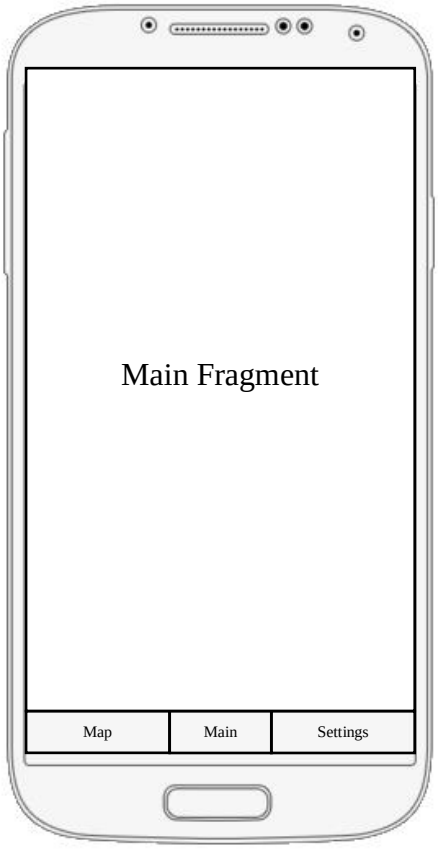
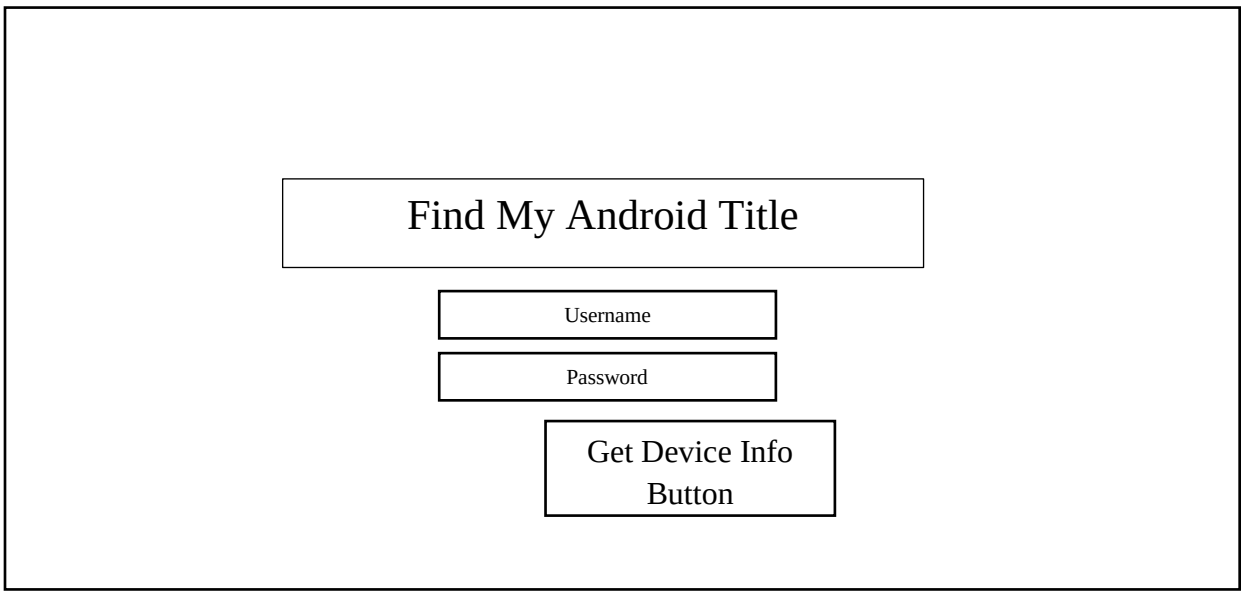


Figure 7. Initial wireframe design of the Android GUI, showing the main fragment and a bottom navigation drawer

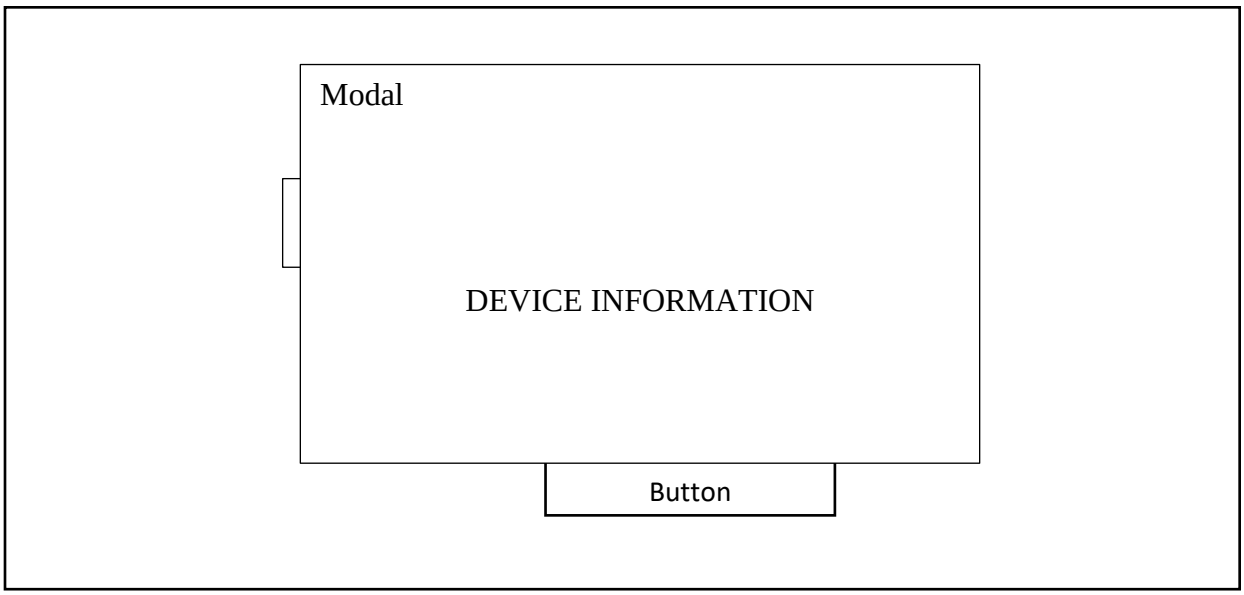
The first option would be a ‘Map’ fragment, this would show the users’ current location along with other metadata, this should be used to test different location providers and let the user

visualise the accuracy of their chosen provider. The second would be the main configuration fragment, this would contain all options a user may wish to alter, including all the triggers available and backup information such as a secondary phone number and email address. This fragment would also be used to launch all associated activities such as those used for calibration and fingerprinting. Lastly, there would be a simple informative screen as the rightmost fragment option on the drawer; this would include information about the phone's cellular connection, Wi-Fi access point, last known location, and other device settings. It may also include information about the OS's location settings, and offer options to change these from within the application.

The web interface, which was initially conceived as an entry point for accessing location data from the database, also had a minimal design, presenting the user with fields for obtaining this data using username and password combinations. This authentication capability would allow the data from the database to be presented to the user via popups of "modals", shown below.



(a)

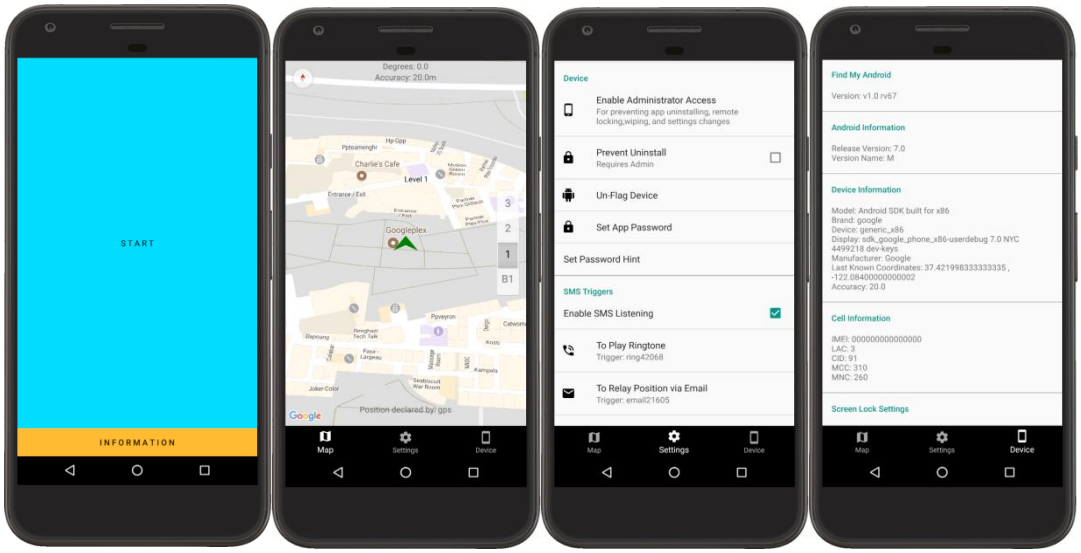


(b)

Figure 8. The overall web interface (a) with login options and (b) showing an example dialog with device information

11.4. *Final Designs*

The final design for the Android GUI did not differ from the initial. The application retained the different fragments that are presented upon button presses of the bottom navigation bar buttons. Additionally, a login screen was implemented and presented to the user upon application launch to allow for password protection, preventing unauthorised users from altering the application's settings. Other than this, the application design did not deviate, retaining the original fragments and their purpose within the system.



(a) (b) (c) (d)

Figure 9. Final Android GUI fragments, "login" (a), "map" (b), "settings" (c) and "device" (d) respectively

Oppositely, the web interface underwent significant changes. This was mainly to incorporate the functionality necessary to send messages from the web interface itself via Google Cloud Messaging. Additionally, fields were added to customise the contents of these messages before transmission as well as a response box to hold any information returned by the Android device resulting from the initial GCM trigger sent from the webpage (GCM relay). Other features were ultimately chosen for end user convenience, such as a manual coordinate lookup with an embedded map that can be used with received coordinates.

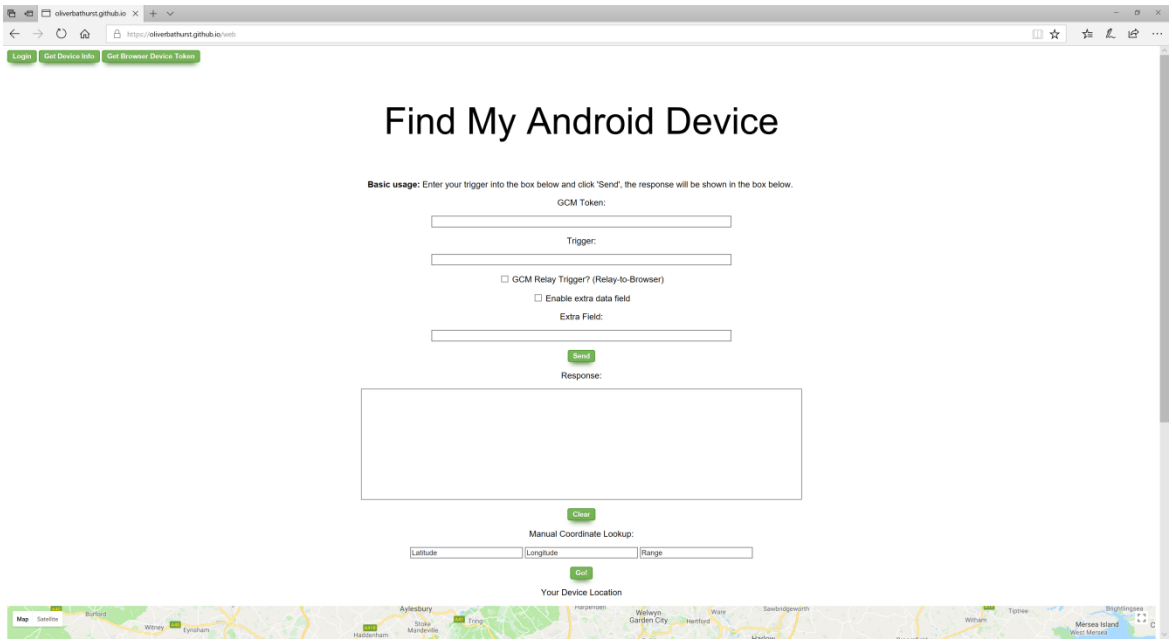


Figure 10. The final web interface design

12. Location Methods

12.1.1. *Obtaining a Location Fix*

To create a request for a location fix, Android's *LocationManager* class is used. This provides access to the system location services; these services allow applications such as Find My Android to obtain periodic updates of the device's geographical location based on parameters [32]. The level of accuracy that can be available to the application is largely intertwined with the permissions the user has granted the application, all *Location* public API methods require *ACCESS_COARSE_LOCATION* or *ACCESS_FINE_LOCATION* for full functionality. For example, if the *ACCESS_FINE_LOCATION* is not granted or revoked, the application will no longer be able to access the true results of GPS provider usage, instead returning an obfuscated level of accuracy. To determine the availability of a provider, the *LocationManager* class has a public Boolean API method *isProviderEnabled(String provider)*, returning true if enabled. This is used to determine which service should be used instead of a blind request implementation.

```
private boolean isGPSAvailable() {
    LocationManager locMan = (LocationManager)
        c.getSystemService(LOCATION_SERVICE);
    return (locMan != null &&
        locMan.isProviderEnabled(LocationManager.GPS_PROVIDER));
}
```

Figure 11. Checking availability of a location provider (GPS)

After an initial check, the location is enquired via the *requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)* method. The arguments accepted by this method are: provider e.g. GPS, Passive etc, the minimum time between location updates, the minimum distance between updates, and lastly, the listener for call-backs if necessary. Therefore, to immediately trigger a location request, the value of zero can be used for the minimum distance and time parameters. The location fix stored by the request can then be returned in the form of a *Location* object using the *getLastKnownLocation (String provider) LocationManager* method. This method returns a *Location* indicating the data from the last known location fix obtained from the given provider, *NETWORK*, *GPS*, or *PASSIVE* [33]. It is then necessary to obtain the location by *getLastKnownLocation* using the same provider used in the initial request, otherwise the last location extracted from other providers may prove outdated.

```
private Location getLocationByWIFI() {
    Location loc = null;
    try {
         LocationManager locationManager = (LocationManager)
c.getSystemService(LOCATION_SERVICE);

        if (locationManager != null) {
         locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
          loc =
         locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        }
    }catch(SecurityException ignored){}
    return loc;
}
```

Figure 12. Requesting and returning a location with Android's "network" provider using the LocationManager class

Once the *Location* object is returned, its public methods *getLatitude()*, and *getLongitude()* are used to derive meaningful coordinates for transmission for the user. These use the double data type for precision.

```
Location loc =
locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
Double d = loc.getLatitude();//fetch the latitude from the location object
Double e = loc.getLongitude();//fetch the longitude from the location object
```

Figure 13. Extracting meaningful coordinates from a *Location* object

12.1.2. *Find My Android Usage of Default Providers*

As with any location-oriented application, Find My Android makes use of the pre-existing services within the Android OS. For this, a specialised class, *LocationService* was created to encapsulate all location methods, as well as fetching specific device information, for use when relaying information. A singular method, *getLocation()* is used to return a *Location* object based

on a variety of variables. Firstly, the precedence of the providers is established, this is set by the user of the application via a configuration activity and saved to SharedPreferences, by default, the order of providers is influenced by potential for accuracy and thus: GPS, Wi-Fi and Passive. Subsequently, “GPS”, “Wi-Fi” and “Passive” are used as default strings when attempting to fetch provider names from preferences using “first”, “second”, and “third” as keys, such that if the user has not specified a precedence, the application will return these providers in the order: first, second and third.

These strings are then analysed by a secondary method *switchPref(String provider)* which switches the string name of the given provider and attempts to create and return a *Location* object based on the provider given. *switchPref()* then calls either of three methods: *getLocationByGPS()*, *getLocationByWIFI()*, or *getLocationByPassive()*, depending on the provider string. Each method returns a *Location* object, if unsuccessful, the object returned will be equal to a “null” pointer, which provides filtering capabilities to the calling *getLocation()* method, such that if usage of a provider returns a null object, the next provider can be used and so forth in a cascading pattern until a valid object is returned by either of the three specialised methods and subsequently *switchPref()*. If no valid location is returned, the *Location* object is bound to the default instantiated object of that class, therefore any future calls using *getLongitude()* and *getLatitude()* will inevitably emit the default double value of 0, which can then be identified by the user as a fault. Once the object is returned to its calling method, the coordinates can then be extracted and included within summaries to be transmitted.

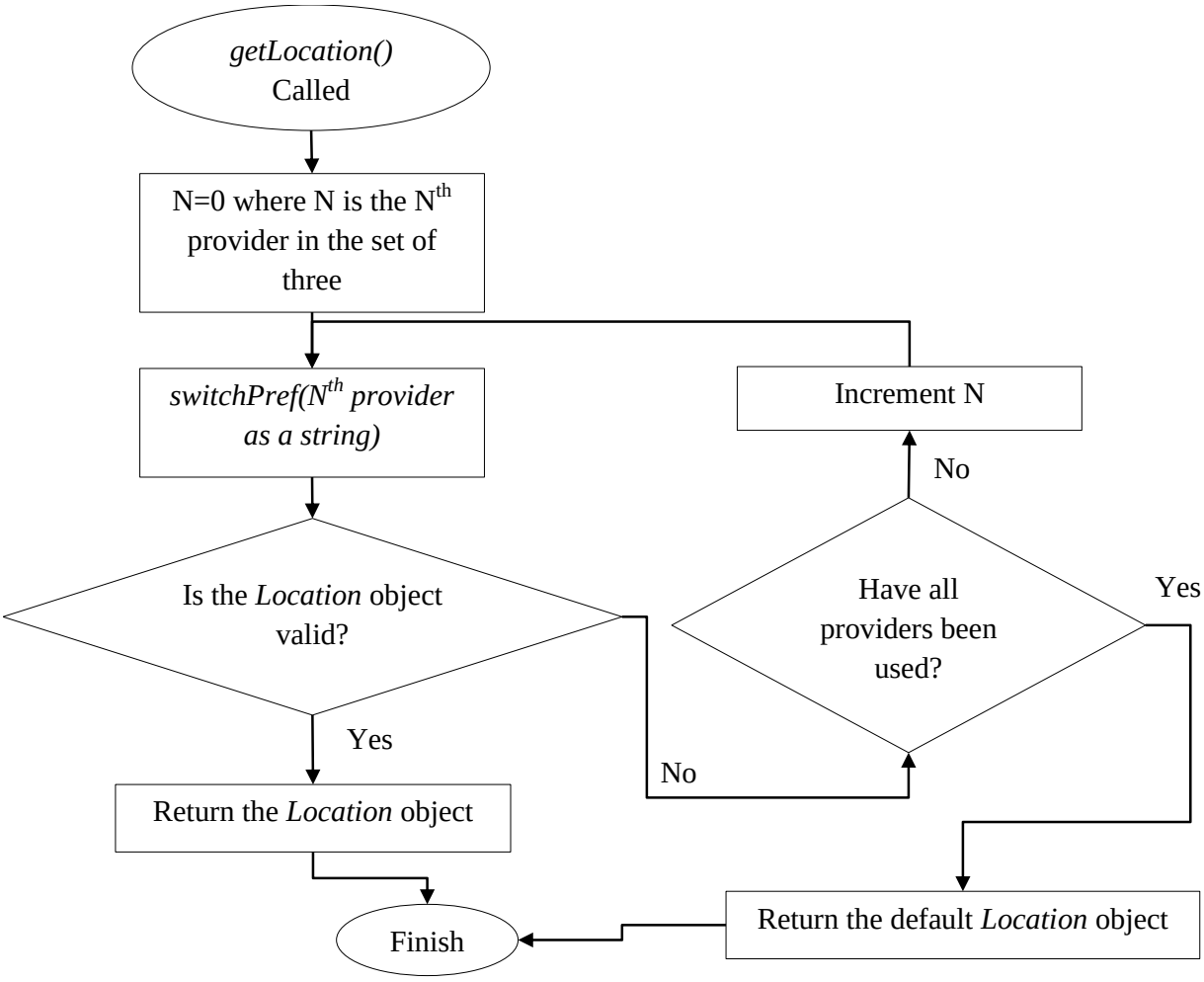


Figure 14. Flow of the *getLocation()*, returning a *Location* object

At each interval described above, *switchPref()* is called, this intermediate method switches the string provider name to decide which "*getLocationByX"* method to call, where X is the designated provider, and returns the resulting object.

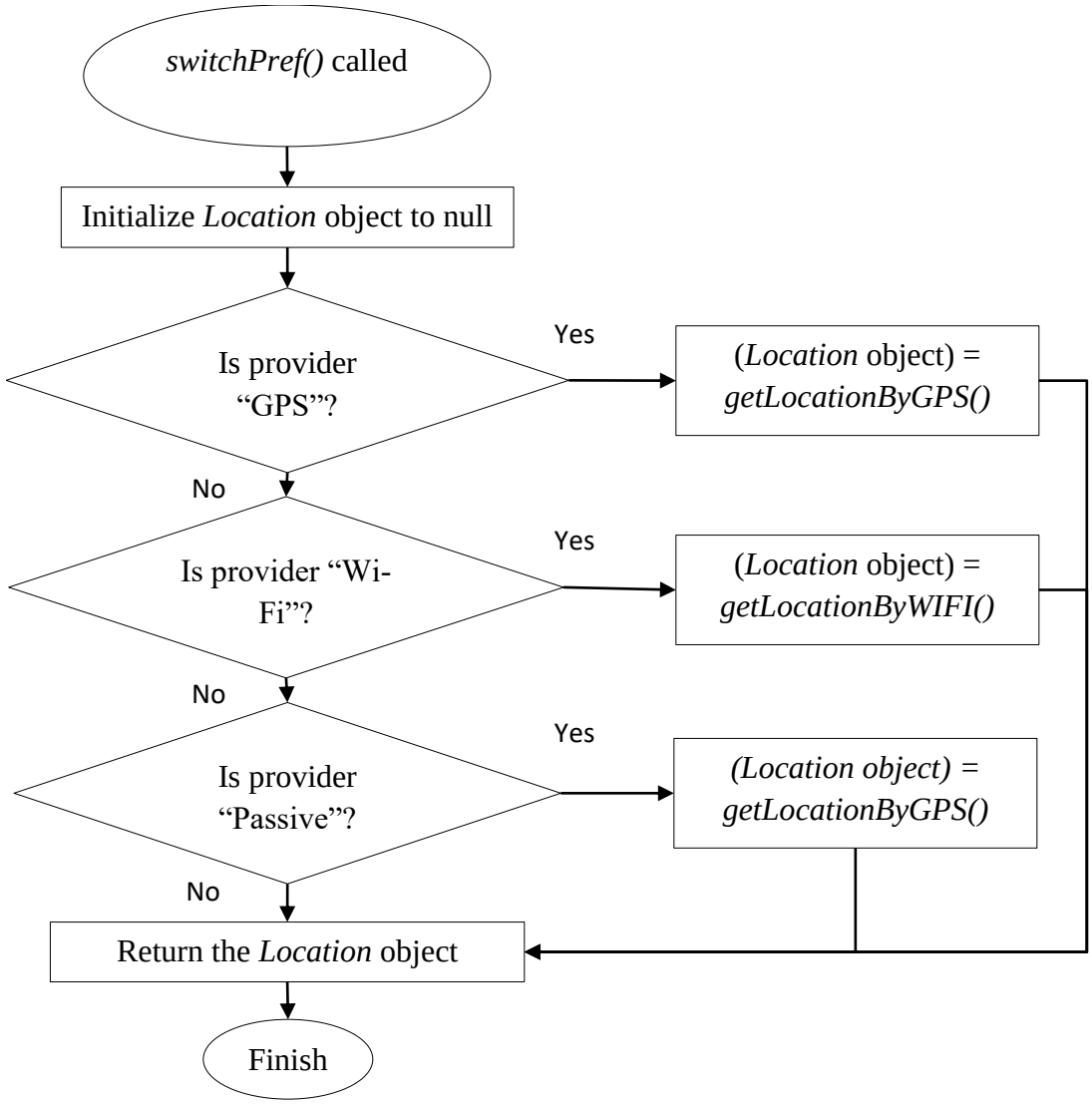


Figure 15. Fetching a *Location* object based on the string representation of a location provider

As shown above, the methodology of obtaining a location from a request given a provider follows a defined format that can be adequately represented with pseudo code. Thus, the three helper methods, *getLocationByGPS()*, *getLocationByWIFI()*, and *getLocationByPassive()* contain largely duplicated code.

```
GIVEN PROVIDER P
INITIALISE location object
CREATE immediate location request with P
FETCH P's last known location as an object
ASSIGN to location object
RETURN location object
```

Figure 16. Pseudo code of obtaining a *Location* object given a provider *P*

As previously stated, the *LocationService* class is used copiously throughout the application when returning the location coordinates to an external entity like the device's owner. These usages are then present in all scenarios in which the location is required, such as within the *SMSHelper* class, which provides a specialised method to collate identifiable information into a string object for transmission over SMS.

```
Location newLocReturn = locationService.getLocation();

return "This is a location alert, your device location is:"
 + newLocReturn.getLatitude()
        + "," + newLocReturn.getLongitude() …
```

Figure 17. An example usage of the *LocationService* class to produce Android-provided location coordinates

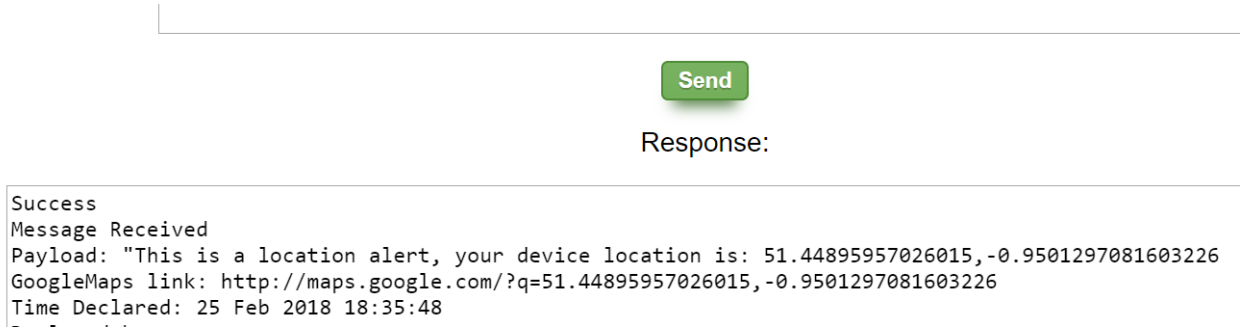


Figure 18. An example location response enabled by the *LocationService* class (using Cloud Messaging triggering and transmission)

12.2. *Introduction to Cell Tower Localization*

Mobile phone tracking traditionally used cell phone towers or "base stations" as a method of geolocation, with the multilateration or triangulation of the radio signals between numerous towers used to ascertain a location fix. The Global System for Mobile Communications (GSM) is based on the phone's signal strength to nearby antenna masts [34]. To receive the best quality of service and latency, a mobile device will typically connect to the closest tower of which the signal strength is highest; this indicates that the currently connected base station is the closest of

all nearby towers geographically to the device. In areas with a high density of towers, the location may then be narrowed to a smaller radius as the switching of a base station would imply a smaller change in physical location. However, in rural areas, the method remains ineffective. Positioning is then achieved by performing a triangulation calculation on the information retrieved from the base stations before being translated into a geographic location [34], a single base station may only produce a location accuracy of three-quarters of a mile, whereas information from multiple base stations can narrow the potential location to within fifty metres. Although this method of location tracking appears redundant in an age of the more accurate GPS and Wi-Fi positioning technology, cell tower localisation remains a necessary addition for devices that maintain only cellular capabilities or connection.

12.2.1. *Cell ID Databases*

Due to the limitations of the related Android APIs, although cell information can be attained, it alone is not enough to determine a physical location, and no standard Android API can programmatically perform this translation. Hence, a public database containing a comprehensive number of recorded/reported base station information is required to convert the raw cell data into meaningful information such as coordinates. For this, OpenCelliD, "The world's largest Open Database of Cell Towers" [35] was used, providing free querying of their database via an API key. The crowd-sourced nature of cell tower and Wi-Fi AP collection ensures that the database is constantly scaling with newly discovered points of reference, useful for future viability. Although this querying is dependent on the device's internet connectivity, redundancy is achieved by also including the raw cell information in the response when a cell tower fix is requested such that the external entity/user can use this information to manually infer location via public databases.

12.2.2. *Cell Tower Codes*

Commonly, but varies with network type adopted, each cell network base station (tower) broadcasts a minimal four codes, MCC, MNC, LAC, and CID [36].

MCC - Mobile Country Code. This code simply identifies the country. For example, the MCC of the United Kingdom is 234.

MNC - Mobile Network Code. This code corresponds to a mobile operator of the base station, such as Vodafone or BT.

LAC - Location Area Code, a unique number designated to a specific location area. A location area in this context is several base stations which are grouped to better optimize broadcasting.

CellID (CID) - the Cell Identification is unique number used to identify each base station or sector of a base station within a location area code (LAC).

12.2.3. *Cell Tower Information Retrieval*

Android's *TelephonyManager* class provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. [37] Understandably, the telephony service provided by Android, *TELEPHONY_SERVICE*, is unavailable on Android devices which do not have cellular capabilities and is excluded as a means of geolocation on such devices. A new instance of *TelephonyManager* is created with the retrieval of the system service.

```
TelephonyManager tel = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
```

To return all observable cell information from the device including primary and neighbouring cell towers, a method call, namely *getAllCellInfo()* is performed, returning a list of *CellInfo* objects (*List<CellInfo>*). Each *CellInfo* object is a superclass of, and is subsequently inherited by, four subclasses: *CellInfoCdma*, *CellInfoGsm*, *CellInfoLte*, *and CellInfoWcdma*. These objects reflect the different types of cell networks, CDMA (Code-Division Multiple Access), GSM (Global System for Mobile Communications), LTE (Long-Term Evolution), and WCDMA (Wideband-CDMA). Consequently, each *CellInfo* object's superclass can be checked via the *instanceof* operator, for example if the *CellInfo* object is of class *CellInfoGsm*, it can be cast as such.

```
if (cell instanceof CellInfoGsm) {
    CellInfoGsm identity = (CellInfoGsm) cell;
}
```

Once a *CellInfo* object has been cast to its subclass, to retrieve the identifying information, its identity must be returned. Each subclass of *CellInfo* described above contains the public method *getCellIndentity()*, this is used to obtain a cell identify object of that network type e.g.

```
CellIdentityGsm identity = (CellInfoGsm object).getCellIdentity();
```

Once this object has been created, the base station information can be extracted with API public methods, *getCid()*, *getLac()*, *getMcc()*, and *getMnc()* [38] to deduce the CellID, Location Area Code, Mobile Country Code, and Mobile Network Code respectively.

```
int cid = identity.getCid();//get cid and assign to an integer
int lac = identity.getLac();//get lac and assign to an integer
int mcc = identity.getMcc();//get mcc and assign to an integer
int mnc = identity.getMnc();//get mnc and assign to an integer
```

12.2.4. *Obtaining a Location Fix*

As previously stated, as of Feb 2018, no programmatic methods exist within Android APIs to perform accurate geolocation via obtained cell information. It is therefore necessary to use the raw data retrieved from the various API calls described above to translate into meaningful location information by querying a database. OpenCelliD, the database used for this project, requires a minimum of five parameters contained with a HTTP GET request to the web service's *get* script: the API key generated upon registration, alongside the MCC, MNC, LAC, and CID of the base station.

```
http://www.opencellid.org/cell/get?key=<apiKey>&mcc=<mcc>&mnc=<mnc>&lac=<lac>&celli
d=<cellid>
```

Figure 19. A sample GET request for location information (OpenCelliD)

Once a valid URL in the format described is resolved, the successful response, HTTP 200, is returned with a small XML document describing, among other things, the longitude, latitude and range of the base station.

```
<rsp stat="ok">
  <script/>
  <cell lat="51.446534" lon="-0.949507" mcc="234" mnc="20" lac="1432" cellid="66818" averageSignalStrength="0" range="837"
  samples="1" changeable="1" radio="" rnc="0" cid="0" tac="0" sid="0" nid="0" bid="0" message=""/>
</rsp>
```

Figure 20. Example response from OpenCelliD

To achieve this within the Android application, the connection to the URL, customised by inserting the parameters according to a single base station's information, is opened using the *HttpURLConnection* class' *openConnection()* method call. After setting the request method (GET) and the XML property of the request the output of the connection is obtained via the *getInputStream()* method. To faithfully extract the locational data from the XML response, the input stream is then parsed into a XML *Document* object using the XML parser class *DocumentBuilderFactory*. With knowledge of the XML layout, each cell tower element is stored within a *NodeList* by searching for elements with the tag name "`cell`" within the created document. Once the element is stored within the *NodeList*, the attributes of that element, containing the imperative cell data, can be retrieved and stored within a map via the *getAttributes()* method. Finally, the latitude, longitude and range nodes are withdrawn from the map by using `lat`, `lon`, and `range` as named item identifiers, the values of which are then parsed into the double format to retain precision before returning the results to the calling method.

```
HttpURLConnection connection = (HttpURLConnection) new
URL("http://www.opencellid.org/cell/get?key=978e483439b03p" + "&mcc=" + mcc +
"&mnc=" + mnc + "&cellid=" +cid + "&lac=" + lac).openConnection();
connection.setRequestMethod("GET");
connection.setRequestProperty("Accept", "application/xml");
Document doc =
DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(connection.getInput
Stream());
doc.normalize();
NodeList nodeList = doc.getElementsByTagName("cell");
if (nodeList.getLength() >= 1) {
    NamedNodeMap attributes = nodeList.item(0).getAttributes();
    Double[] doubleArr = new
Double[]{Double.parseDouble(attributes.getNamedItem("lat").getNodeValue()),
Double.parseDouble(attributes.getNamedItem("lon").getNodeValue()),
Double.parseDouble(attributes.getNamedItem("range").getNodeValue())};
}
```

Figure 21. Parsing the XML response of an OpenCelliD request to obtain coordinates

12.2.5. *Returning Location Fixes*

Returning cell locations relies on applying the same methodology described above on all observable cell base stations. Once a trigger is received requesting the data an object of the helper class, *CellTowerHelper*, is instantiated. This class is used to obtain a string of all primary and neighbouring cells along with their coordinates if available. For each object within the list, the subclass is identified. Once each *CellInfo* object is casted to an object of their subclass, their related cell identity object is recovered to utilise the API calls for the aforementioned cell identifiers. The raw data contained within each *CellInfo* object, including the CID, LAC and other insightful data, is added to the class' return string before being passed to the method responsible for attempting a location fix using the OpenCelliD database. This method modifies the standard URL with the object's identifying codes as parameters, the connection is opened, an XML document object parsed from the input stream, the coordinates derived from the document, and finally the string concatenated with the additional information. Once this has been completed for every encountered cell, the final string is returned to be transmitted to the external user.

12.2.6. *Conclusion*

As previously emphasized, Cell ID's suitability as a means of geolocation extends only to those devices which do not bestow the conventional location providers that the Android OS provides, or in scenarios where the location capabilities have been turned off, potentially by a rogue entity. In which case this method provides a useful bypass in the case of theft. In a general sense there is no benefit to using this functionality other than in the cases described above, the extremely limited accuracy of around 700m pales in comparison to, for example, GPS's optimal 4m accuracy. In instances where this level of accuracy is returned, the retrieval of a stolen device may better be accomplished by police using the device IMEI number present in all cellular devices.

12.3. *Fingerprinting*

A fingerprint, within this context, can be adequately described as a collection of objects or references that uniquely identifies a given instance of something. The act of fingerprinting is therefore the creation of multiple fingerprints; these fingerprints can then be used in the pursuit of achieving higher-level goals such as data mining. Their uniqueness allows for the differences between them to be used as a metric, where a given collection of objects can be compared to that within a fingerprint to identify suitability for a given purpose. Within this application, fingerprints of Bluetooth and Wi-Fi metadata are used as metrics to determine the likeliest location of the host device with respect to a current instance of collected metadata; the fingerprints are also given context by the user by assigning a location alias to each. With this information, fingerprints can be compared to recent metadata to estimate the location of the device, returning the fingerprint's identifying alias (name). This section concerns the two methods of fingerprinting: Bluetooth and Wi-Fi, it will cover the design and contents of the fingerprints, the Android implementation of the creation and retrieval of these fingerprints and associated testing.

12.3.1. *Bluetooth Fingerprinting*

12.3.1.1. *Introduction to Bluetooth*

The Bluetooth™ [39] wireless technology solution is designed as a short-range connectivity solution for personal electronic devices [40]. Originally created in 1994 [41], Bluetooth has developed and progressed to the present day with evolutions in speed and range. Its open standard for wide-range conformance, low cost, and low power makes its uptake extraordinarily prevalent in handheld devices such as mobile phones and televisions where file transfers, wireless synchronicity, and internet connectivity are demanded by the consumer. Bluetooth technology works as an alternative to data cables by exchanging data using radio transmissions along the ISM (industrial, scientific and medical) band of 2.4GHz using a signal at a rate of 1600hop/s, 1600 changes in wavelength per second, to mitigate interference from devices on the same wavelength [41]. The strength of the Bluetooth signal is proportional to the power of the device, measured in mill watts, specialist devices such as Bluetooth LE (Bluetooth Low Energy) beacons operate on a reduced level of power and as such have a smaller range, adding conciseness for locational information detailed later in this paper.

Class	Power (mW)	Power (dbM)	Distance (m)	Sample Devices
1	100	20	~100	BT Access Point, dongles
2	2.5	4	~10	Keyboards, mice
3	1	0	~1	Mobile phone headset

Figure 22. Classes of Bluetooth devices [41]

12.3.1.2. *Android's Bluetooth APIs*

Like many technologies within mobiles phones, the Android OS has various API calls that can be performed on devices that have Bluetooth capabilities:

Using the Bluetooth APIs, an Android application can perform the following:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections [42]

In order to utilise Bluetooth features, two permissions are declared in Android's AndroidManifest.xml, `BLUETOOTH`, for basic communication of any kind and more crucially, `BLUETOOTH_ADMIN`, for the ability to perform scans and discovery of local devices. [42]

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Figure 23. Declaration of required permissions (AndroidManifest.xml)

Some Android devices have support for what is called "Bluetooth Low Energy", the Android 4.3 OS introduced support for BLE and various API calls that applications can use to discover and transmit data to such devices. BLE is designed to utilise less power and facilitates the communication with the devices that have stricter requirements such as beacons, fitness trackers etc [43]. To declare support in the manifest, and imply that the application is available to BLE and non-BLE devices, a "uses-feature" XML node is placed with the Bluetooth LE hardware feature defined. The flag "android: required" is set to "false" to allow communication with both types of devices, and enables non-BLE enabled Android devices to access features such as scanning within the application. If the device does not support BLE, the results of a scan will return only non-BLE devices.

```
<uses-feature
    android:name="android.hardware.bluetooth_le"
    android:required="false" />
```

Figure 24. Declaration of BLE feature requirement

12.3.1.3. *Performing scans*

For the purposes of this paper, the scanning of Bluetooth devices is covered exclusively for the purposes of fingerprinting.

The first stage of scanning and any Bluetooth operation is the initialisation of a Bluetooth adapter object to `BluetoothAdapter.getDefaultAdapter()`. If a device does not support BT, the object will equal null, and no further action should be taken. The second step is to enable the non-null adapter if not enabled currently. It is best practice to prompt the user to do this by calling `startActivityForResult()` with the static BT enable request code: "`REQUEST_ENABLE_BT`" and Bluetooth Adapter intent: "`BluetoothAdapter.ACTION_REQUEST_ENABLE`."

```
BEGIN
      GET current Bluetooth adapter
      IF adapter is null
            PRINT Bluetooth not available, fingerprinting not possible;
            STOP;
      ELSE
            IF Bluetooth not enabled
                  Request BT enable;
            ELSE
                  Start Fingerprinting Activity;
END
```

Figure 25. Pseudo code of enabling Bluetooth and checking availability

The application then needs to declare a call-back for the request called `onActivityResult()`, this intercepts the result code from the request, if the code equals "`RESULT_OK`", the request has been satisfied, else the request was rejected by the user and the application/feature should not proceed.

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_ENABLE_BT) { //if the request is to enable
         Bluetooth
        if (resultCode == RESULT_OK) { //if enabled by user
            startActivity(new Intent(SettingsFragment.super.getActivity(),
        }
    }
}
```

Figure 26. *onActivityResult* call-back example

And finally to perform a scan the `startDiscovery()` API call is made on the adapter object. To receive updates on the devices discovered, a broadcast receiver must be introduced to capture the `BluetoothDevice.ACTION_FOUND` intent that's fired when the device-found intent is

fired by the OS. The `BluetoothDevice.ACTION_DISCOVERY_FINISHED` intent is used such that once the intent is received, discovery can be restarted to update the GUI with the latest results.

```
registerReceiver(mReceiver, new IntentFilter(BluetoothDevice.ACTION_FOUND));
registerReceiver(mReceiver, new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED));
if(blue != null){
    blue.startDiscovery();
}
```

Figure 27. Registering receivers for results of a scan/starting Bluetooth discovery on a Bluetooth adapter object

Once a device has been found, the extra device can be cast to a `BluetoothDevice` object for extra analysis by calling `getParcelableExtra()` on the intent object that's passed into the receiver.

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (BluetoothDevice.ACTION_FOUND.equals(intent.getAction())) {
         BluetoothDevice  btDev =
         intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        }
    }
}
```

Figure 28. Example BroadcastReceiver to catch the "Bluetooth device found" intent, and creating an object of that device

12.3.1.4. *Positional Insights with Bluetooth*

Due to the broadcasting nature of Bluetooth devices it is possible to analyse the signal strength (RSSI) of such devices. By doing this, a short-range location fix can be obtained by inferring from generalised signal strength and distance data. Naturally, this method of geolocation only proves useful if the user is knowledgeable concerning the whereabouts of the different devices. A general approach is not recommended however, the varying power usage of Bluetooth devices greatly affects their range such that any distance report obtained from tables of results can be nullified as inaccurate. Therefore, a level of calibration is required to analyse the RSSI-to-distance relationship for each device separately to more accurately determine distance from each device. For this, the Android application was equipped with Bluetooth scanning and individual device calibration features to adequately confirm the distance the device is away from each Bluetooth device based on the distance measurements given by the user and the BT devices' signal strengths, both attained during the calibration phase.

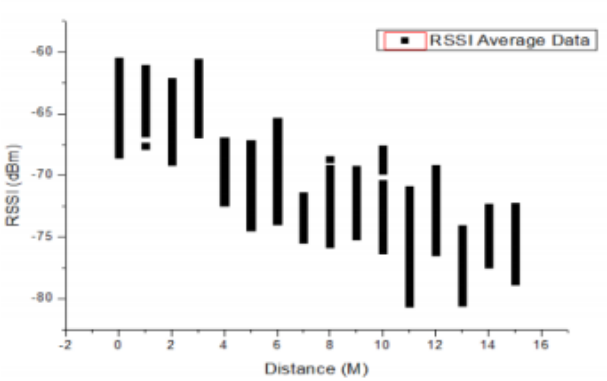


Figure 29. General graph on average RSSI over distance [44]

12.3.1.5. *Performing Bluetooth Fingerprints with Android*

Due to power variations, and subsequently signal strength variations across distances, to establish an accurate fingerprint of a Bluetooth device, a certain level of calibration is required. This section will detail the implementation of Bluetooth/Bluetooth Low Energy fingerprinting within the Android application. To comprehend how far away a device is away from the Android device, it's necessary to establish an equation. For simplicity, this was chosen to be the RSSI (signal strength) divided by the distance in CM, this gives us the change in signal strength per CM of distance from the device. This calibration is performed multiple times at different distances to obtain an accurate average of the variance per CM ($\frac{\Delta RSSI}{cm}$). (Δ denotes change).

$$device_{new\frac{\Delta RSSI}{cm}} = \frac{\left((last\ saved\ reading) + \frac{RSSI\ in\ dBm}{distance\ in\ CM}\right)}{2}$$

Figure 30. The calibration of a device by calculating the signal strength change per CM of distance and averaging with the last saved reading.

Therefore, to get the distance of a calibrated device *D*, given the RSSI signal strength of *D*:

$$D_{distance}\ (CM) = \frac{RSSI\ in\ dBm\ (D)}{Current\ saved\ average\ dBm/xCM\ for\ D}$$

$$= \frac{RSSI\ in\ dBm}{dBm/xCM}$$

$$= RSSI\ in\ dBm(\frac{1}{\frac{dBm}{xCM}})$$

$$= RSSI\ in\ dBm(\frac{xCM}{dBm})$$

$$= \cancel{RSSI\ in\ dBm}(\frac{xCM}{\cancel{dBm}})$$

$$= xCM$$

Figure 31. Calculation of device distance given the device's signal strength and saved dBm/CM ratio for that device.

Therefore, if the Android device is in range of a calibrated device and subsequently receives a RSSI from the device's broadcast, a distance can be calculated.

12.3.1.6. *Android Implementation*

The Android implementation consists of two parts, a GUI element for scanning/discovering and calibrating, resetting and deleting Bluetooth devices, and an underlying non-GUI helper class which performs a Bluetooth scan and returns the distances of devices that are both found in the scan and have been saved (calibrated) by the application (matched devices). This section covers the calibration and retrieval of distances processes. The first step is the enabling of the Bluetooth adapter, a broadcast receiver is then registered with two `IntentFilter` objects, these filters specify the types of intents to be caught by the receiver. In this case, the `ACTION_FOUND` intent is registered to obtain new devices discovered in order to display them to the user, and secondly `ACTION_DISCOVERY_FINISHED` which is fired at the end of the discovery cycle, this is registered such that the discovery process can restart when the intent is fired, creating a constant feedback to the user of available devices. Finally, discovery is started.

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (BluetoothDevice.ACTION_FOUND.equals(intent.getAction())) {
            if(deviceList != null){
                BluetoothDevice btDev =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if(!deviceList.contains(btDev)){
                    deviceList.add(btDev);
                    redrawListView();
                }
            }
        } else if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(intent.getAction())) {
            blue.startDiscovery(); //restart discovery
        }
    }
};
```

The receiver for the GUI's activity performs two functions: is the intent is a found device, a `BluetoothDevice` object is created from the intent and added to an `ArrayList` data structure if

not already added, this object represents a single physical Bluetooth device. The second restarts the discovery process if finished to feedback updated results to the user. `redrawListView()` then presents the updated list of results to the user by iterating over all `BluetoothDevice` objects stored and printing the names and attributes of the devices with API calls such as `getName()` and `getAddress()`. To hold all saved Bluetooth device objects, an `ArrayList<BluetoothDevice>` data structure is initialised, as no searching is performed to warrant a `HashMap` implementation. Once a user adds a device to be saved into preferences, a check is performed to identify if the serialised data structure already exists in shared preferences, by using a common key "BeaconKeys", the retrieval process entails getting the associated value with that key:

```
PreferenceManager.getDefaultSharedPreferences(context)
        .getString("BeaconKeys", null);
```

This returns a string that equals null if no corresponding value exists for that key and a new data structure needs to be written to that key with the new device added, if not, the value (String) associated with that key is deserialised using Google's Gson library into the `ArrayList<BluetoothDevice>` type format, the new Bluetooth device added, and the data structure written back after being serialised into a JSON-formatted string, again using the associated key. SharedPreferences enables the application to save all Bluetooth devices into a single data structure under a single key which persists indefinitely across reboots and launches, allowing the helper class to access all saved devices.

```
private ArrayList<BluetoothDevice> getBTArray(){
    return new
       Gson().fromJson(PreferenceManager.getDefaultSharedPreferences(context)
       .getString("BeaconKeys", null), new
       TypeToken<ArrayList<BluetoothDevice>>() {}.getType());
}
```

Figure 32. Method that deserialises a JSON string from preferences into an `ArrayList<BluetoothDevice>` data structure and returns

To calibrate a device, the structure is deserialised from preferences (into an `ArrayList<BluetoothDevice>` object) and the device object at the required index is sent to the `BTConfig` activity. This is achieved by a customised Intent, using the `putExtra()` API method call to attach the JSON-serialised string version of the device object along with a key to the `startActivity()` request. Using `putExtra()` allows the efficient transmission of objects of certain data types such as strings between activities, along with keys, these objects can then be retrieved in the proceeding activity.

```
startActivity(new Intent(getBaseContext(), BTConfig.class).putExtra("BT_DEVICE",
new Gson().toJson(get.get(selectedIndex))));
```

Figure 33. Starting the `BTConfig` calibration activity, passing a single JSON-serialised `BluetoothDevice` object with key "`BT_DEVICE`"

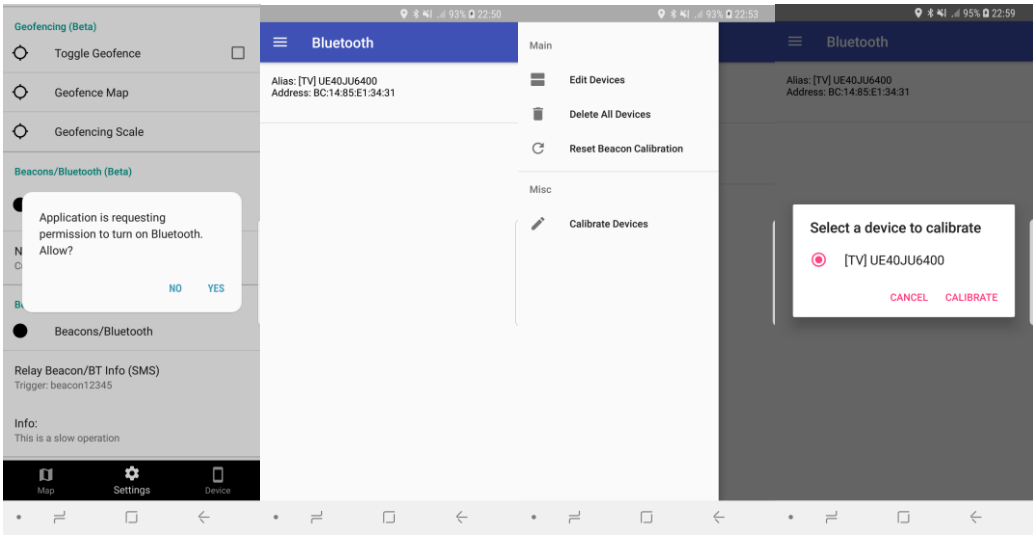


Figure 34. Bluetooth GUI, used to select devices for calibration, editing, and deletion

12.3.1.7. *The Calibration Activity*

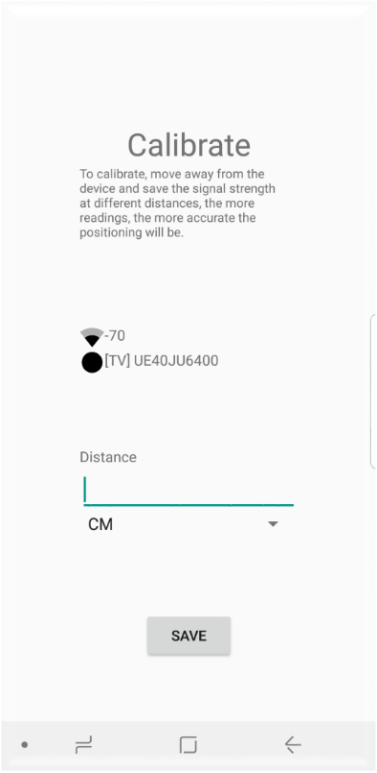


Figure 35. Calibration GUI

To effectively calibrate a device, there are certain conditions that are imposed by the Android operating system. Firstly, to receive the devices' signal strength (RSSI), the device cannot be paired (connected) to the Android device, the RSSI of a device is only encountered and obtainable during the discovery lifecycle. This necessitates consistent, repeated Bluetooth scanning to update the GUI with the latest reported signal strength. Secondly, if the device in question is cast to a Bluetooth device object, the RSSI will remain static and not change or respond to differing distances. To mitigate both nuances of the API, a comprehensive scanning-reporting solution was chosen. Once the `BluetoothDevice` object, the device being calibrated, is deserialised from the activity's Intent, it's used as a class member, this is the object that is used when validating found devices upon a scan result. A scan is then executed, if a device is found, its address (MAC) is compared against the device that's being calibrated (the `BluetoothDevice` object). If they match, the RSSI signal strength is retrieved using a call to `getShortExtra(BluetoothDevice.EXTRA_RSSI)` on the intent, this is then updated as the latest reading. The discovery process is then restarted to rescan for the device such that whenever the matching device is found, the process restarts to consistently receive the latest RSSI reading. The choice of using the MAC address was a simple one, if one were to use the device name for matching, any change in this identifier, via a user changing their device name, the calibration

process would fail, requiring a full recalibration of the newly changed device. The MAC identity was then adopted for matching due to its ability to uniquely identify hardware whilst simultaneously being harder to change or spoof on most devices.

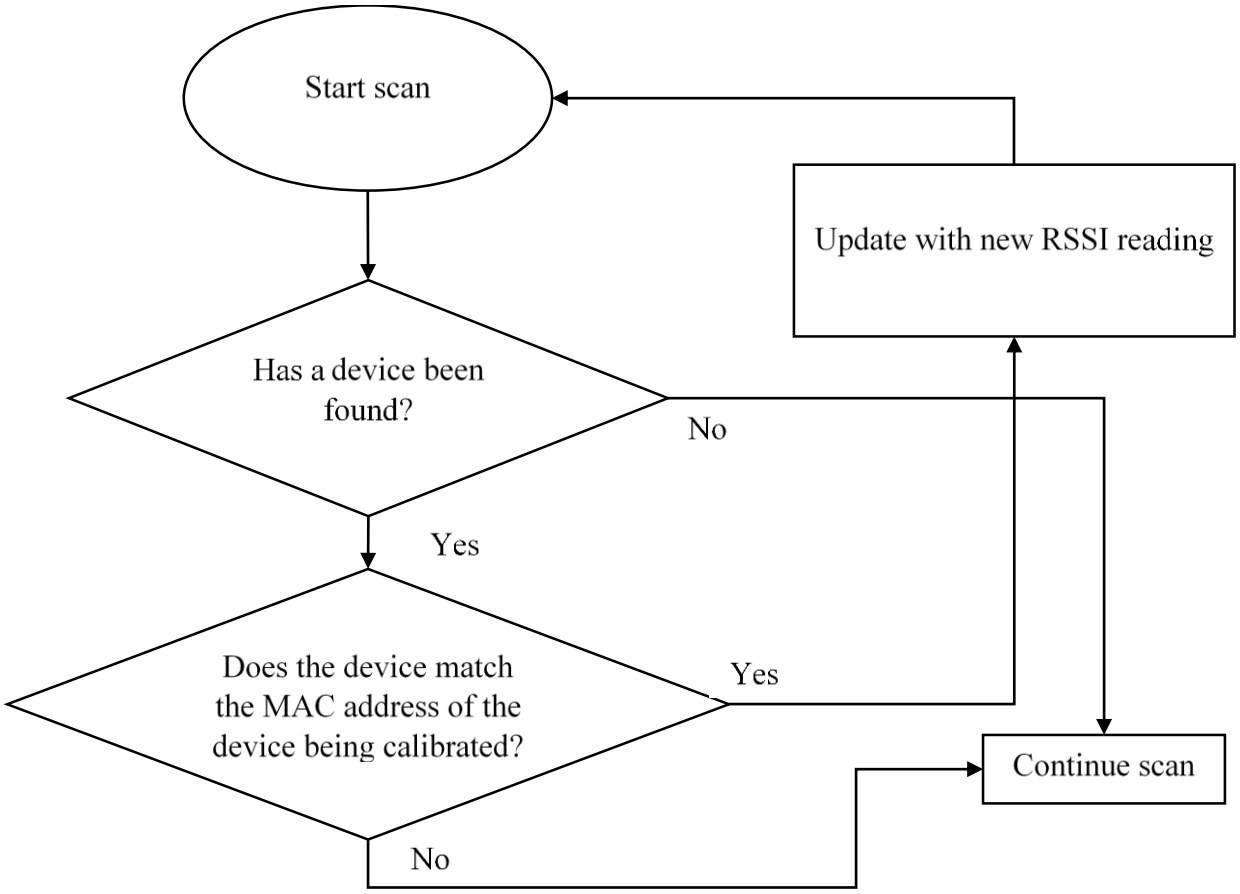


Figure 36. Continual RSSI scanning solution

Upon using the 'Save' feature, a fingerprint is taken. This involves taking the user-inputted distance measurement and the current RSSI, calculating the dBm/CM at that instance, averaging with the previously saved reading if available, as shown earlier in the chapter. The device dBm/CM is then saved to preferences as a float number, using the device name as the key, this is the number that's retrieved when calculating distances when a position update is requested by the user.

12.3.1.8. *Returning Results*

To return the distances of nearby, calibrated Bluetooth devices, a simple process is performed within the `BTNearby` class. A single scan is performed, each device found is added to a single data structure with their name and RSSI reading at that instance, once the scan has completed,

each one is compared against the deserialised list of saved devices from SharedPreferences. If a match is made, the float value of the calibration reading (dBm/CM) is retrieved from storage by using the device name as a key. If found, the RSSI received from the scan is divided by that reading to return a distance in CM, as described in the fingerprint equation.

12.3.1.9. *Testing*

To access the accuracy of such an implementation, two Bluetooth-enable devices need to be present: a broadcaster, to send the Bluetooth signal, and a receiver running the Android application to create fingerprints of the broadcaster. Due to the nature of fingerprinting, it's necessary to perform two types of test to determine whether the accuracy of static fingerprinting (at one given distance) improves with a greater number of scans or whether fingerprinting at multiple different distances improves upon the accuracy of the static fingerprinting method.

For this test, two devices were chosen based on their availability to the researcher, a Sony Xperia L1 was used as a broadcaster, and a Samsung Galaxy S8 was used as the finger printer. The methodology for the static test was to take N=6 calibrations and reported distances at one distance D, chosen to be a random distance of 100cm, a reading was then taken at a different distance (200cm) to the device and its accuracy compared to the dynamic fingerprinting method. The dynamic method involved taking an equal number of fingerprints, but at different distances: 25cm, 50cm, 75cm, 100cm, 125cm, 150cm, and the reported distances noted. For control with the previous method, a reading was then taken at 200cm and compared. To further reduce inconsistencies, the two devices were placed in a clear sightline of each other with no obstacles in between. To receive the reports, the associated GCM relay trigger was used to send the results to the researcher's browser instance over the internet.

The static calibration test results were as follows:

Number of Calibrations at 100cm	Reported Distance (cm)
1	105.77125
2	95.56993
3	102.66494
4	110.94437
5	101.21562
6	108.28227

Table 3. 100cm static calibration results

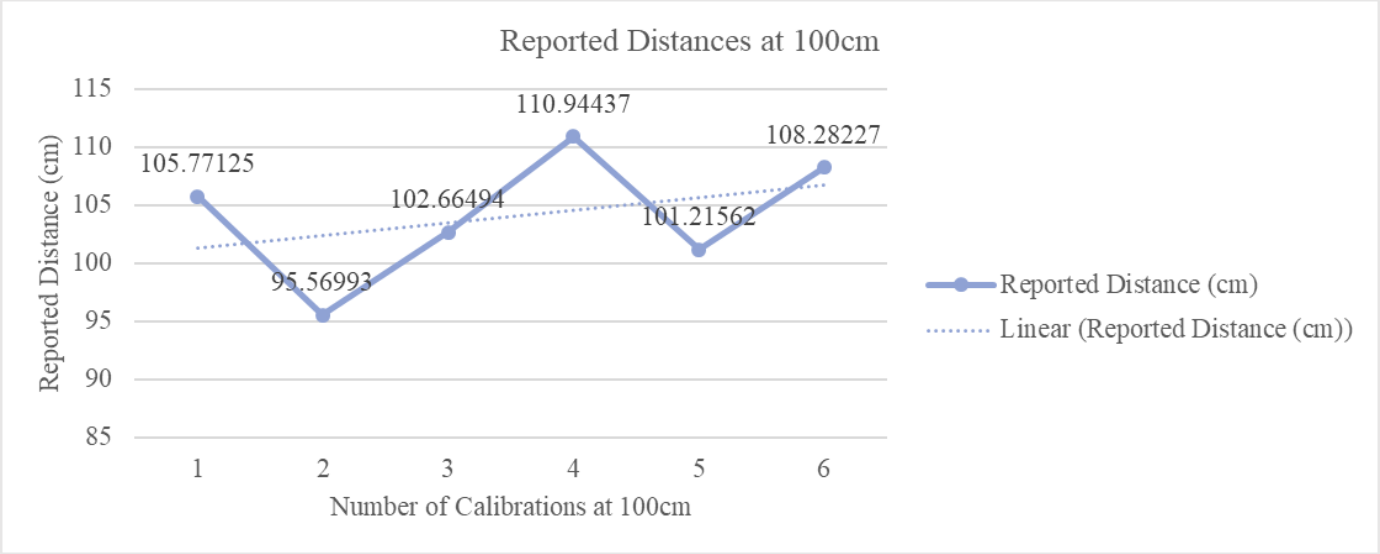


Figure 37. Graph of reported distances at a static 100cm distance

At 200cm a result of 87.58125cm was returned for the static method.

The dynamic calibration test results were as follows:

Calibration Distance (cm)	Reported Distance (cm)
25	51.85185
50	44.715446
75	64.45013
100	78.11023
125	108.30876
150	139.85475

Table 4. Dynamic calibration results

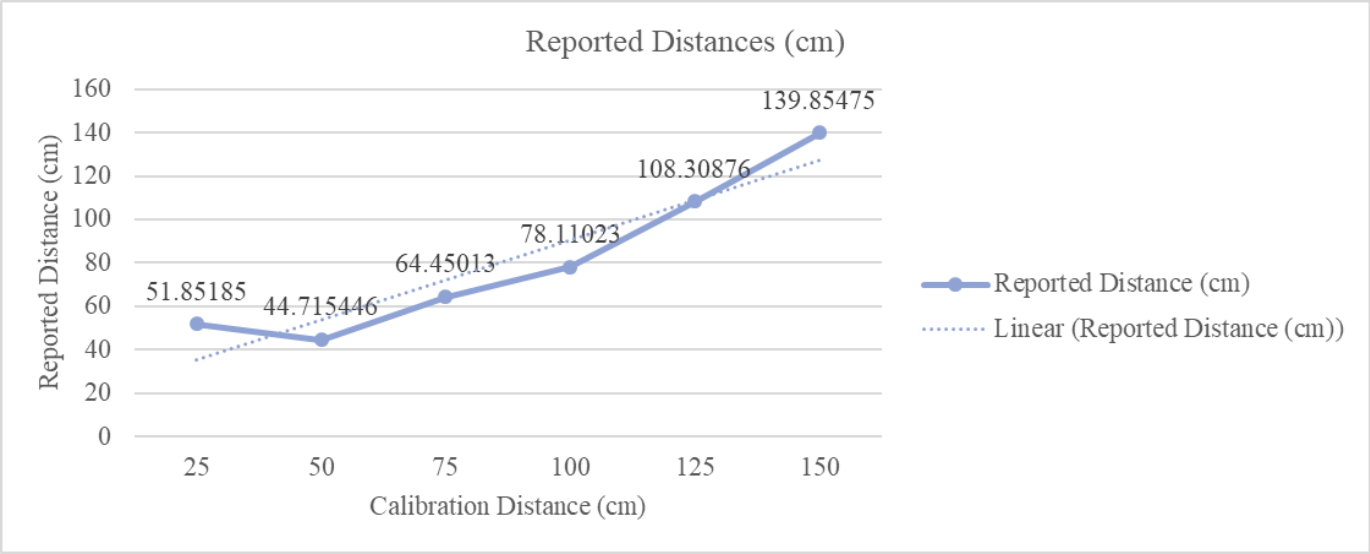


Figure 38. Graph of reported distances at varying distances

At 200cm a result of 147.81798cm was returned for the dynamic method.

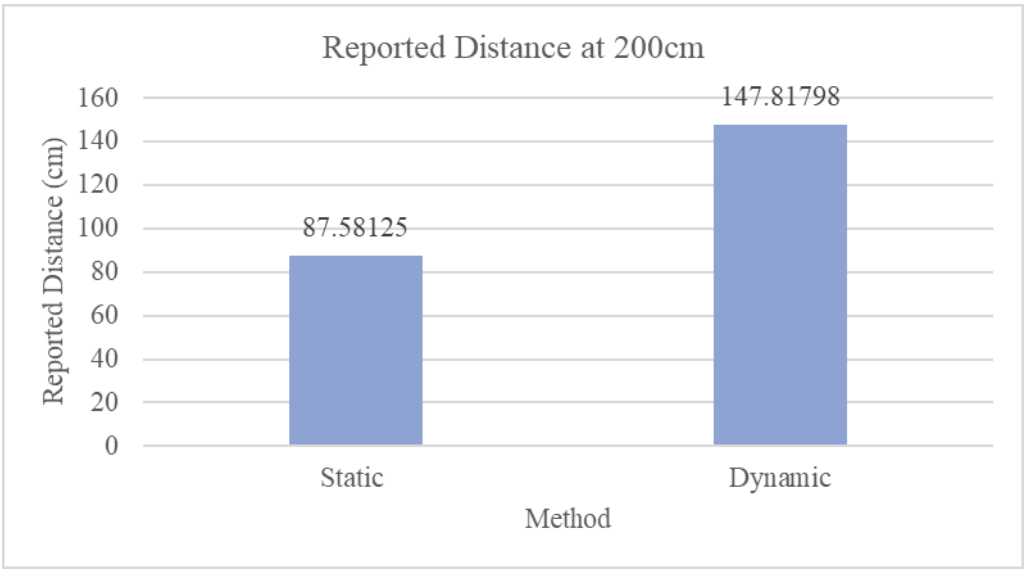


Figure 39. Differences in distance reporting at a distance (200cm)

12.3.1.10. *Analysis*

The static method proved to be consistent over the number of scans performed, the reported distance varying between 95 and 110cm, producing a standard deviation of 5.48. Although the SD suggests in this context that the results are acceptably spread and shows a relatively small error margin regarding the actual distance of 100cm, the same cannot be replicated at the larger distance of 200cm. At 200cm, the reported distance falls behind the dynamic method at a reported 87cm, suggesting the signal strength of the device remains consistent over that range, causing the algorithm to suggest closeness. However, the static method does well to recognise the subtle deviation in strength at closer distances. The graph has a slight upward inflection, potentially caused by the small differences in signal when performing the reporting and the weak trend does not provide further insight. Oppositely, the dynamic method correctly distinguishes the differences in distance inferred by the changes in strength, with a standard deviation of 36.49 adequately reflecting the required spread in reported distances over the six increasing distances. At lower distances, and subsequently a lower number of calibrations/fingerprints, at distances 25, 50 and calibrations 1, 2 respectively, the results returned show a much larger error margin in comparison to the static method. However, as the distance D and calibrations N increase, we see a linear trend upwards from D=50 to D=150, that is, as the distance increases, the reported distance also increases uniformly, giving a mean, -6(cm) disparity between the actual and reported distances, with the most accurate result at 150cm. The same uniformity then breaks when the reading at 200cm is presented (147.81cm), showing a 52cm difference, although the change in distance is recognised with respect to the previous reading at 150cm (139.85cm), this is insufficient.

12.3.1.11. *Conclusion*

The accuracy of Bluetooth fingerprinting is then dependent on how it's adopted, the universal algorithm produces inconsistent results in both static and dynamic calibration scenarios. However, they fit two unique use cases. Static calibration serves results with a small spread of data and a low standard deviation, the caveat being that it's an inadequate technique for extrapolating and estimating different distances based on the single calibration distance. This method should therefore only be considered viable if the broadcaster device remains stationary in a confined location where the Android device may be a set distance away. Alternately, dynamic tends to scale much better at varying ranges. The differing signal strengths at different distances improves the accuracy of the algorithm at distinguishing distance as seen above and can adequately differentiate between distances based on signal strength, although not pinpoint, it suggests that the algorithm works well as an approximation tool, where the report is within an acceptable margin of error for the user. As a linear, positive trend is shown with the dynamic method, using predictive reasoning, the trend will continue over more calibration distances, making this method a realistic choice for portable Bluetooth device calibration. A major concern to the development of Bluetooth fingerprinting is the increasing range and power of embedded Bluetooth chips, such as those found in mobile phones. This makes them increasingly redundant for short-range positioning, as their RSSI remains stable over larger distances, the algorithm becomes less effective at identifying distance. It may be that the future of this method aligns

more with medium-range geolocation, or as a viable solution for lower powered devices that complement the algorithm's methodology such as BLE-enabled beacons.

12.3.1.12. *Bluetooth Low Energy Beacons, an Alternative*

Bluetooth Low Energy (BLE) beacons are an attractive solution for a plethora of Internet of Things (IoT) applications, from micro localization to advertisement and transportation. BLE beacons are small, low cost devices that are capable of providing contextual and locational information to the users [45]. Their low power can result in lifespans of over two years which makes them suitable for constant operation as location-insightful sources, with trilateration achieved using N>1 beacons for increased accuracy when in range of all. Due to their low power and subsequently lower range, BLE beacons are commonly used to distribute personalised messages at different points of interest such as a specific section of a department store. BLE beacons don't conventionally support connectivity, they instead broadcast their identifier to nearby electronic devices, such that other devices can perform actions when in proximity. Due to the one-way, broadcast-only nature of BLE beacons, a software application is necessary to be installed on a device, such as an Android mobile phone, to facilitate and perform actions when within range of a beacon. This section includes the calibration of a generic Bluetooth beacon within the Android application and the efficacy and accuracy of the positioning algorithm when used in conjunction with said device. As with conventional Bluetooth devices the received signal strength of the beacon is used as the indicator of distance.

12.3.1.12.1. Beacon Device and Initial Observations

Figure 40. The Bluetooth beacon used for testing

To investigate the usefulness of BLE beacons, a generic BLE beacon, seen above, was used to analyse the variations in signal strength over distance. Initial testing involved obtaining the RSSI of the beacon at N=7 different close distances from the application's host Android device (N=30, 25, 20, 15, 10, 5, 0), taking N=5 samples at each distance. The samples were averaged to obtain an average RSSI for each distance and the results plotted.

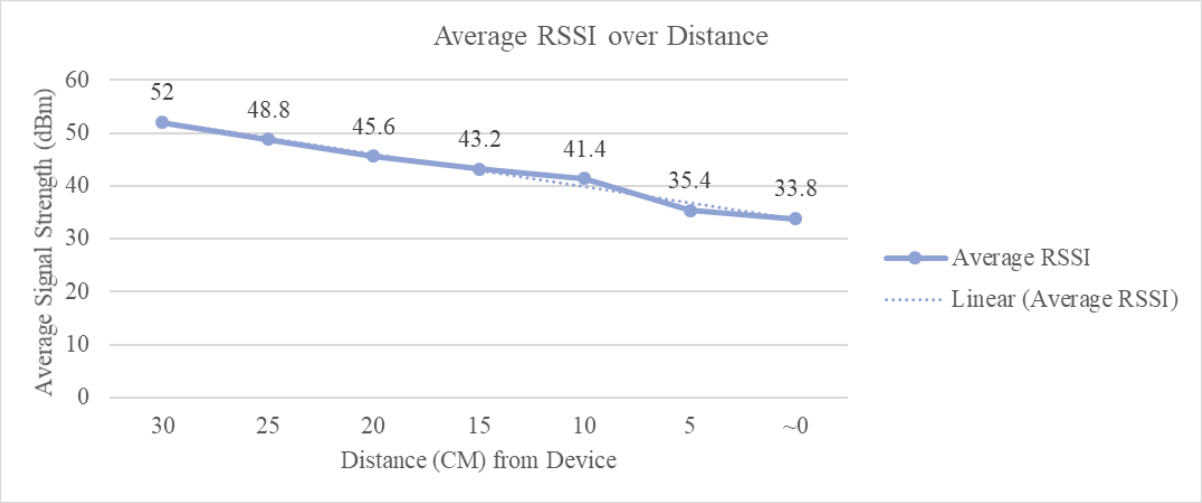


Figure 41. Results of RSS increases caused by closeness to application's host device (0 dBm being the strongest RSS possible)

As shown from the graph above, the RSS increases linearly as the beacon is moved closer to the device. What is particularly notable about the results is the large increase (N=28.2dBm) in signal strength over a negligible distance of approximately 30cm, behaviour not seen with other Bluetooth standards, such as those implemented within mobile devices investigated previously in this chapter with which connectivity range is commanded by users. However, this variance in signal strength is expected for the nature of the device and desirable for the purposes of this application. The trend shown results in a greater level of fine-grain positional accuracy being available over short distances, as the receiving device reports distinct strength changes over minimal distance, this nuance can be exploited by the algorithm to produce a more accurate average dBm/CM over a conventional Bluetooth device.

12.3.1.12.2. Testing

Like the previous testing methodology, the calibrations were taken with two different strategies, static and dynamic. The static test was to take (N=6) calibrations at a singular distance (D=100cm) and plot reported distances, a reading was then taken at a different distance (D=200cm) to the beacon. The dynamic method involved taking an equal number of fingerprints, but at different distances: (D = 25cm, 50cm, 75cm, 100cm, 125cm, 150cm), and the reported distances noted. For control with the previous method, a reading was then taken at 200cm and compared. The beacon and the calibrating Android device were placed in clear sightline of each other, to receive the estimated distance reports, the associated GCM relay trigger was used to send the results to the researcher's browser instance over the internet.

Number of scans (N)	Reported Distance (CM)
1	102.16867
2	109.50023
3	98.27763
4	111.45528
5	100.355675
6	102.085945

Table 5. Static beacon calibration results

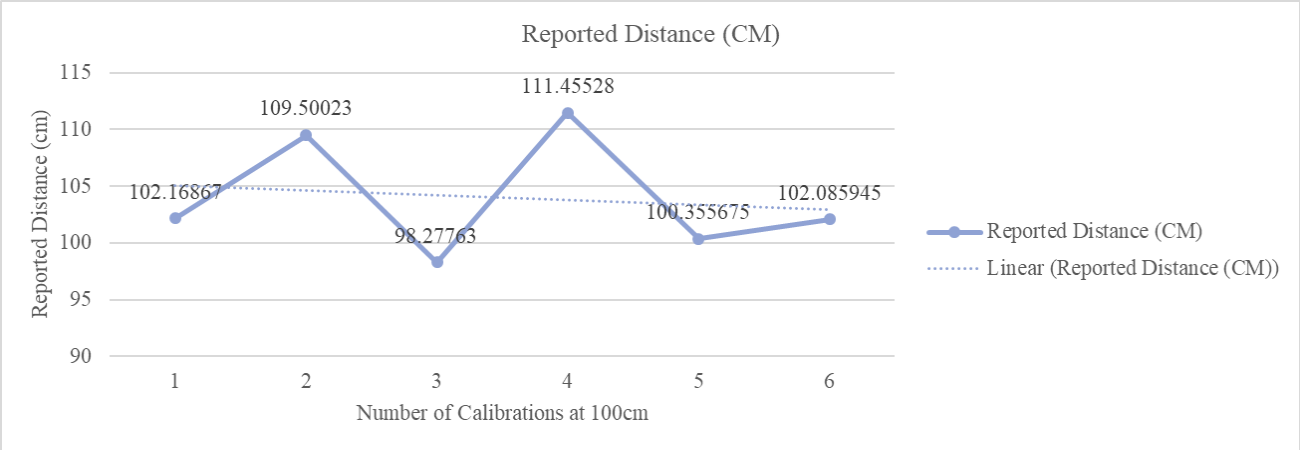


Figure 42. Graph of reported distances over N beacon calibrations

At 200cm, a result of 103.74788cm was returned for the static method.

Scan Distance (CM)	Reported Distance (CM)
150	138.51944
125	127.3456
100	106.06995
75	72.902596
50	55.367634
25	23.770672

Table 6. Dynamic beacon calibration results

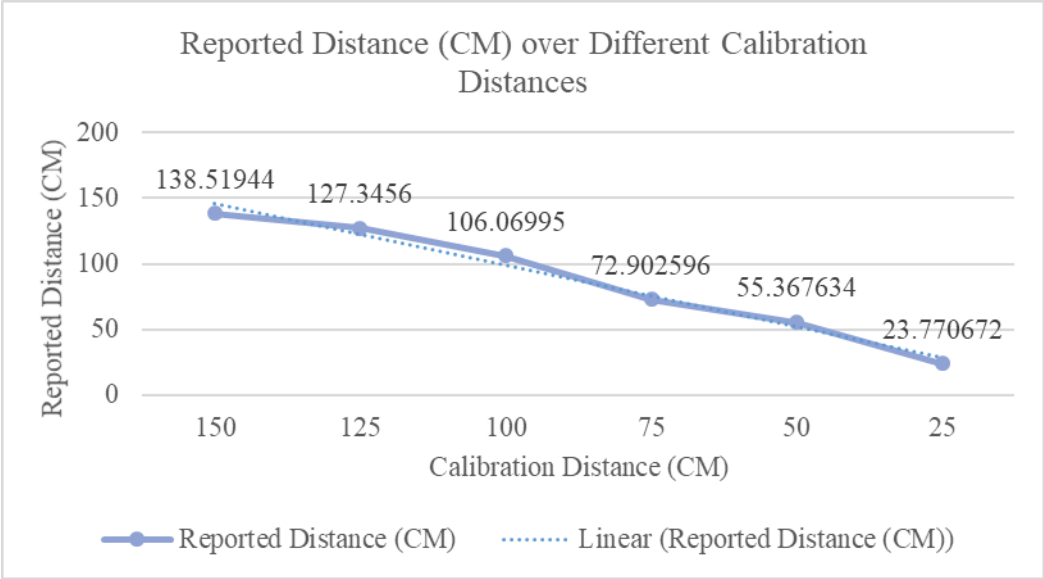


Figure 43. Graph of reported distances over beacon calibrations at (N=6) different distances

At 200cm, a result of 193.79432cm was returned for the dynamic method.

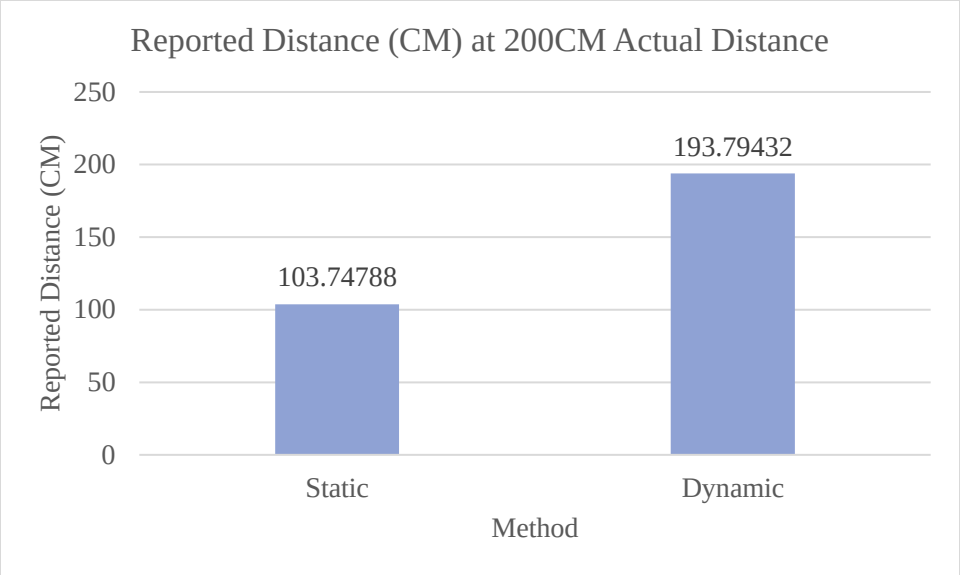


Figure 44. Differences in distance reporting at distance 200cm by method

12.3.1.12.3. Analysis

The static method proved again to be unreliable over the number of scans performed, the reported distance varying between 98 and 111cm, producing an average of 103.97cm, a similar error range suffered by previous static results. The spread is largely similar as well, showing a slightly tighter but otherwise negligibly-different standard deviation of 5.27cm compared with 5.48 experienced in the previous results, implying methodology is largely responsible for the variance, rather than the devices themselves. Although the average suggests an acceptable level of accuracy, the error range is still not optimal for accurate locational reports and may be exacerbated over longer/larger distances after calibration at smaller distances. However, the static method may become more accurate over a larger (N>6) calibrations. It can be observed from the static results that there is a slight linear decline trend shown, originating at ~105cm at N=1 calibrations and finalises at N=6 with ~103cm. It can therefore be predicted that over larger number of calibrations, a convergence to the calibration distance (100m) may be observed, although this would require extensive testing. The dynamic method shows the same promising trend as previously shown, with the reported distances almost exactly reflecting the related calibration distance with the average deviation from actual distance being just +0.17cm over all reports and the largest reported error range being just ~11cm. The distinct trend shows noticeable and accurate reports of distance as the host device is presented closer to the beacon, showing a much stronger trend over previous dynamic testing. This suggests that due to the controlled nature of the method, the large variations in signal strength exhibited by the beacon device are primarily responsible for the accuracy improvements. It is also notable that the error ranges experienced by the beacon reports do not suffer largeness to the same extent as those presented in the non-beacon test, especially at lower distances. However, the two devices report similar distances and subsequent error ranges at the largest calibration distance (150cm), a nuance that

may be replicated over a larger range of calibration distances. If experienced, the accuracy of reporting may be shown to degrade over both calibrated and uncalibrated distances, making the Bluetooth fingerprinting method less competitive against similar local fingerprinting strategies such as Wi-Fi.

12.3.1.12.4. Conclusion

As previously concluded, the dynamic calibration methodology consistently creates accurate results, with distance reports remaining within a small range of error. Furthermore, dynamic appears to scale more effectively at (N=6) varying distances on both devices tested. However, there are cases in which the inaccuracies produced by static fingerprinting may be mitigated, such as if the broadcaster(s) of the Bluetooth signal is/are in an open environment that the recipient Android device is likely to reside within. In these instances, a nominal to large level of inaccuracy (+-20cm) would not necessarily affect a user's ability to discern the location of their device. Therefore, use cases can feasibly be extracted from static calibration at low levels of calibration (N<7). Alternately, dynamic tends to familiarise the algorithm with the signal variations and behaviour of the device (broadcaster) over distance in order to scale better at varying ranges and to produce a more precise average dBm/CM average. Preferential for most, if not all, use cases. It could also be argued that the convergence seen within the beacon results suggests a greater level of calibration (N>6) would yield a tendency to the calibrated distance. Although, as this is not replicated within the earlier test it can be reasoned that the algorithm effectiveness is limited during signal fluctuations at a fixed distance. The reports created from both broadcasters examined present different usages and user preference. The precision of a beacon-oriented configuration outweighs that of a generic broadcaster such as the mobile phone tested. It is therefore a question of cost, the initial cost of beacons may be greater, however the alternative would be to staticize their current devices or maintain a map of their device positions to use as landmarks. The latter may only demand a minimal amount of investment; the same approach would also require a larger demand for user resources/devices that would experience repositioning, affecting reports. It is therefore preferable to employ a system of BLE beacons for maximised positional accuracy at the expense of cost.

12.3.2. *Wi-Fi Fingerprinting*

12.3.2.1. *Introduction to Wi-Fi Fingerprinting*

Fingerprinting is the process of creating a unique point of reference, characterised by the uniqueness of human fingerprints. Wi-Fi fingerprinting uses an approach based on the Received Signal Strength (RSS) transmitted by nearby Wi-Fi access points [46]. The RSSI (Received Signal Strength Indicator) is a measurement of the power that exists in a given radio broadcast [47]. The SI unit of RSS is expressed as dBm (decibels relative to one milliwatt). Within the Android OS the RSSI of a Wi-Fi device exists as an integer between -100 and 0, the larger, less negative value of a Wi-Fi broadcast's RSSI, the stronger the signal and the closer the broadcaster is from the device performing the reading. With this technology a position can be obtained using two common methods: location fingerprinting and trilateration. The former collects fingerprints of RSS distributions and estimates the user's location by matching online measurements with the closest predetermined location, and the latter uses a logarithmic function can be applied to obtain the relationship between the measured RSS and the range to the transmitter, i.e., the AP in case of Wi-Fi positioning [48]. This section introduces an algorithmic approach towards fingerprinting via the collection of multiple APs and their RSSs at each location to be used as a point of reference when identifying the whereabouts of a device.

12.3.2.2. *Methodology*

The approach that was taken when analysing what method to pursue for fingerprinting was twofold based on a simple specification: the accuracy of the results obtained must be equal to or better than conventional means, and secondly redundancy, to increase the reliability of feedback. The problem with conventional approaches to Wi-Fi fingerprinting is that generally the application relies on a single AP (Access Point). By using this method, although accurate, it presents a single point of failure. For example, if a user fingerprints a room in relation to a particular AP, many technical faults and changes could arise affecting the validity of reporting, e.g. SSID changing, signal dropping, reboots. To mitigate this, a "snapshot" method was devised. This involves taking several (N>0) Wi-Fi scans in a static area and averaging the signal (RSSI) of each AP discovered and saving the results in a structure along with the context (alias for the location) for later reference. Then, once a user requests a location fix, a Wi-Fi scan is performed, and the results compared against the results with each stored alias. For each matching AP found in both structures, the signal strength difference is recorded, after all matching APs (identified by SSID) are discovered, the alias receives a "score", the sum of all signal differences of matching APs. The alias (location) with the lowest score is then likely to be the accurate location of the device. Using this method, there is added redundancy gained by utilising more than one data source, and by attempting multiple scans, location accuracy is obtained, shown below.

12.3.2.3. *A Points Based Fingerprinting Approach*

Points based fingerprinting revolves around the observed variances in RSSI for APs. It uses these variations as an association to a particular location, for example if a device has a signal strength of 34dBm from access point AP in a room A and signal strength 47dBm from AP in room B, then a scan result that returns the device's RSS from AP being 35dBm has a closer proximity to the RSS of that experienced in room A (34dBm). Therefore, room A is the likeliest location of the device. The algorithm described within this paper performs a single scan, storing all APs discovered along with their RSSIs at that instance. All stored aliases from SharedPreferences are deserialised and iterated over, at each interval the list of scan results is searched, if an AP SSID found within a scan also exists within the alias' HashMap, the AP's RSSI encountered from the scan is subtracted from the stored averaged RSSI for that AP attained during the fingerprinting stage (*Stored AP_n RSSI – Scanned AP_n RSSI*). This is the score for that AP, the scores of all matching APs are added for an alias such that an alias' score is the summation of the signal differences of all matching APs that exist between the scan results and the alias' map of stored APs. The alias with the lowest sum therefore has the smallest total difference (variance) in signal strength over all matching APs and is therefore likely to be closer to that location.

$$Alias_{score} = \sum ((Alias\ AP_0 - Scan\ AP_0) + \ldots + (Alias\ AP_n - Scan\ AP_n))$$

Figure 45. Calculating the "score" for an alias with the stored average dBm for APs and their currently scanned dBm readings

12.3.2.4. *Worked Example*

Alias: Bathroom	AP1	60 dBm
	AP2	30 dBm
	AP3	15 dBm
	AP4	90 dBm

Alias: Bedroom	AP1	54 dBm
	AP2	22 dBm
	AP3	10 dBm
	AP4	78 dBm

Figure 46. Anatomy of snapshots (fingerprints) with alias and map of APs with their averaged signals

Assuming the user has created the two fingerprints present and has requested a location with the device residing in an unknown location, the initial scan may return the following results:

AP1	56 dBm
AP2	24 dBm
AP3	11 dBm
AP4	80 dBm

Figure 47. The result of a Wi-Fi scan, filtered by AP SSID and averaged RSSI

The comparison algorithm is then run with the scan results and existing fingerprints to assess the signal variances between the stored RSSI for an AP and the AP's RSSI returned in the scan to calculate scores for the two fingerprints:

Matching AP	Bedroom	Bathroom
AP1	+2 dBm	-6 dBm
AP2	-2 dBm	+8 dBm
AP3	+1 dBm	-5 dBm
AP4	-2 dBm	+12 dBm
	Total: 7 points	Total: 31 points

Table 7. The calculated scores of two stored aliases when compared against an initial scan

The algorithm therefore determines that the device is likely in the bedroom as:

$$\text{Bedroom} \sum(\textit{signal variances}) < \text{Bathroom} \sum(\textit{signal variances}).$$

12.3.2.5. *Finger Printer Implementation*

The initial process of taking a "fingerprint" entails taking numerous (N>0) Wi-Fi scans whilst in a static location with a user-provided alias, a memorable word or phrase for the location. The results of a scan are combined after every sequential scan with pre-existing results to maintain an average RSSI for each AP over N scans. Once a Wi-Fi manager object has been initialised, a scan API call is executed to start the process. To catch the result of a Wi-Fi scan, a Broadcast Receiver is registered with the required intent that's fired by the OS once a scan has been completed.

```
registerReceiver(wifiReceiver, new
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
wifiMan.startScan();

private final BroadcastReceiver wifiReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        adder(wifiMan.getScanResults());
    }
};
```

Figure 48. Registering of a receiver for scan results

At the end of each scan, a `List<ScanResult>` object is received that contains the details of each AP found. This object is then passed to the combinatorial function in figure X to merge with existing results. For storage, a `HashMap<String, Integer>` object is utilised to hold SSID-Signal Strength pairs found during a scan. The HashMap data structure is used here, the statistical properties of hash maps are well documented, particularly its aptitude in searching in constant time, a time order of O(1) [49]. Due to its speed of search, it enables for the fast searching of the map for existing RSSI values for a given SSID. If a valid RSSI is returned using an SSID as a key using the public method `get()`, the existing RSSI is averaged with the current value from the scan and the SSID-RSSI pair written back to the HashMap. If no value is returned (RSSI Integer value is null), the SSID is added along with its scanned signal strength to the data structure as a new entry.

```
private void adder(List<ScanResult> wifiList){
    if(SCANS != 0){//if there's scans left
        for(ScanResult scanResult: wifiList){//iterate over scan results
            Integer RSSI = wifiHashMap.get(scanResult.SSID);//get current RRSI for
that SSID
            if(RSSI != null){//if already in hashmap
                wifiHashMap.put(scanResult.SSID, ((RSSI + scanResult.level) / 2));
//calculate average and overwrite previous value with key
            }else{
                wifiHashMap.put(scanResult.SSID, scanResult.level);//else add it to
the list as a new AP
            }
        }
        SCANS--;//decrement scans left
        wifiMan.startScan();//restart scan
    }else{
        save();//finally save
    }
}
```

Figure 49. Adding SSID and RSSI of each AP to list, if already existent the RSSI is averaged with the new, current value

Once all scans have been completed, the saving phase begins. This involves deserializing the existing list of fingerprints from SharedPreferences such that the new fingerprint can be

added and the structure re-serialised and saved. To store a fingerprint, Android's SharedPreferences is used. SharedPreferences allows the "putting" and "getting" and deletion of various datatypes such as Strings and Integers. This is preferable to primitive file IO, in both speed and the usage of code. To effectively store the fingerprint, the list of fingerprints structure: `ArrayList<Pair<String, HashMap<String, Integer>>>` (a list of pairs of aliases (String) and AP-RSSI maps (HashMap)) must be converted into an acceptable format. To do this, the additional Google Gson library is used to serialise the Java object into a JSON formatted string for storage. A check is made such that existing fingerprints remain untouched by identifying whether the structure already exists within storage by using the designated key to return the serialised string. If so (the string is not null), the string is deserialised into the appropriate structure before the fingerprint is then added as a new pair, the alias (context) and the HashMap of SSIDs and averaged RSSIs. The newly modified structure is then serialised and written back to preferences.

```
private void save(){
    ArrayList<Pair<String, HashMap<String, Integer>>> toStore;//store alias along
with a hashmap of SSIDs and averaged signal strengths
    SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
    String doesExist = sharedPreferences.getString("WIFI_PRINTS", null);

    if(doesExist != null){ //if the structure exists, deserialize
        toStore = new Gson().fromJson(doesExist, new
TypeToken<ArrayList<Pair<String, HashMap<String, Integer>>>>() {}.getType());
//deserialize
        toStore.add(new Pair<>(aliasString, wifiHashMap));//add pair
    }else{//if null (non-existing)
        toStore = new ArrayList<>(); //create a new arraylist
        toStore.add(new Pair<>(aliasString, wifiHashMap));//add pair
    }
    //serialise and save into shared preferences
    sharedPreferences.edit().putString("WIFI_PRINTS", new
         Gson().toJson(toStore)).apply(); //write back to storage
}
```

Figure 50. Saving a fingerprint alias and the list of discovered APs with their averaged RSSI readings

12.3.2.6. *Scanner Implementation*

The scanner class is a helper class with the sole purpose of performing a AP scan, reasoning and calculating scores and returning the likeliest fingerprint location of the device as a string. This class is used to return the alias as a string once the getter method `getResults()` is executed, the class performs numerous checks, such as if the device has Wi-Fi capabilities and that fingerprints exist, else the process will cancel with an exit message. This is an important step to prevent wasted computation time. After the Wi-Fi scan is performed, the method then enters an empty while loop with a boolean flag for the duration of the scan to prevent the method from being

completed until the flag is set to true (process has finished). This was the most important addition to prevent premature termination as WiFiManager's `startScan()` method is non-blocking in regards to the main thread and executes in the background. Once the `SCAN_RESULTS_AVAILABLE_ACTION` intent is fired, the list of results (*List<ScanResult>*) is extracted and passed as an argument to the comparison method.

The method necessitated an additional data structure to hold the alias and the calculated score for that alias. This was chosen to be an ArrayList of pairs for ease of sorting the pairs by value (score). Once the fingerprints data structure is deserialised the individual aliases are iterated over. Within this loop, the scan result list is iterated in an inner loop, at each interval searching the alias' HashMap for the SSID of the current scan result object. Once an equivalent SSID is found, the signal from the scan is deducted from the alias' stored signal strength for that AP. This gives an integer difference between the two results. The difference is then added to a temporary counter for that alias and a Boolean flag is set to true, such that a fingerprint with no matching APs in reference to the scan results is not added to the list and consequently not considered viable and discarded from further consideration. After all cross-referencing is performed, the aforementioned points list will contain the aliases of all fingerprints that have at least one matching AP in relation to the scan, along with the score for that alias. If empty, there are no suitable aliases for the device's location and the process terminates. The points list is then sorted in ascending order by the integer score present in the pair's value (*pair.second*), such that the alias stored in the last index (*list.size()-1*) has the lowest score, and is therefore set as the return string and returned to the calling method by setting the finish flag to true.

```
private void compare(List<ScanResult> results){
    final ArrayList<Pair<String, Integer>> points = new ArrayList<>(); //store
aliases and score
    for(Pair<String, HashMap<String, Integer>> alias: fromPrefs){ //loop through
aliases
        int temp = 0;//initialise score
        boolean hasFound = false;
        for(ScanResult scanned: results){//loop through scan results
            if(alias.second.containsKey(scanned.SSID)){//if the SSID exists in the
alias' map
                temp += Math.abs(alias.second.get(scanned.SSID) - scanned.level);
//add the difference in signal to the score
                hasFound = true;
            }
        }
        if(hasFound){
            points.add(new Pair<>(alias.first, temp)); //if at least one matched AP
was found, add to score map
        }
    }
    if(points.isEmpty()){
        finish("No Saved Points");
    }else{
        Collections.sort(points, (o1, o2) -> o1.second - o2.second);//sort
ascending
        finish("Alias: " + points.get(points.size()-1).first);
         //pick alias of last entry
    }
}
```

Figure 51. The Wi-Fi fingerprint scoring algorithm

12.3.2.7. *Testing*

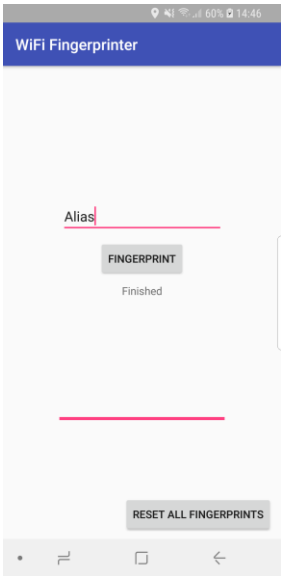


Figure 52. Fingerprinter GUI

For testing, the GUI was used to take (N=1…13) readings (calibrations) in two adjacent rooms, the device was then positioned in one of the two rooms randomly and the accuracy of ten responses received from location requests were noted.

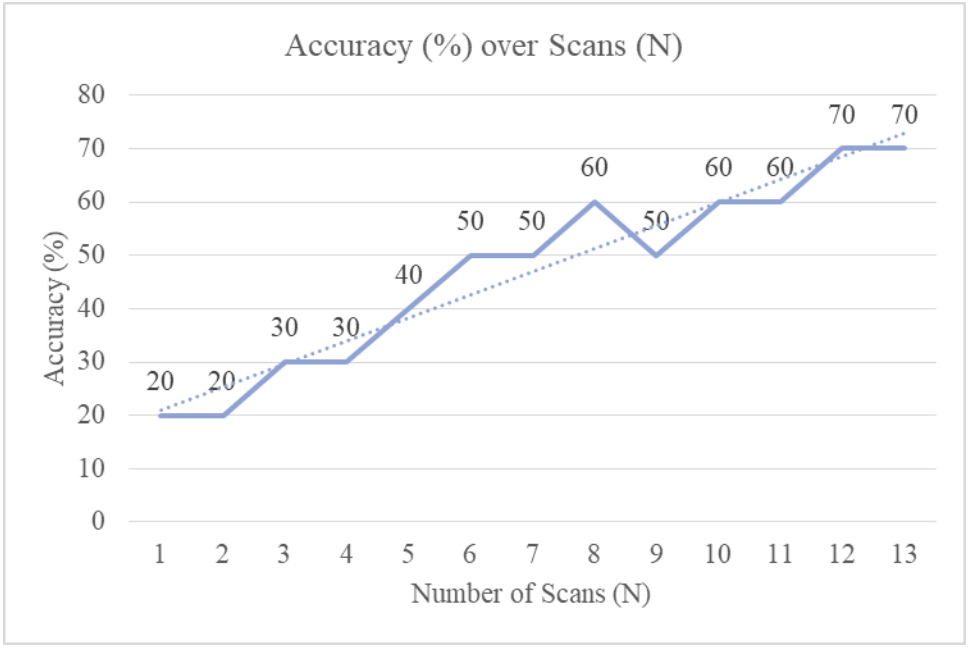


Figure 53. Chart showing the accuracy of fingerprinting in relation to scans performed

The inference that can be made from the data is that the number of scans performed is largely proportional to the accuracy of the data returned. The advantage of multiple scans tapers at N=12, and plateaus thereafter. Many conclusions can be drawn from this data, the lower levels of scanning understandably do not adequately capture the mean of variances needed to distinguish between the two fingerprints in this test. A large increase is noted between N=4 and N=6 followed by another plateau, at this point the algorithm suggests that the averages effectively reach an equilibrium, this trend is unlikely to be replicated in different indoor environments and layouts and should only be present in rooms with negligible distance. The same trend is not seen again whilst N>7 until an apex at N=12, whereafter there is a stagnation in accuracy. This chart ultimately suggests then that N=12 scans is optimal for the most accurate location fix in a compact layout, it can then be inferred that greater accuracy may be available in more sparse environments.

As the graph displays a gradual increase in accuracy over successive calibrations, the natural progression was to test at larger numbers of calibrations. Based on the behaviour seen in the initial testing the number of additional calibrations needed to observe a change in accuracy that will converge on one hundred per cent would be between N=4 and N=6. Hence, the calibrations of N=14 to N=20 and the returned accuracies were taken and noted.

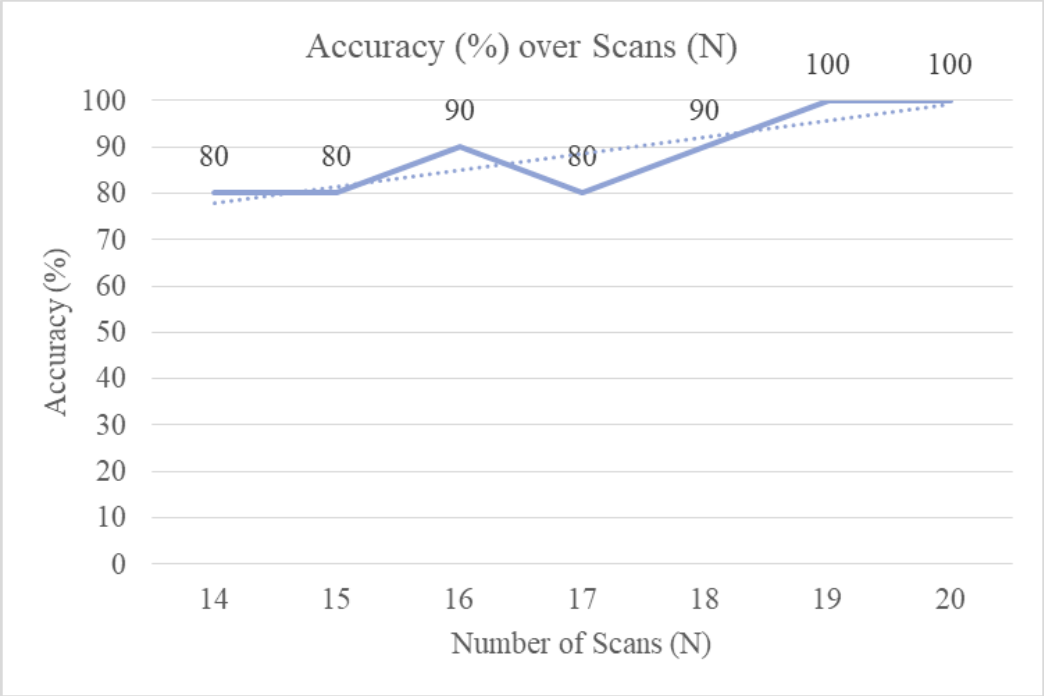


Figure 54. Chart showing the accuracy of fingerprinting in relation to scans performed (N=14 to N=20)

The extra calibrations were shown to be largely successful in the context of this experiment. The 80% plateau seen between N=14 and N=15 is a trend seen in prior testing that

appears to always be mitigated by further calibrations. This additional testing proves to be in keeping; there is a slight deviation at N=16 which is later repeated at N=18 after suffering a dip to 80% accuracy from 90% at N=16, this is presumably due to subtle deviations in signal strength across APs impacting largely on the scoring of each alias due to the close proximity of where the two aliases were created and calibrated. It can be reasonably assumed that the contrast of connectivity strength experienced in two more distant locations would result in less uncertainty shown in the graph and thus would tend and converge more strongly upon one hundred per cent. However, the accuracy at N=19 and N=20 defines the pinnacle of the observed results, correctly reporting the room it resided within ten out of ten times, giving a one hundred per cent success rate. In similar scenarios we can now ascertain that the optimal number of scans within each room is between N=19 and N=20. Further testing could reveal adverse or stronger results; this test is also not representative of all use cases that an end user may encounter and the results would understandably differ with the context in which this method is applied.

12.3.2.8. *Caveats*

There are systematic flaws with this method of fingerprinting, the most prominent is the reliance on multiple APs and the inherent need to make multiple fingerprints to improve accuracy, alike conventional approaches. For example, if a user performs a location request with a single stored fingerprint and at least one matching AP is found, the fingerprint's alias will logically always be returned in the request, giving a potentially incorrect location fix. It is also worth noting that although many APs remain static (stationary) if one or more APs disappear or change SSID after a fingerprint is made and saved, the scanner will therefore detect fewer matching APs on a new scan and will subsequently produce unreliable results and a potentially a degradation in accuracy. It is therefore necessary to inform users of the nuances of the method before it is utilised as an effective means of geolocation. The glaring fault with method is the taxation on the end user; the efficacy of this method is largely dependent of the environment, whereas it may perform well in adjacent rooms, another layout may require an extensive number of scans to become an accurate means of indoor localisation. It is therefore the duty of the user themselves to test the accuracy of location reports with a varied number of scans for each scenario.

13. Online Database of User Location Data

In engineering, redundancy is the duplication of components or functionality of a system with the intention of increasing reliability [50]. The purpose of the online database is to act as a failsafe system, storing user device information, such as IMEI, that can be used by law enforcement, as well as the latest location coordinates, date the location was declared, the accuracy of the location fix in metres, and the required usernames and hashed passwords. It was designed around added redundancy to increase the chances of locational data being made available to the user without explicit intervention through triggering. With conventional triggering, the user is relying on the lost or stolen device having that specific data connection at that instance. For example, if the user sends a GC message, the device will have to have an internet connection at that given instance for the communication to be successful and timely. The online database was designed to mitigate this by enabling the Android application to attempt "phoning home" [51], contacting the database silently in the background, updating the user's database entry with new location coordinates. This occurs when certain intents are received within the Android OS that would suggest a data connection is available, and other frequently broadcasted intents for consistent updating. With this accompaniment to the application, the user need not necessarily use triggers, as the device location is updated frequently on the database, bypassing the need to contact the device through other means. The user's information within the database can then be queried directly through the web interface by using the "Login" and "Get Device Info" functions respectively.

13.1. *Database Format*

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**device**	longtext	latin1_swedish_ci		No	*None*		
2	**user**	varchar(12)	latin1_swedish_ci		No	*None*		
3	**pass**	varchar(64)	latin1_swedish_ci		No	*None*		
4	**date**	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP		ON UPDATE CURRENT_TIMESTAMP
5	**latitude**	double			No	0		
6	**longitude**	double			No	0		
7	**accuracy**	float			No	*None*		

Figure 55. Database structure

The data types and attributes used within the database reflects those within the Android OS. The username field is, both within the application and the database, limited to twelve characters, this provides a huge number of available usernames while limiting the size of the database. The password length, however, is determined by the size of the hash generated, using the default hashing algorithm will produce a hash of 60 characters length [52]. The varchar type that stores character data needs to be of sufficient length to accommodate the hash length, failure to do so causes the hash verification function (`password_verify()`) being performed on an incomplete, truncated hash, resulting in recurring failed verifications when updating and retrieving information from the database. For accurate storage of longitude, latitude and accuracy values,

the fields must be of the same type as within the application. The Android operating system returns the latitude and longitude, and accuracy of a Location object as double and floating-point objects respectively. The double type is necessitated due to the format of coordinates, double has a precision of 15 digits, compared to for example, float, with 7 digits of precision [53]. It would therefore be illogical to use a data type with lower levels of precision that would truncate the longitude and latitude values, corrupting the true location data from the Android application. Finally, the date column uses "`on update CURRENT_TIMESTAMP`" and "`CURRENT_TIMESTAMP`" attributes to store the time stamp of the most recent change or update operation performed, providing context to the user regarding the time of the latest location update.

13.2. *Registration*

The sign-up activity presents the user with a simple username, password form and a submit button. The PHP registration script takes three parameters, a JSON-formatted string of device information "device", a username "user" and a password "pass". This is preliminary information that remains unchanged/static after the initial sign up process. Like other processes that rely on accessing external scripts, the sign-up activity performs a background AsyncTask that reads the registration script's current URL from a text file hosted online. Once this is retrieved, a JSON-formatted string object is created containing the model, device, and manufacturer of the Android device. The JSON format makes this data easier to parse when requested by the web interface. If this JSON object fails to be created, an error message will take its place as the "device" URL parameter. This enables the account to be created successfully regardless of the error. Finally, the URL is modified to contain the device string and the username and password input provided by the user as the "device", "user" and "pass" parameters respectively. Once a connection is opened to the URL, each line of the connection's output is read and appended using the `BufferedReader` class. o`nPostExecute`, the method called when the background task has completed, then receives the string of the connection's output. It is then passed, along with the username and password, as arguments to the `showDialog` method. This method scans the string for a keyword, namely "`USER_REGISTRATION_SUCCESS`" that is outputted or "echoed" by the script only when the user was created successfully and entered into the database, seen below. If the keyword is present, a confirmation dialog is presented, the username and password are then stored in SharedPreferences for access in updater methods and associated broadcast receivers. If not present, the output string is printed explicitly to the user in a dialog, which will contain the error message printed by the script.

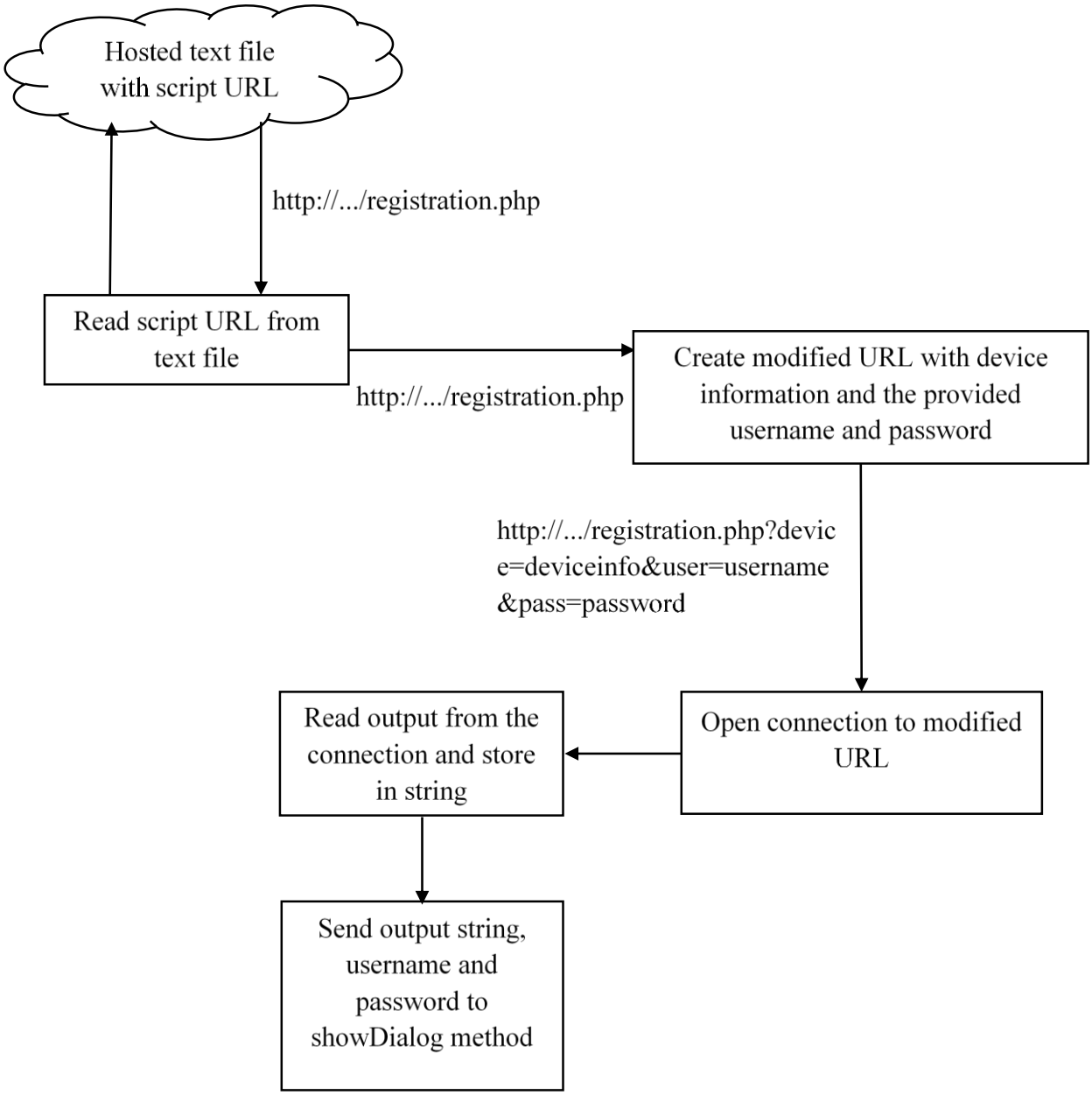


Figure 56. The user registration background task process

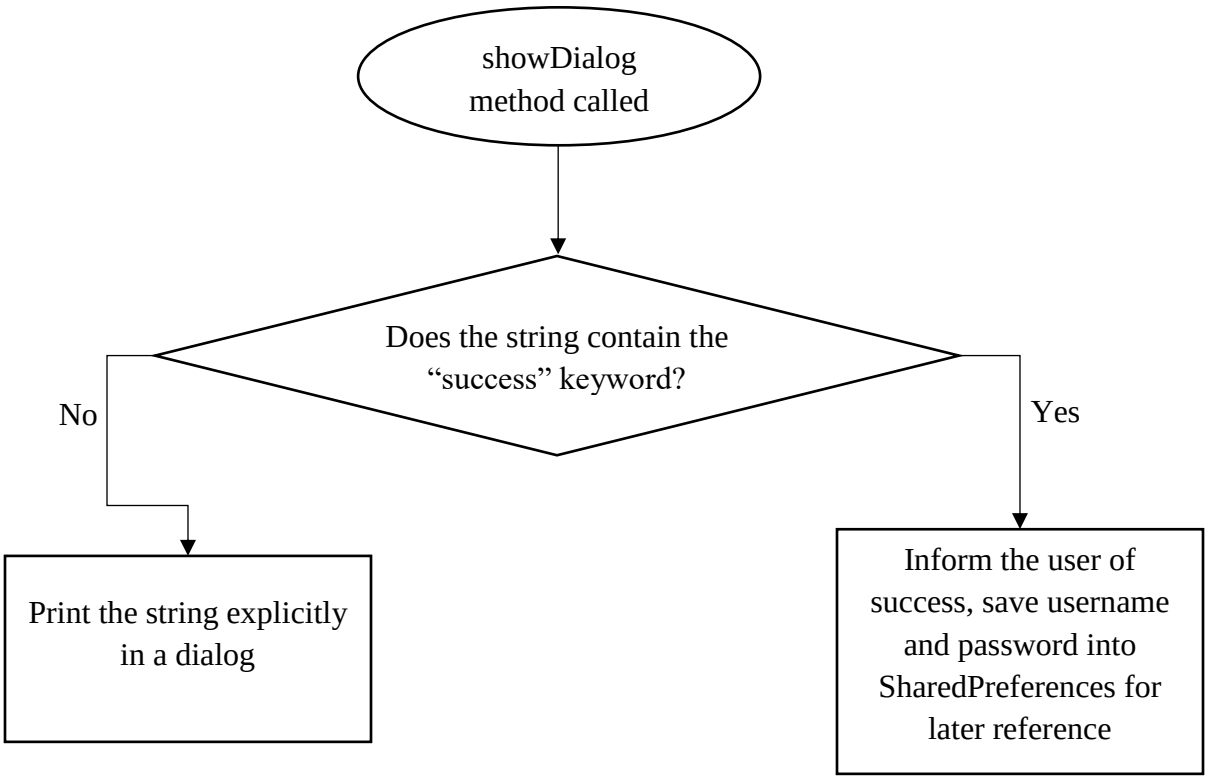


Figure 57. Scanning the connection’s output and giving feedback to the user in the showDialog method.

The registration script itself performs multiple actions to ensure the uniqueness of users, the safe storage of passwords and the elimination of database redundancy. It asserts the following: each username must be unique, passwords must be stored in a secure, hashed format, and finally, the result of each important action performed within the script must be printed, such that the Android application receives this information in the connection’s output and can present an informative dialog to the user. The first step is to check that the required parameters are there. This is done by assigning them to variables and only continuing after all variables are verified to be not empty. This is the most important step in preventing data corruption caused by incomplete data being entered onto the database. The following steps include creating a connection to the server and selecting the required database through SQL functions. If the initial connection returns an error, the script will print the error using “`echo`” or “`printf`” and will quit using the `exit();` function to prevent unnecessary further execution, likewise with the database selection process. After a connection to the database is established, a simple SQL query is performed on the database’s table:

```
"SELECT * FROM $tbl_name WHERE user='$user'"
```

This query searches the “user” column of the table using the username passed to the script as a parameter, this is then stored in a variable for analysis. It then follows that if the

number of rows in that query result exceeds zero, then a user by that name already exists and thus the user should be informed that the name is unavailable for use. If the username is declared available, the password parameter variable is then hashed using PHP's `password_hash()`, creating a new password hash using a strong one-way hashing algorithm [3]. This prevents password data theft if the database were to become compromised. Although the data contained is not particularly insightful or confidential, it's best practice to store user's passwords in their hashed form that can then be verified against the original password as an authentication measure when a data request or update is performed. The user is created by an `INSERT_INTO` SQL query performed on the table using the device parameter variable, username parameter variable, and the newly formed hashed password variable.

```
"INSERT INTO `$tbl_name` (device, user, pass) VALUES
          ('$device','$user','$hashed')"
```

Figure 58. Insertion of a new user into the database via an SQL query

```
<?php
    $device = $_GET["device"]; //get device info string, username, and password from URL
parameters
    $user = $_GET["user"];
    $pass = $_GET["pass"];

       if(!empty($device) && !empty($user) && !empty($pass)){//make sure they're all not
empty (valid)
               $host="ft025024.webs.sse.reading.ac.uk"; // Host name
               $username="ft025024"; // MySQL username
               $password="passwordhere"; // MySQL password
               $db_name="ft025024_project"; // Database name
               $tbl_name="lookup"; // Table name
               //Create connection to server as a new variable, else print error message.
               $connection = mysqli_connect("$host", "$username", "$password")or
die("FAILURE : Cannot connect");
        if (mysqli_connect_errno()) { //if error occurred
            printf("Connect failed: %s\n", mysqli_connect_error()); //print error
            exit();//exit, nothing else can continue
        }

        if(!mysqli_select_db($connection, "$db_name")){//select the database via the
connection
            echo "FAILURE : Cannot select DB"; //if database cannot be selected
            exit();//quit
        }
        $result = mysqli_query($connection, "SELECT * FROM $tbl_name WHERE user='$user'");
//get the result of user name lookup, store in variable

        if($result){ //if the query hasn't failed
            if(mysqli_num_rows($result) > 0){ //calculate rows from query, if greater than
0, a user by that name already exists
                echo "FAILURE : An account with that username already exists"; //already
registered, print feedback
            }else{ //if there's no-one by that name...create the user
                $hashed = password_hash($pass, PASSWORD_DEFAULT); //hash the password
given , (use password_verify on other end)
                $query = "INSERT INTO `$tbl_name` (device, user, pass) VALUES
('$device','$user','$hashed')"; //insert new user info
                if(mysqli_query($connection, $query)){ //perform the query
                    echo "SUCCESS : User created successfully. USER_REGISTRATION_SUCCESS";
//query has succeeded, print message with special token
                }else{
                    echo "FAILURE : User registration failed";//failure
                }
            }
        }else{
            echo "FAILURE : Unable to retrieve results from table";
            exit();
        }
       }else{
       echo "FAILURE : Incorrect params";//can't get the necessary params for creation
       exit();
    }
?>
```

Figure 59. Registration PHP script

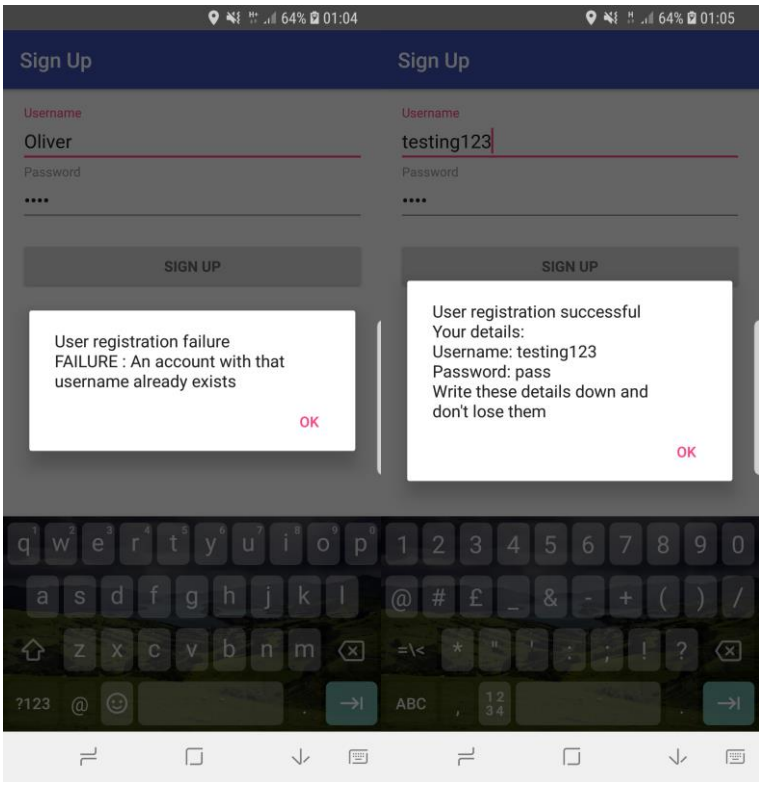


Figure 60. Android dialogs based on connection response string keyword

13.3. *Updating User Database Entries with Location Data*

Updating applies many of the same processes of other scripts such as performing the connection to the database and selecting the necessary table, but the queries performed differ. The script takes five parameters: the username and password for validation and the latitude, longitude, and accuracy of the location fix. Again, before any processing is performed, the script determines whether all parameters are present, the table is then queried for the username provided. If a result is present, the user's database entry is retrieved via an SQL function call "`mysqli_fetch_row()`" which returns the row of the result as an enumerated array [54] such that the length of the array is equal to the number of columns in the row. This is where the authentication step is performed, the password parameter (`$pass`) is validated against the stored, hashed password which resides in the third column of any database row, and thus in the third position of the fetched row array e.g. (`$row[2]`).

```
password_verify($pass, $row[2]);
```

`Password_verify()` is a PHP function that verifies that the given password matches a hash and returns a Boolean value, returning true if the password and hash match, or false otherwise [55]. Therefore, if the function returns true, the user has been successfully

authenticated and the script can proceed in updating the user's row with the updated location information.

```
"UPDATE $tbl_name SET latitude = '$lat', longitude = '$lon', accuracy =
                    '$range' WHERE user='$user'"
```

The final query updates the table with the set latitude, longitude, and accuracy variable values where the user exists. Due to the data redundancy measures including enforcing uniqueness of usernames within the registration script, this query will always correctly update the single row of user data only.

```
<?php
    $user = $_GET["user"];
    $pass = $_GET["pass"];//get user name and password for authentication
    $lat = $_GET["lat"];//and the location details
    $lon = $_GET["lon"];
    $range = $_GET["acc"];

       if(!empty($user) && !empty($pass) && !empty($lat) && !empty($lon) &&
!empty($range)){ //check all fields are not empty
               $host="ft025024.webs.sse.reading.ac.uk"; //Host name
               $username="ft025024"; //Mysql username
               $password="passwordexample"; //Mysql password
               $db_name="ft025024_project"; //Database name
               $tbl_name="lookup"; //Table name

               $connection = mysqli_connect("$host", "$username", "$password")or
die("FAILURE : Cannot connect"); //connect to server
        if (mysqli_connect_errno()) { //if there's an error
            printf("Connect failed: %s\n", mysqli_connect_error()); //print error and exit
            exit();
        }
               //select the database with existing connection
        if(!mysqli_select_db($connection, "$db_name")){
            echo "FAILURE : Cannot select DB";  //cannot get database
            exit();
        }

               //get the user from the table
        $result = mysqli_query($connection, "SELECT * FROM $tbl_name WHERE user='$user'");
//perform the query, assign result to variable

               if($result){//if the result is true (a user by that name exists)
                       $row=mysqli_fetch_row($result); //fetch the row from the query
                       if(password_verify($pass, $row[2])){ //verify the given password
against the stored hash
                                //update the row with the new location data
                                if(mysqli_query($connection, "UPDATE $tbl_name SET latitude
= '$lat', longitude = '$lon', accuracy = '$range' WHERE user='$user'")){ //perform the
query
                                        echo "SUCCESSFUL UPDATE";//print success message
                                }else{
                                        echo "FAILURE UPDATING: " .
mysqli_error($connection);//print query error
                                }
                       }else{
                                echo "FAILURE UPDATING: PASSWORD VALIDATION
FAILED";//incorrect password parameter given, does not match hash
                       }
               }else{
                       echo "USER DOES NOT EXIST"; //The user search query returned 0 rows
               }
       }else{
        echo "FAILURE : Incorrect params"; //not all parameters were given
        exit();
    }
?>
```

Figure 61. *update.php* PHP script

Client-side, the Android application performs a similar fetch and resolve URL process to other networking operations. The user can configure two preferences regarding the transmission of location data to the database: update on a battery status change and update on a connectivity

change. The former takes advantage of the existing broadcast receiver utilised for the purposes of email triggering, the latter is achieved through a separate broadcast receiver that encompasses network-specific intent broadcasts, such as `STATE_CHANGE`, fired whenever the device connects/disconnects to a Wi-Fi network, and `CONNECTIVITY_CHANGE` for mobile network connection broadcasts. Although the second intent, when fired, is likely to result in a successful update due to its data connection-oriented nature, the "on-battery-change" preference is also recommended as it will result in more consistent connection attempts due to the frequency of the intent, increasing the chances of an update.

```
<receiver
    android:name=".SimStateChangedReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter android:priority="999">
        <action android:name="android.intent.action.SIM_STATE_CHANGED" />
    </intent-filter>
</receiver>
```

Figure 62. "Updater" broadcast receiver responsible for causing database update requests on receipt of a connectivity-based intent

The broadcast receiver then uses the `LocationService` class with the `getLoc()` method call to obtain a new `Location` object, a new `UpdateDatabase` object is initialised with the location object and a context for SharedPreference searching, and a method call to `update()` is called. This performs all networking operations in the location updating process. The update method applies a variety of checks to prevent any kind of data redundancy server-side, and consults SharedPreference preferences to determine whether an update should be executed. The most important check is to prevent unnecessary network traffic by ensuring the that username and password strings contained within SharedPreferences are not null (unset), implying that an account does not exist, along with the checking of the `Location` object for nullity, caused by the `LocationService` class being unable to return a valid location. This prevents the networking task from passing null values as parameters to the update URL, resulting in unnecessary server-side processing. The settings GUI also gives the user a level of control of this network process, giving the user the option to only update their location when connected to a Wi-Fi network, reducing data usage. This is checked before every update.

```
/**
 * This class performs an update request upon receipt of a connectivity intent
(if below Android 7.0)
 */
public class Updater extends BroadcastReceiver {
    @Override
    public void onReceive(Context c, Intent intent) {
        if(intent.getAction() != null) {//check the intent is valid

if(PreferenceManager.getDefaultSharedPreferences(c).getBoolean("status_update",
                new UpdateDatabase(new LocationService(c).getLoc(), c).update();
                 //if user has enabled feature, update database
            }
        }
    }
}
```

Figure 63. "Updater" BroadcastReceiver

The `UpdateDatabase` class utilises an AsyncTask inner class that performs the same fetch-URL operation that's adopted for many other server-based communications to collect the current URL of the PHP script responsible for updating user data. Once retrieved, the URL is modified with the username, password, latitude, longitude, accuracy, passed as "user", "pass", "lat", "lon", "acc" parameters respectively.

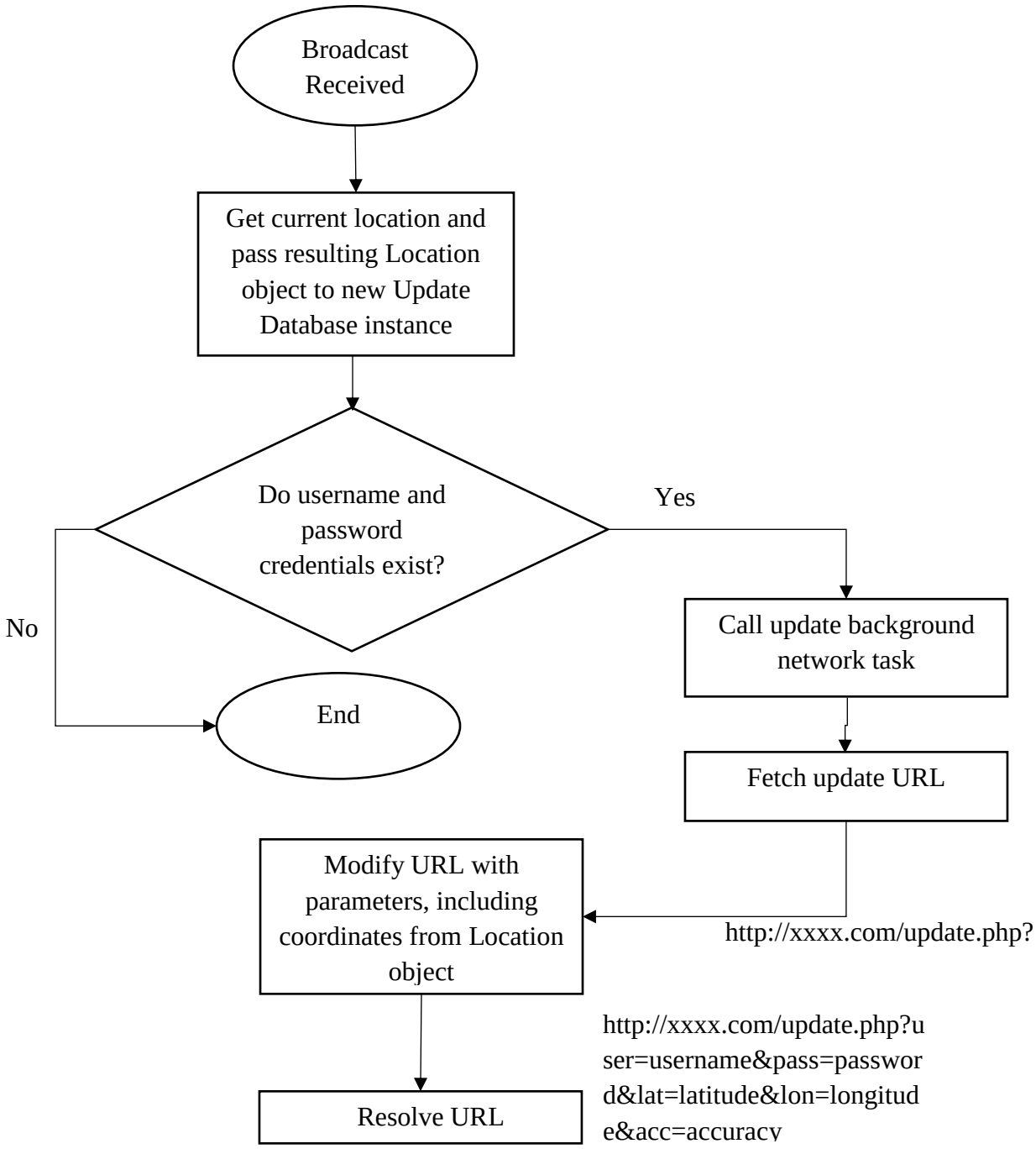


Figure 64. The background update process

A notable caveat of this process is in its design. As the update must be executed silently in the background, and with no user interaction for maximum effectiveness, it eliminates the potential for error reporting and meaningful feedback to the user generated from the connection's

output. Issues such as failed connection attempts by the application will subsequently be ignored, although this is desired behaviour for an error-resilient application of this kind, it may result in a waste of user bandwidth. An alternative to this approach would be to create a mechanism of printing error messages or notifications on a connection or update failure. However, in places of limited connectivity, the number of error reports would constitute spam.

```
StringBuilder sb = new StringBuilder();
BufferedReader br = new BufferedReader(new InputStreamReader(new
BufferedInputStream((new
URL(c.getString(R.string.update_redirect)).openConnection()).getInputStream())))
;
String inputLine;//read current update URL from hosted text file
while ((inputLine = br.readLine()) != null) {
    sb.append(inputLine);//read output
}
br.close();
//append parameters to update URL and resolve
String fullURL = sb.toString().trim() + "user=" + username + "&pass=" + password
+ "&lat=" + loc.getLatitude() + "&lon=" + loc.getLongitude() + "&acc=" +
loc.getAccuracy();
BufferedReader read = new BufferedReader(new InputStreamReader(new
BufferedInputStream((new URL(fullURL).openConnection()).getInputStream())));
read.close();//finally close
```

Figure 65. Update networking operation

13.4. *Retrieving User Data*

Retrieving data relies on the same password hash verification measure, returning a delimited string with the protected character '|' and consisting of all row contents. The script accepts two parameters: the username and password of a user. The script performs a lookup for the username, if identified, the row is fetched and the password parameter validated against the stored hash. If authentication succeeds, the desired array contents from the row variable are concatenated with the delimiter into a single string that is then printed, allowing the web application to parse the information correctly from the script's output.

```
if(password_verify($pass, $row[2])){ //verify the password against the hashed password
        echo $row[0] . "|" . $row[3] . "|" . $row[4] . "|" . $row[5] . "|" . $row[6];
}else{ //return row contents if the password parameter is correct
        echo "Cannot verify password";//else print error message
}
```

Figure 66. Printing concatenated string of fetched row array contents using a delimiter

On the web client, the user is presented with two options related to the retrieval of data: 'Login' which saves username and password input from the user to local storage, and 'Get Device Info' which uses the saved credentials as parameters to the script URL, displaying an informative dialog after parsing the information from the connection's output.

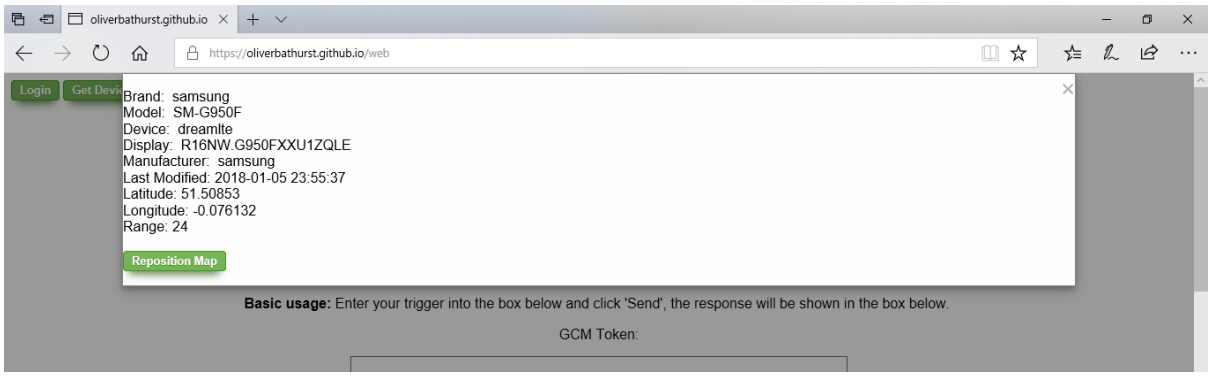


Figure 67. Web interface displaying dialog of information parsed from database

The request is performed using the jQuery JavaScript library within a JavaScript function *getDeviceJSON()*, called upon the 'Get Device Info' button event.

```
function getDeviceJSON(){
    if(localStorage.username !== null && localStorage.password !== null){
        if(read_row !== null) {
            var url = read_row + 'user=' + localStorage.username + '&pass=' +
localStorage.password;
            $.get(url, function (data) {
                var array = data.split('|');
                displayModal(array[0], array[1], array[2], array[3], array[4]);
            });
        }else{
            alert("Cannot read the fetch row URL");
        }
    }else{
        login();
    }
}
```

Figure 68. Fetching a user record via a jQuery *get()* call performed upon a PHP script URL

Firstly, the username and password are retrieved from local storage, stored via the 'Login' function, these parameters are then concatenated with the script URL to be resolved. Here, jQuery's *get()* function, which is used to load data from the server using a HTTP GET request [56], the result of the request is then stored within an object "`data`". This object contains the entire connection response from the GET request, this string can then be split using the specified delimiter into an array of strings and each string presented as a different section within the dialog or "modal".

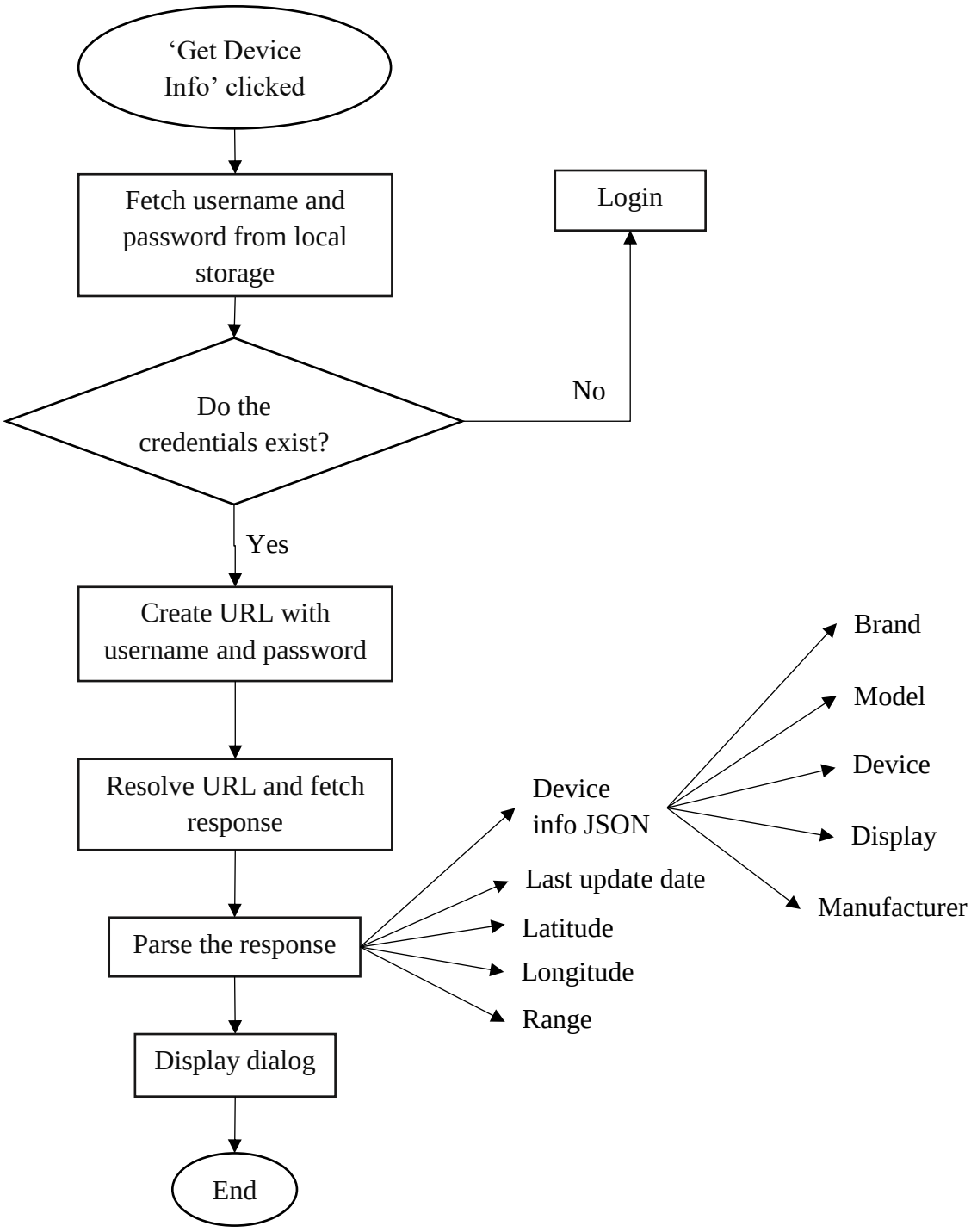


Figure 69. Flow of obtaining user information from the database via the web interface

14. Triggering Methods

“Triggers”, within the context of this project, are phrases or special words designed specifically to illicit a response upon receipt by the target system. Extrapolating, the usage of this feature permits an external user to send phrases across a myriad of different communication mediums in pursuit of causing an action to take place within the target device. This can be a simple lock or erase of a lost device, in which case the user is not expecting any feedback from the request. However, triggering also extends to remote retrieval of information, achieved by the system formulating, and subsequently sending, a response message to the sender upon receipt of certain triggers, this is called a remote two-way communication channel. This chapter details the implementation of these trigger methods, and how a two-way communication channel is established with that medium to send device information back to the sender of the trigger.

14.1. *Battery-Delayed Email Triggering*

14.1.1. *Introduction to Battery-Delayed Email Triggering*

Before the conception of GCM triggering, a more rudimental method was devised and implemented. It relies on the user of the application providing the login credentials of an email account, specifically a Gmail account for simplicity. Once enabled, on a battery level change the application attempts to retrieve emails from the Gmail mail server over TCP/IP using the IMAP (Internet Message Access Protocol). If authentication is successful, it then scans each message within the inbox for unread or “unseen” messages, parsing the subject line and sender of each unread email to an analyser method for comparison against saved triggers. If a trigger is oriented upon information response, the application opens a connection to an online, hosted PHP mail script with the original sender address and a message body passed as HTTP POST parameters. This relays the desired information back to the email address of the trigger sender by sending an email.

14.1.2. *Android Implementation*

The Android operating system can receive many different battery intent broadcasts. The most general was chosen due to its susceptibility of being fired, implying that the trigger-checking can be executed on a semi-regular basis whilst the app is not in the foreground, something that would have been a requirement without the use of a broadcast receiver. One action was registered in the filter for the battery broadcast receiver, namely `BATTERY_CHANGED`. This was chosen based on the probability of the related intent broadcast being fired. The broadcast has many general “triggers” that result in frequent firing of the broadcast, encompassing the battery level rising, falling, and external power being disconnected and connected to the device [57]. This makes it appealing as a method for performing background actions at regular intervals.

```
<receiver
    android:name=".BatteryReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter android:priority="999">
        <action android:name="android.intent.action.BATTERY_CHANGED" />
    </intent-filter>
</receiver>
```

Figure 70. Declaration of a battery BroadcastReceiver

Within the receiver, the application checks that the user has enabled the feature and has supplied email credentials. Once the email and password has been retrieved from shared preferences, a new `EmailFetcher` object is created with the username and password as constructor parameters, the class inherits AsyncTask, which permits the networking operation to be performed silently on a background thread [58], without UI interaction. To facilitate the connection to the mail server over IMAP, an external library, javax mail, was used to simplify the email fetching process. The library provides many classes that models the email system. Most important of which is the "Store" class, an abstract class that models a message store and its access protocol, for storing and retrieving messages [59]. Firstly, an instance of a Store object is created with the mail server properties as a parameter, with another method call `getStore()` with the required IMAP protocol:

```
Store store =
Session.getDefaultInstance(getServerProperties()).getStore("imap");
```

The mail connection settings are encapsulated in a `Properties` object, including the port and host settings:

```
private Properties getServerProperties() {
    Properties properties = new Properties();
    properties.put(String.format("mail.%s.host", "imap"), "imap.gmail.com");
    properties.put(String.format("mail.%s.port", "imap"), "993");
    properties.setProperty(String.format("mail.%s.socketFactory.class", "imap"),
"javax.net.ssl.SSLSocketFactory");
    properties.setProperty(String.format("mail.%s.socketFactory.fallback",
"imap"), "false");
    properties.setProperty(String.format("mail.%s.socketFactory.port", "imap"),
String.valueOf("993"));
    properties.put("mail.smtp.starttls.enable", "true");
    return properties;
}
```

After the Store object is adequately initialised, a call to authenticate with the account is performed with a public method call `connect()` using the user-supplied credentials. Once authenticated, the mail inbox is represented as a Folder object and opened.

```
store.connect(user, pass);//connect with supplied credentials

Folder inbox = store.getFolder("INBOX");//get main inbox folder
inbox.open(Folder.READ_WRITE); //write access to set "seen" flag
```

Here, write access is requested, this is warranted by the methodology requirement of exclusively scanning unread emails, creating a filtering mechanism that prevents repeated triggering based on past emails in the inbox. Each message within the Folder object is treated as a separate Message object, an initial check is performed that the message does not contain the "seen" flag with a method call to `getFlags()`, that is, the message is unread. Once the email subject and sender are passed to the analysis method, if the message's subject line contains a trigger, the necessary action is performed before the message's "seen" flag is set to true such that the message will not be considered in the future, preventing execution of past triggers.

```
for (Message message : inbox.getMessages(1, inbox.getMessageCount())) {
//iterate over messages
    if (!message.getFlags().contains(Flags.Flag.SEEN)) {//if the message is
unseen
        this.currMessageObj = message;
        this.currMessageSubject = message.getSubject();
        this.currMessageSender =
         message.getFrom()[0].toString().split("<")[1].split(">")[0];
        analyseEmailSubject(); //start analyser
    }
}
```

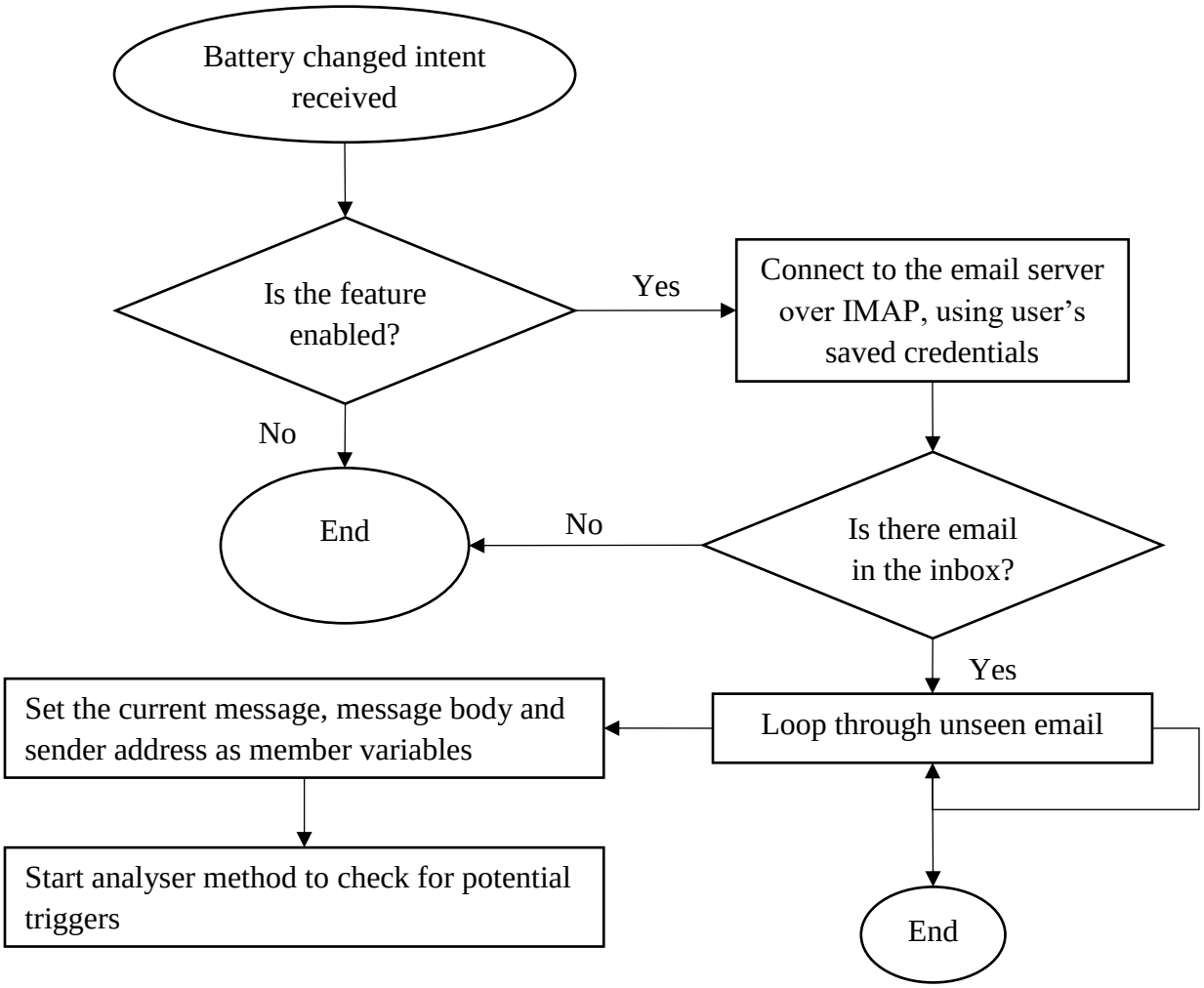


Figure 71. Lifecycle of checking for email triggers

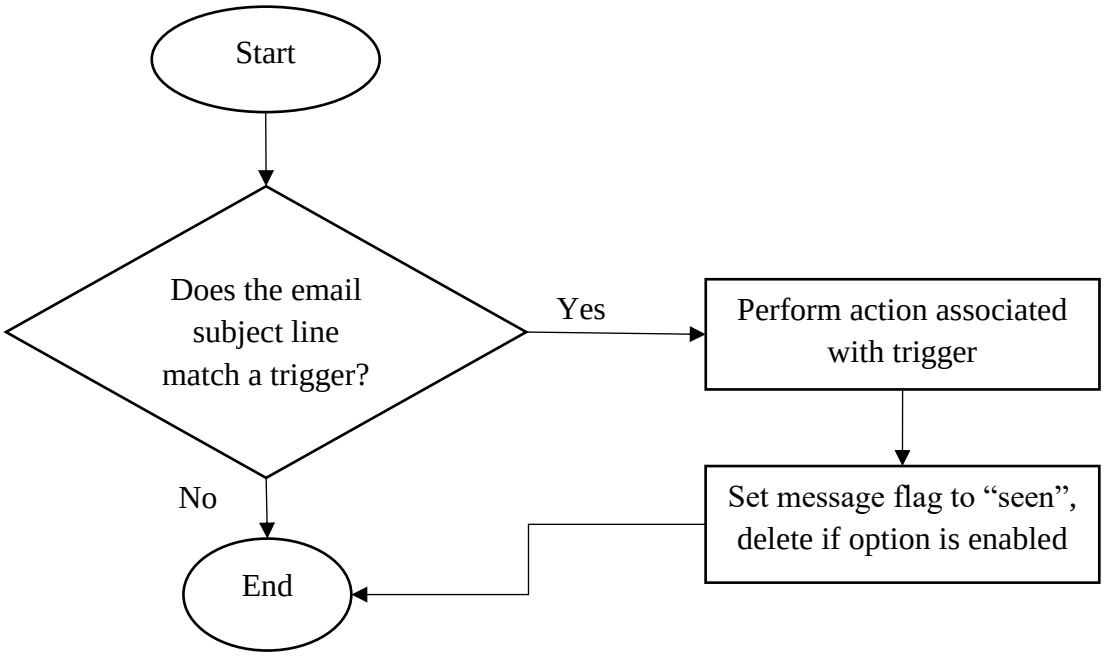


Figure 72. Lifecycle of analyser method

14.1.3. *Two Way Communication Methodology*

14.1.3.1. *PHP Mail Script*

Initially, the method of relaying information based on email was achieved by utilising the user’s credentials to send an email reply to the original sender, but due to the authentication that was required in accessing and sending email from the application, it produced unreliable results and was flagged and blocked temporarily from accessing the account, requiring manual unflagging of the mail account. It was therefore necessary to create an authentication-free solution to send emails, this came in the form of PHP mail. PHP mail, or rather the `mail()` PHP function, allows the sending of emails directly from a PHP script. For the mail functions to be available for a script to utilise, PHP requires an installation and an operational email system [60]. The program to be used is defined by the configuration settings in the php.ini file on the host of the PHP script. If available, the sending of an email can be completed in a single function call:

```
mail(to,subject,message,headers,parameters);
```

This provides a great amount of abstraction from the underlying methods responsible for transmission and does not provide a viewer of the script with sensitive authentication information that could be exploited. The hosted mail PHP script takes three POST parameters based on the PHP mail function’s required parameters: “to”, “subject” and “text”, these are obtained within the script with calls to `$_GET`, used to collect the data passed in the URL.

```
<?php
    $to = $_GET["to"]; //get the recipient from URL
    $subject = $_GET["subject"];//get the subject from URL
    $text = $_GET["text"];//get message body from URL
    $headers = "From: webmaster@locationservice.com"; //custom sender name

    if(mail($to,$subject,$text,$headers)){ //if email was successfully sent
        echo "The email was successfully sent."; //print response
    } else {
        echo "The email was NOT sent.";
    }
?>
```

Figure 73. *mail.php* email script

To use this script to send a simple email, a URL containing the location of the hosted script along with the required parameters is resolved by the Android application e.g.:

https://xxxxxxxxx/mail.php?to=recipient@mail.com&subject=subjecttext&text=textbody

The script will then attempt to send an email with the following characteristics:

- To: recipient@mail.com
- Subject: subjecttext
- Message: textbody

14.1.3.2. *PostPHP Class*

The `PostPHP` class inherits AsyncTask for background networking tasks and is responsible exclusively for triggering email sending via the hosted PHP mail script, and is used in multiple methods throughout the application that revolve on an email response to a trigger, and excels as a method for relaying large amounts of information compared to the SMS relay methodology. The class object is initialised with a context, used for additional call log and contacts fetching operations that may be required in the URL's construction as parameters. To run the email task, it executes using the first array of string objects passed as the argument. This array should contain the recipient address, the subject, and finally the text body of the message respectively to be used in the email creation. That is, to execute an email sending task, a string array must be presented in the following format/syntax:

```
new String[]{recipient, message, subject}
```

Thus, to execute the background email task with recipient "recipient", subject "subject" and message "message":

```
new PostPHP(context).execute(new String[]{recipient, message, subject});
```

For robustness, the PostPHP AsyncTask firstly opens a connection to a statically hosted text file which contains the current URL of the mail script, such that if the server needed to be

switched, the text file could be edited to contain the URL of the new server, removing the need to alter the Android application on a server change. Once the URL is obtained from the text file, it is then modified with the information attained from the string array, array[0] being the recipient, array[1] being the subject and array[2] holding the desired email message string. Finally, other information required such as call and contacts logs are then concatenated to the message string. The URL is then resolved, and a connection opened.

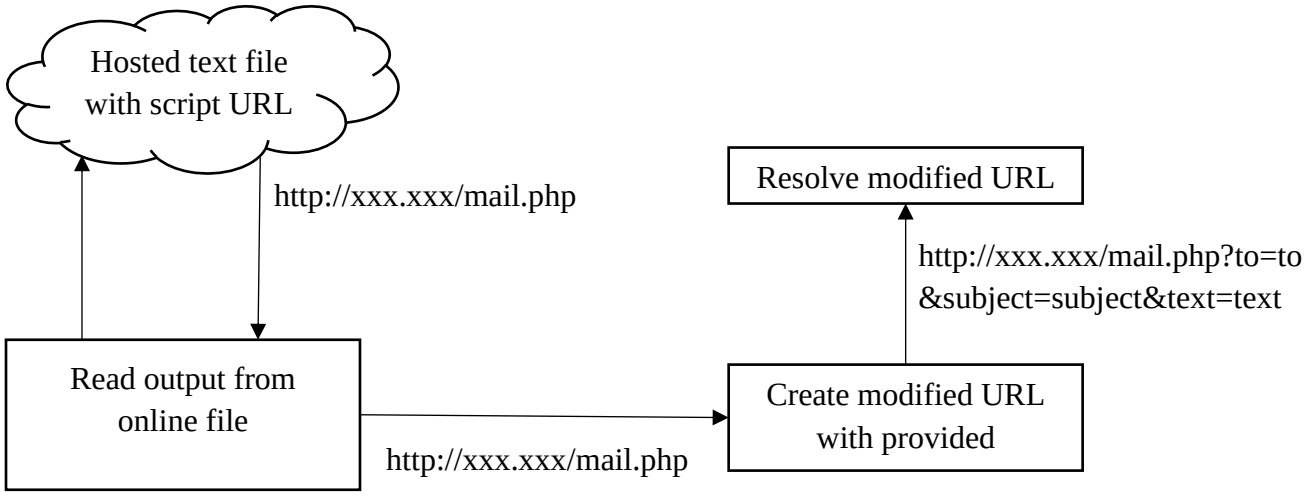


Figure 74. *PostPHP* lifecycle

```
String[] finalArr = strings[0];//get the first string array

StringBuilder sb = new StringBuilder();//new string storage for the URL
BufferedReader br = new BufferedReader(new InputStreamReader(new
BufferedInputStream((new
URL(c.getString(R.string.redirect_mail)).openConnection()).getInputStream())));/
/open connection to text file
String inputLine;
while ((inputLine = br.readLine()) != null) { //read each line of the file
    sb.append(inputLine); //append line of file to string builder
}
br.close();//close the connection
HttpURLConnection connection = (HttpURLConnection) new URL((sb.toString().trim()
+c.getString(R.string.to) + finalArr[0] + c.getString(R.string.subject_param) +
finalArr[1] + c.getString(R.string.text_param) + finalArr[2] +
addExtraInfo(c))).openConnection();
connection.getInputStream(); //open a connection to the modified URL with
parameters taken from the string array
connection.disconnect(); //finally disconnect
```

Figure 75. *PostPHP* background task to trigger the execution of the hosted PHP mail script

14.1.4. *Caveats and Disadvantages*

The systematic disadvantage of this method is that it is dependent on a battery-related intent being fired, whether that be the level dropping or rising or charging taking place, it will not provoke an immediate action and the response time, including email response time, will vary based on this variable. The PHP `mail()` function itself is performed on a "best-effort" basis on many different hosts, this does not guarantee delivery and transportation of the message [61] and can be blocked by the host's outgoing email spam filter. Another issue is authentication; the accessing of the provided inbox can produce unwarranted flagging of the email account, especially when a connection is made from a suspicious location. Although it can be mitigated by enabling IMAP access on the monitored inbox and allowing greater access from applications, this makes the method unreliable as the main form of communication over GCM and SMS, with sporadic results throughout testing.

14.2. *SMS Triggering*

14.2.1. *Introduction to SMS*

SMS or Short Messaging Service is a text messaging component service present in all modern mobile device systems; it uses communication protocols enabling users to exchange small, plain-text messages of a maximum size of 160 characters [62]. The text can comprise of any alphanumeric combination of words and/or numbers. As of 2010, SMS was the most widely data application, with 80% of all mobile users (3.5 billion) utilising the service [63]. The widespread adoption of SMS makes it an appealing mechanism and communication channel to send trigger messages from a cellular-enabled device to another target device and receive responses, with the only limitation being the recipient device requiring a cellular connection. One benefit of using SMS as a triggering mechanism is the "confirmation of delivery" it provides [64], unlike other methods of communication, the sender of the message can intercept a return message on receipt of message delivery, providing more feedback within SMS applications.

14.2.2. *Android Listener Implementation*

The Android OS provides the ability to implement an SMS listener service to receive incoming messages. Within this, the message can be parsed, and a look-up performed against a list of triggers stored in the application. Along with message content, the sender's mobile number is enclosed, adding the capability of information relay based on particular triggers as well as simple, one-way trigger message delivery. For an application to intercept the messages, it must declare a Broadcast Receiver in the manifest with the specific `android.provider.Telephony.SMS_RECEIVED` intent filter, the intent of which is fired on receipt of an SMS message. A good addition to this declaration is a priority. Priority is given as an integer value between -999 and 999 inclusively, the priority value controls the order in which broadcast receivers are executed to receive broadcast messages. Those with higher priority values are called before those with lower values [65]. Setting this priority to the highest value

ensures that the message body can be consulted for triggering before any other registered receivers are executed, allowing for a fast response time, and a fast turnaround for relay-based triggers.

```
<receiver
    android:name=".SMSReceiver"
    android:enabled="true"
    android:exported="true"
    android:permission="android.permission.BROADCAST_SMS">
    <intent-filter android:priority="999">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Figure 76. AndroidManifest declaration of SMS listener/receiver service

14.2.3. *The PDU and Message Parsing*

To evaluate an SMS message, the message must be recreated from a PDU. A PDU, or Protocol Data Unit, is the standardised format of SMS messages, it contains metadata of the message such as sender and the text encoding as well as the message body. A large message may be broken down into multiple PDU objects when entering the Android system. To process the message effectively, the PDUs are extracted from the Broadcast Receiver's intent using "pdus" as a key in a API call for extra data (`getExtras()`).

```
Object[] smsExtra = (Object[]) intent.getExtras().get("pdus");
```

The result of this call is an array of objects, these are the fragments of a complete SMS message. Fortunately, Android provides a specific method call to recreate a message from a given PDU with the `SmsMessage` class offering the `createFromPdu()` method.

```
SmsMessage sms = SmsMessage.createFromPdu((byte[]) smsExtraObject);
```

This method takes the raw PDU object as a byte array as the parameter, and, after construction, creates an `SmsMessage` object [66]. From here, metadata can be analysed through various method calls performed on the object, such as `getMessageBody()` to get the message as a string, and `getOriginatingAddress()` to identify the sender for response/relay purposes. This provides the Android application will all the information required to take appropriate actions based on the message content.

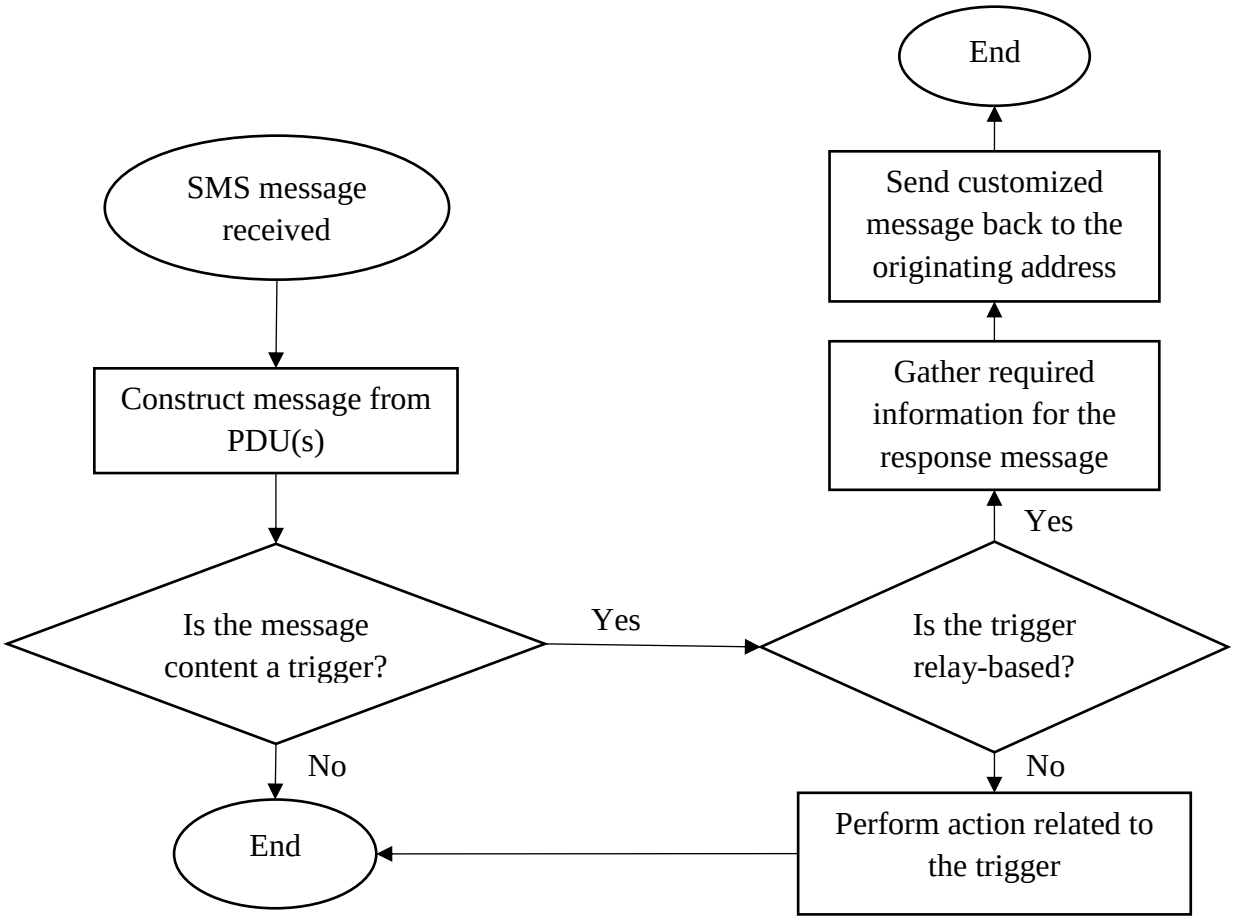


Figure 77. SMS Receiver lifecycle

```
public void onReceive(Context context, Intent intent) {

if(PreferenceManager.getDefaultSharedPreferences(context).getBoolean("enable_tri
ggers", true)) { //check if triggering is enabled
        if (intent.getExtras() != null) { //check that the intent has extras
            Object[] smsExtra = (Object[]) intent.getExtras().get("pdus"); //get
pdus store
            if(smsExtra != null) {
                for (Object smsExtraObject : smsExtra) { //iterate over objects
                    SmsMessage sms = SmsMessage.createFromPdu((byte[])
smsExtraObject);//create an SMS object from the object's byte array
                    this.message = sms.getMessageBody().trim();
                    switchMessage(context,
sms.getOriginatingAddress().trim());//pass the text, sender to an analyser
method
```

Figure 78. The `SMSReceiver` class constructs SMS message objects from PDUs and analyses their content in a separate method.

14.2.4. *Two Way Communication Methodology*

As in the figure above, for relay-based triggers it is also necessary to identify the sender such that `getOriginatingAddress()` must be called and the result sent to the analyser method along with the message body. If the message body matches such a trigger, first the information that needs to be sent back is summarised and returned as a string object. There are limitations to this however, if the string's content exceeds the 140-byte data size, or 160 characters, the sent SMS message will be split. Although equally as readable to the recipient with modern SMS inbox applications, the extra message(s) being sent understandably contributes to the host mobile's carrier SMS limits, it is therefore necessary to prioritise the imperative text such as location coordinates to be inserted into the text message, a limitation not present with email relaying. The `SmsManager` class manages all SMS operations such as sending data, text, and pdu SMS messages, the object of which is gained by calling the static method `getDefault()` [67]. A simple text message can then be sent using the public `sendTextMessage()` method:

```
SmsManager.getDefault().sendTextMessage(String destinationAddress, String
                        sourceAddress, String text, …)
```

This method allows the Android application to utilise the originating address obtained from `getOriginatingAddress()` as the destination address of the SMS message, creating a useful two-way communication channel between the application's host device and the external user.

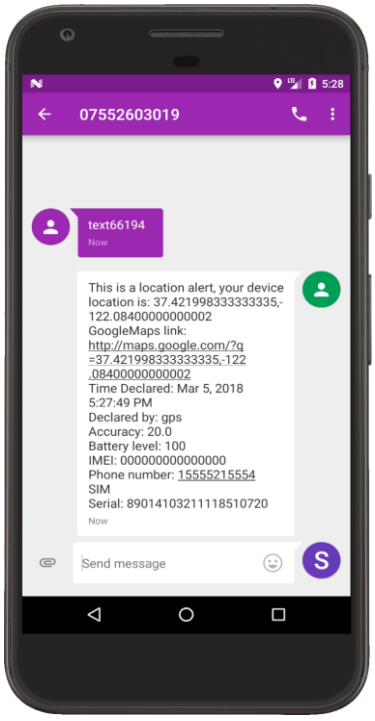


Figure 79. Example location SMS message sent by the application in response to an SMS trigger

14.3. *Cloud Messaging*

14.3.1. *Introduction to Cloud Messaging*

Google Cloud Messaging (GCM) or Firebase Cloud Messaging (FCM) is a free service offered by Google that facilitates the transport of messages between servers and client applications (downstream messaging) and likewise client-server messaging (upstream messaging). [68] The GCM service handles all transport and delivery which makes it appealing as a powerful and code-efficient notification and messaging technique for Android and web applications.

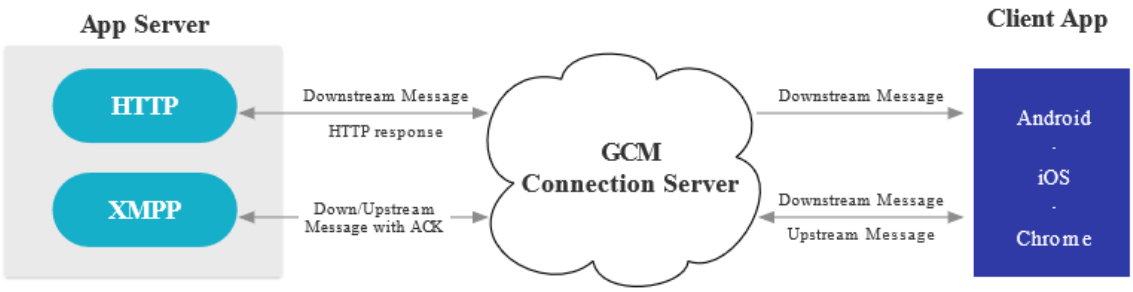


Figure 80. GC message flow [68]

The figure above explains how each component in the GCM process operates. The connection servers accept the downstream messages that are sent by the application server, the connection server then facilitates the transport of the message to a client application on a platform such as android. To communicate cloud messages from a web application to a client application, an API key needs to be generated registered with GCM servers for authentication, this is utilised in the app server and wherever sending is required. The client app, to send and receive messages, must register with the server and obtain a registration token. The one caveat with using this choice of messaging is that the payload of a message is limited to a size of 4KB which can restrict the level of information transmitted within the data body of a message. The messaging options that are available are defined by payload: notification and data. Notification messages are lightweight and carry a limiting payload size of 2KB, for the purposes of this application the data option is utilised to allow the transfer of more relevant information and the added ability of the client application to parse the extra data. The typical components of a message include the “to” or target of the message (device token) and the data payload, in this paper messages are represented as JSON objects with customised data payload options.

```
{
   "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
   "data" : {
     "Nick" : "Mario",
     "body" : "great match!",
     "Room" : "PortugalVSDenmark"
   },
 }
```

Figure 81. Anatomy of a typical JSON GC message

14.3.2. *GCM Triggers*

The customisation aspect of the data field within the payload allows for the transmission of plain text within the GC message that can be parsed and checked against a list of saved triggers by the Android client application. This can be referred to as GCM triggering. By implementing cloud messaging within the app, the Android application can register with the GCM server to receive a token, this can then be used within the web application to send trigger words and phrases to the target device.

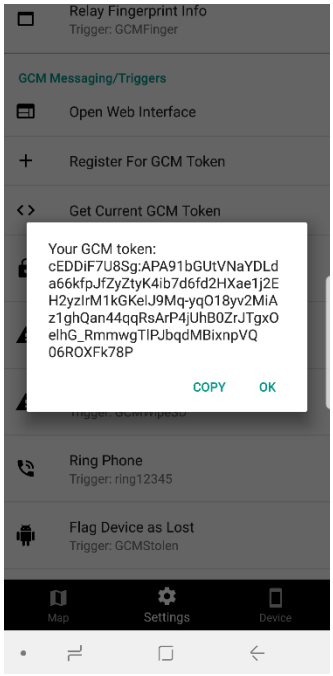


Figure 82. Find My Android GUI displaying a generated device token

For device registration, the application starts a registration service to retrieve an Android token from the GCM server. Once this token is generated and displayed to the user, it can be used as a recipient of messages through the web interface. This token is then saved to shared preferences to be retrieved by the user at any given instance through the GUI.

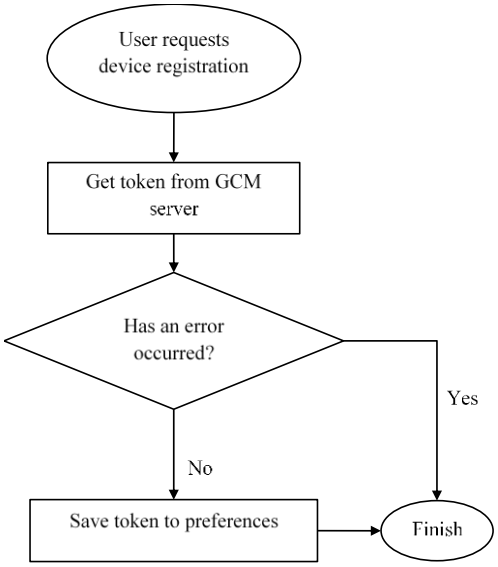


Figure 83. Process of obtaining an Android device token from a GCM server

```
Intent registration;
try {
    String token =
InstanceID.getInstance(getApplicationContext()).getToken(getString(R.string.gcm_def
aultSenderId), GoogleCloudMessaging.INSTANCE_ID_SCOPE, null);//get a token

PreferenceManager.getDefaultSharedPreferences(getApplicationContext()).edit().putSt
ring("GCM_Token", token).apply();//save to prefs
    registration = new Intent(REGISTRATION_SUCCESS);//specify success intent
    registration.putExtra("token", token);//put token in intent
} catch (IOException e) {
    registration = new Intent(REGISTRATION_ERROR);//send error intent
}
LocalBroadcastManager.getInstance(this).sendBroadcast(registration);//send
broadcast
```

Figure 84. Android's RegistrationIntentService's *onHandleIntent()* for handling the registration of a device token

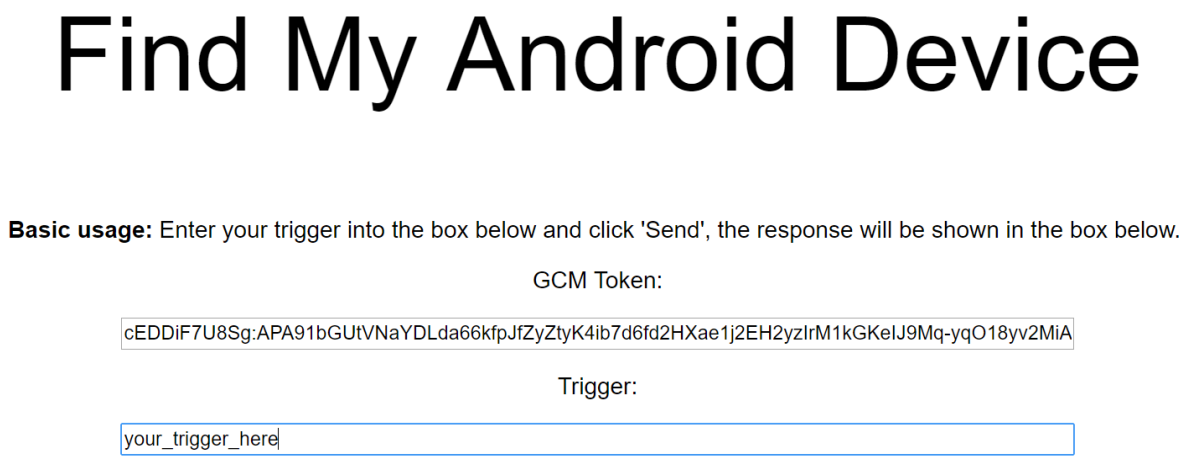


Figure 85. The web interface with device token (recipient) and trigger entry

To send messages from the website to an Android device, the Android API key needs to be used as the authorisation key within the GCM authorisation key field. Upon loading the webpage, the current Android API key is attained from a statically hosted text file on the website. After this is performed, the interface obtains a token from the GCM server for optional two-way or "GCM Relay" communication, detailed later in this paper. The website now has the capability to send GCM triggers with the user-submitted device token and trigger as payload information, this is achieved through a POST request to the GCM URL.

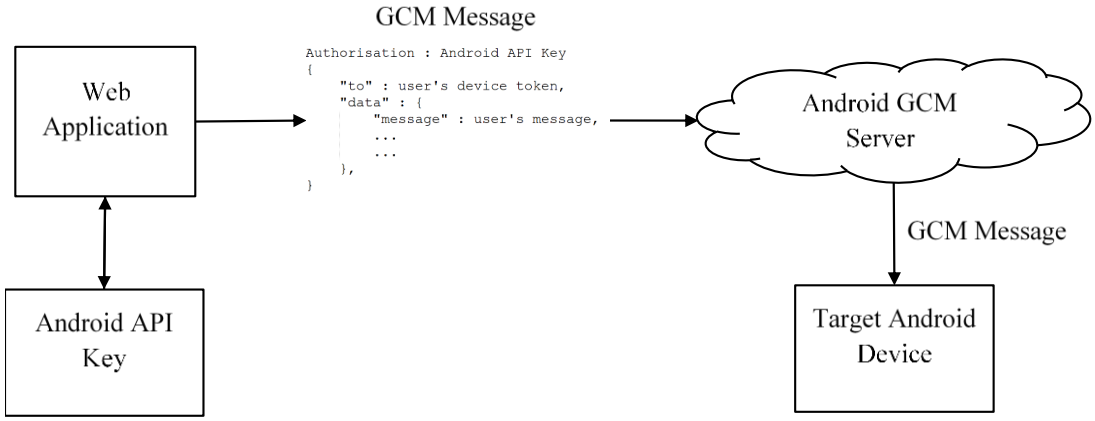


Figure 86. Example of the transmission of a GC message with user submitted receiver (device token) and trigger message

The website sets the "to" or receiver of the payload to the device token the user has submitted and the message payload equal to the trigger text, with the loaded API key used as

authorisation for the downstream message. Other options within the payload, sender and extra, are explored later in this paper.

14.3.3. *Client-Side GCM Parsing*

Client side, the message is received within the `onMessageReceived` method with the class inheriting `GcmListenerService`, this service allows devices to accept GC messages without the app being in the foreground, saving battery usage. The message is represented as a bundle object that's accompanied with the sender as parameters of the method. To parse this data, knowledge is required concerning the fields/keys that were applied to the GC message upon sending. The data is then extracted from this bundle by a call to `getString()` on the bundle object with the message/trigger key, "*message*". The string object can then be handled and analysed against saved triggers to perform the necessary action(s).

```
public class GCMReceiverService extends GcmListenerService {
    @Override
    public void onMessageReceived(String from, Bundle data) {
        String isNull = data.getString("message");
        if(isNull != null && isNull.trim().length() != 0){ //if a valid message
            new GCMHandler(isNull,data, getBaseContext()).examine();//send to handler
        }
    }
```

Figure 87. Extracting a string message from a received GC message bundle and calling examine method

14.3.4. *GCM Relay Triggers and Two-Way Communication Methodology*

With some triggers there is an inherent need to relay data, such as responding with a location update. For information that results in a small payload size, the GCM service can be used instead of using rate-limited traditional email services. This relies on providing the receiver, in this case the Android client, with sufficient data to identify the sender of the message and respond according to the trigger contained within the message body. For a user to send messages to a specific Android device through the webpage and receive a reply, certain technical conditions must be met. Firstly, the web application must have a separate API key to that used within the Android application; the web application cannot send messages to an Android device using a generated web API key as the device is registered with a different GCM server. It was then necessary to establish a means of communicating cross-platform and creating a two-way communication link. To do this, the web application registers with its GCM server and obtains a token upon loading the page, it then attaches its own token to the data payload of the GC message when a reply is required, sending the message using the Android API key as authorisation. Once the Android device receives the data, it can be parsed to identify the device token of the browser. The app then establishes if the message component of the payload matches a "GCM relay" trigger, if so, the app performs the necessary action and replies to the message by sending an upstream message using a HTTP POST request using the Web API key for

authorisation, with the recipient being the parsed device token belonging to that browser instance.

The user interface provides a checkbox to flag the message being sent as a relay request. Once the user presses ‘Send’, the script within the webpage attaches the unique token generated when the page loaded and attaches it to the “sender” field of the message’s data payload. For ordinary messages this field reads “null” for easier parsing client-side. Once received, the Android app can determine the sender token with a call to `getString()` on the message bundle, using “sender” as the key.

Two-way communication check list:

- Two separate API keys, one Web, one Android
- The web application must have a token
- The sender of the message (token) must be contained within the message
- The message must be flagged as a “reply trigger” to differentiate from ordinary GCM triggers by either a separate data field or protected text such as “null”

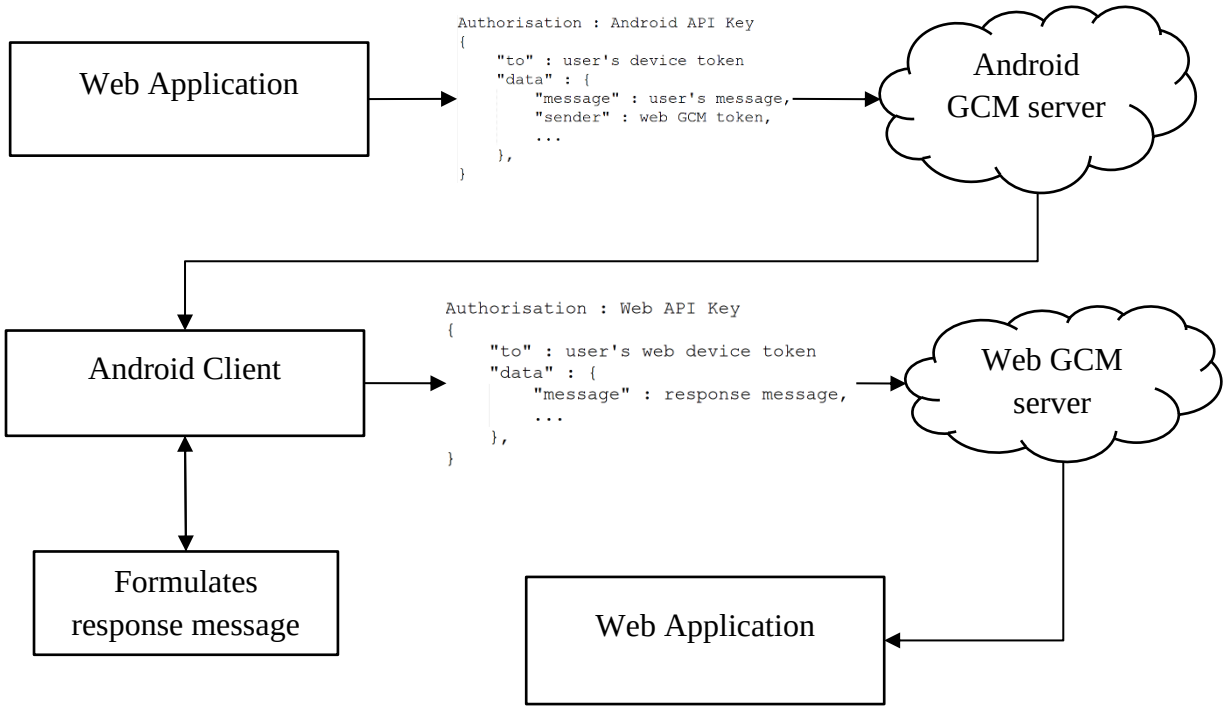
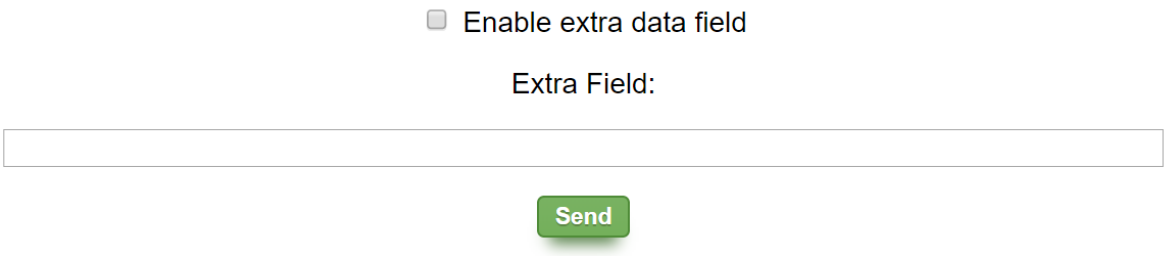


Figure 88. A two-way GC messaging solution

14.3.5. *"Extras"*

"Extras" was devised as an extra field within the GC message with the sole purpose of providing information to override default preferences within the application when analysing a GCM trigger. Certain GCM triggers such as those that engage in email and SMS sending perform an additional check for this extra data field to understand where the data should be sent to and if it should be redirected. For example, if a user sets their preferred phone number within the app, when the GCM trigger to send an SMS location message is intercepted, the device will respond by sending the message to that number. Whereas if the user has forgotten their preference, the user can send a different phone number within the extra data field to overwrite the saved preference.



If the user ticks the 'Enable extra field' option, the contents of the text field is added to the data payload under the 'extras' field when sent, otherwise it defaults to "null" when extras is disabled alike the sender field in GCM relaying. If an extra field is attached with a trigger that does not support extras, such as a trigger to sound an alarm, the contents of the extra field is ignored, and the action performed.

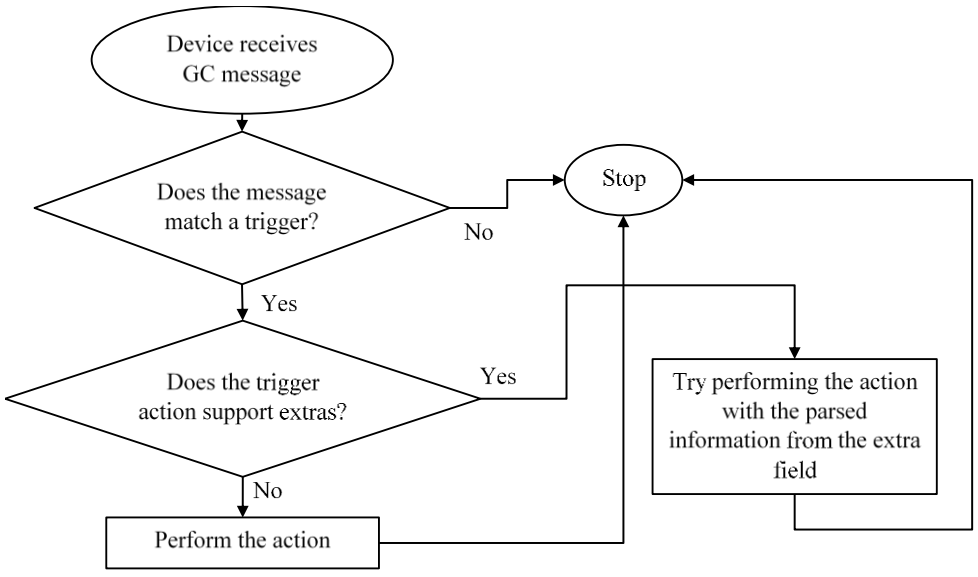


Figure 89. "Extras" example usage

14.3.6. *Caveats of GC Messaging*

Due to the additional device token of 255 chars contained within the relay message, the size (length) of the trigger that can be sent is limited. Although a technical challenge from the web client perspective, the client does not have such an addition, so can reply using a maximised level of payload size availability for the data field. A disadvantage of the web client is that if the user utilises all the functionality at once, with a large trigger and extras string, the message may refuse to send completely. It is then important to inform end users of this restriction. One other general consideration is speed. GCM has no guarantees of latency as a free service, not an inconvenience with one-way triggering with which there is no urgency, however, with a two-way communication channel the user has a reasonable expectation of speed of response, which may not always be satisfied with this service.

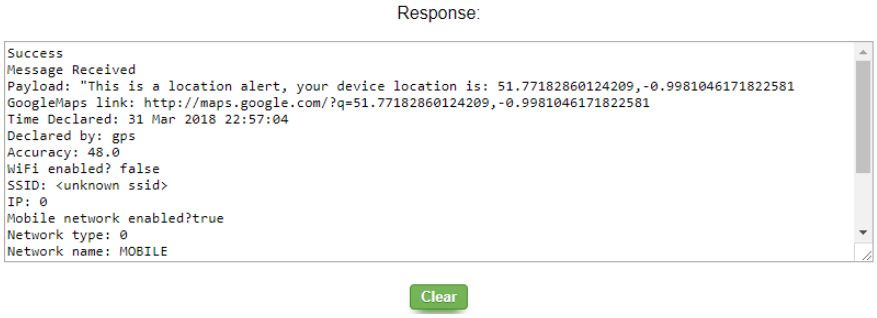


Figure 90. Example response from a GCM relay trigger, presented within the web interface

15. Testing: Verification and Validation

Testing is an important phase of software projects, it identifies and highlights any faults and errors within the implementation that could result in not meeting the target objectives of the project. This section will discuss the testing that was conducted for the project on a subsystem level. These subsystems, the Android application and the web interface, are the only components needed to test. These components are related to each other and the database subsystem. However, as the database is simply a store of user data, it does not require additional subsystem testing. Furthermore, the functionality and use cases of the database are adequately tested within the web interface and Android application subsystem testing.

15.1.1. *Approach*

The objective of testing the implementations created in this project is to verify that they satisfy the requirements outlined. The tests will provide coverage of the technical objectives as well as the user experience and acceptance criteria. Subsequently, the tests performed will be at a high level to prove the functionality of the application and other implementations. This section will present the test scenarios, results, and commentary from these tests where appropriate. The approach taken tested each subsystem (component) separately, as these components are

intrinsically linked and sometimes dependent on one another, this was seen to be sufficient to test the system overall.

15.1.2. *Limitations*

Most of testing will be constrained to emulated devices. These will run varying Android versions to check that functionality is maintained over versions, and restricted where appropriate. Compatibility with other devices will also be checked to ensure the application deploys and runs on other versions of Android, including devices with varying screen sizes. The emulated devices will be run by the Android Studio IDE used to develop the application itself. Moreover, testing will only be performed by a single user, limiting the scope and speed of testing, which may cause issues to go unnoticed.

15.1.3. *Subsystem 1 (Application) Test Cases*

The following tests concern the Android application subsystem only, rather than an evaluation of the system as a whole (system testing).

Test ID	**Test Title**	**Test Description/Expected Result(s)**	**Actual Result(s)**	**Comment(s)**
1	Pressing a trigger opens the text dialog	The dialog opens for user configuration	The dialog opens for user configuration	N/A
2	Changing the trigger within the dialog and confirming saves the trigger	Changing a trigger within its GUI dialog alters the actual trigger	Changing a trigger within its GUI dialog alters the actual trigger	N/A
3	On first use dialogs	On first use of the application a permissions prompt is displayed	As expected	Runtime permissions dialog displayed
4	Map fragment	The map fragment shows the current location of the user	As expected	Shows the map fragment with declared provider and stats

5	Changing/reordering	Reordering the providers causes the map fragment to use the preferred provider in displaying the location	As expected	Map fragment uses position obtained with the new user-preferred provider
6	Device fragment	Device fragment displays device information correctly	As expected	IMEI is default on emulator images, other details unaffected
7	Enabling device administrator	Enabling the device administrator works correctly	As expected	The application becomes a device admin after user prompt
8	Password protection	Adding a password prevents the automatic login	As expected	A blank password bypasses the login
9	Flagged devices	Flagging a device enables emergency features (battery flare)	Battery flare email is fired only when device is flagged	N/A
10	Password hints	An incorrect password attempt displays the hint	As expected	Hint displayed temporarily
11	GCM tokens	GCM token registration completes and displays token to user	As expected	Delay caused by network
12	Geofencing	Exiting a Geofence causes an email alert	As expected	Will systematically not work without a

				connection
13	Account signup	Account signup with valid credentials causes a successful signup	As expected	App shows the newly created account credentials
14	Account signup, not valid	Error message displayed when invalid/duplicate credentials used	As expected	When duplicate or empty credentials used
15	Account signup causes insertion	New account information is inserted into the database	As expected	New user information (device info) is inserted along with username and hashed password of user
16	Account updating	User database entry updates on battery level change	As expected	"Time declared" column used to verify when user account is updated
17	Updating manually	Pressing "Update Now" updates the database instantly	As expected	Instance-based, limited use cases
18	Hiding app feature	Pressing the "hide app" button hides the application from the Android drawer	As expected	Requires triggering to un-hide
19	Cell tower feature	Clicking "Cell towers" GUI option displays the nearest cell tower and appropriate radius	As expected	Requires network connectivity (to database), an informative-only feature
20	"Map type" map	Altering the map type	As expected	An aesthetic

	fragment	changes the display of the map within the map fragment		feature
21	Battery below threshold	Email alert sent when battery flare enabled and level drops below threshold	As expected	Network connectivity required
22	Salted security	Triggers modified with a random sequence of numbers on first launch	As expected	Tested on two devices to check for randomness
23	Triggering by Gmail (battery delayed email triggering)	One-way triggering occurs when Gmail credentials are supplied and the battery level changes	As expected	"Trigger To Remotely Lock Phone" was used to verify this test
24	Response by Gmail (battery delayed email triggering)	A response occurs when Gmail credentials are supplied, a secondary email is specified and the battery level changes	As expected	"Trigger To Relay Position" was used to verify the response (viewed in the secondary email account.)
25	SMS triggering	One-way triggering occurs when a SMS-based trigger message is received and the feature is enabled	As expected	"Trigger To Lock Phone" trigger phase used to verify SMS triggering functionality
26	SMS trigger response	A response occurs when SMS triggering is enabled and an SMS relay trigger message is intercepted	No response	Fixed: large SMS responses messages now broken down and sent as a multipart message. Trigger to relay

				position used to test the affirm the fix
27	GCM token display	Application can display registered GCM device token	As expected	If token not saved/generated: “No GCM token” message printed
28	GCM token registration	Application able to register a GCM device token that persists over reboots	As expected	Device restarts and “Show GCM token” feature used to complete the test
29	GCM triggering	Application performs the required action upon receiving a trigger message phrase	As expected	“Trigger To Lock Phone” trigger phase used to verify GCM triggering functionality
30	GCM trigger response	A response occurs when a GCM relay trigger message is intercepted	As expected	Response observed in the output box on the web interface
31	SIM card monitoring	An alert email is sent upon a SIM card change	As expected	A physical Android device (Samsung Galaxy S8) was used to facilitate this test
32	Wi-Fi fingerprinting	The calibration activity is able to take and save N AP samples under a user-provided alias	As expected	Entering the same activity again results in the saved fingerprint being visible after a reboot of

				the device
33	Bluetooth fingerprinting activity	Calibration activity allows the calibration of a visible Bluetooth device	As expected	A Bluetooth beacon and a physical Android device used
34	Editing existing Bluetooth calibrated devices	Calibration activity allows for the editing of saved calibrated devices (recalibration)	As expected	Relayed positional data used to confirm a new calibration
35	Deleting Bluetooth calibrations	Deleting existing calibrations works correctly	As expected	No calibrated devices visible after a reboot
36	Resetting Bluetooth calibrations	Resetting the calibration of a given device works correctly	As expected	GCM response used to verify a default value is returned for a distance report (0.0cm)
37	User-preferred location provider used in a location response	The precedence of Android location providers is used when obtaining and returning a location fix	As expected, the preferred provider (1st choice) is used to obtain location information	GPS was used as the preferred location provider, the location information returned confirmed the desired provider was used to declare the location

Table 8. Subsystem 1 (Application) Test Cases

15.1.4. *Subsystem 2 (Web Interface) Test Cases*

The following test cases concern the functionality of the web interface, used to communicate with the device through cloud messaging and the retrieval of user information through communication with the database.

Test ID	**Test Title**	**Test Description/Expected Result(s)**	**Actual Result(s)**	**Comment(s)**
38	Database information retrieval	"Get Device Info" displays all user information including last known location when a valid login is provided	As expected	No information displays when invalid credentials are supplied
39	Get browser GCM token	"Get Browser Device Token" displays the current GCM device token used by the interface	As expected	A small delay is present, meaning the token is only generated once the page has fully loaded
40	Response box	Response from a GCM relay trigger displayed in response box	As expected	GCM trigger to rely device location used in test
41	Invalid trigger	An invalid trigger phase does not cause any device action/response	As expected	A random phrase used to measure device response
42	GCM relay trigger without flag	A GCM relay trigger sent without the relay flag set does not result in a response	As expected	The option alters the payload that informs the application whether to respond
43	"Extras" field	Specifying a valid extra field with an	As expected	Used the trigger responsible for

		acceptable/valid trigger causes the extra field value to override the current response destination		sending a location update in an SMS message, the extras field was specified with a different phone number to the saved number within the application

Table 9. Subsystem 2 (Web Interface) Test Cases

15.1.5. *Website Cross-Browser Tests*

The website testing was dissected into two sections. Firstly, cross browser compatibility was tested with the use of IE9, IE10, Edge, Firefox, Safari and Chrome web browsers, currently the most popular browsers in the market. Secondly, the web interface page was analyzed by the W3C validator [69] to check for issues from which cross browser problems and accessibility problems may arise. This ensures that the website meets W3C standards, and eliminates the potential of cross browser issues. Upon performing a scan of the website, no issues were found, this was reaffirmed through manual checking using the browsers listed above.

15.1.6. *Emulator Screen Size Testing Results*

The majority of issues found when testing using the Android emulator were based on logic issues concerning APK installations which are trivially fixed, although the most concerning problem surrounded the wide range of screen sizes. Within Android, screen sizes are categorized as "x-large", "large" etc. As of 28th March 2018, screen sizes of emulated Android devices can range from 2.7" up to 10.05". On some larger devices, elements can be positioned incorrectly and appear visually stretched due to poor design considerations. To test compatibility on a range of devices, emulator devices were then chosen based on differing screen and resolutions. These were: 2.7" QVGA (240x320), Nexus 10 (2560x1600), Nexus 5X (1080x1920) and the Pixel XL (1440x2560). After visually inspecting all activities within the Android application, no scaling issues were found, with visual fidelity achieved over all tested devices. This was mainly accomplished through the widespread usage of reliable layouts Android applications can utilize, such as FrameLayout and RelativeLayout which don't rely on absolute positioning on a pixel basis that would result in differing visual representations.

15.1.7. *Fingerprinting Accuracy Tests*

Finally, to assess the accuracy of the two fingerprinting methods, additional testing was performed. Firstly, to reassert the accuracy claims made in an earlier analysis of the algorithm, the Wi-Fi fingerprinting algorithm was tested in a scenario in which two adjacent rooms where subjected to (N=30) samples each under two aliases, “Room #1” and “Room #2”. The device was then placed in each room, alternating N=20 times. At each interval, reports were obtained from the device to establish which room the algorithm determined the device to be in. These results were then analyzed, whether the location was accurately guessed by the algorithm (a correct classification) or not (a generalization error). Naturally, for this test, a physical Android device was used (a Samsung Galaxy S8).

Alias	**Correctly classified**	**Incorrectly classified**
Room #1	9	1
Room #2	10	0
Total	**19**	**1**
Accuracy	**95% (19/20 Correct)**	

Table 10. Wi-Fi fingerprinting test results

Reflecting upon the initial requirements set out, an obtained accuracy of N=95% adequately satisfies the requirement: “Indoor accuracy: the application should correctly report the room the device resides within 90% of the time”.

Moving onto the Bluetooth fingerprinting functionality, a different method must be applied to quantify the accuracy. For this, dynamic calibration (calibrating at different distances) was performed based on previous tests affirming its accuracy over other calibration approaches. Calibration was performed on a singular Bluetooth beacon device at distances N=25cm, 50cm, 75cm, 100cm, 125cm. The Android device was then placed at the calibrated distances and the reported distance recorded.

Distance (CM)	Reported Distance (CM)	Difference (CM)	Accuracy (%)
25	23.977	1.023	96
50	52.678	2.678	95
75	75.908	0.908	99
100	101.245	1.245	99
125	123.674	1.326	99
Average		**1.436**	**97.6**

Table 11. Table of Bluetooth fingerprinting test results

As demonstrated through the average and singular accuracies calculated, this method also satisfies the accuracy requirement of 90% as well as providing indoor location predictions and an "indoor localisation method" also outlined in the initial requirements specification.

15.1.8. *Requirements Satisfaction*

Requirement	Validated by test(s)	Reasoning
The system will provide user location predictions within an outdoor/indoor environment	37	The GPS provider allows the localisation of a mobile phone outdoors, the response can then be obtained without internet connectivity through the usage of SMS
The system will provide a trigger customisation utility	1, 2	The text input dialog opened when the user presses on a trigger preference allows the customisation and saving of trigger phrases
The system will be able to accept a trigger/phrase over at least one medium and perform the associated action based on	23, 24, 25, 26, 29, 30	The application facilitates the communication with a lost device over multiple channels

the trigger		
The system will provide an indoor localisation method	Fingerprint Accuracy Tests	The Wi-Fi fingerprint alias determined to be the likeliest location of the device can be returned over all communication channels. Distance from a calibrated Bluetooth device can also be returned over these channels.
The system will be able to respond to select triggers over a two-way communication channel	24, 26, 30, 40	The application provides a response method/functionality for each communication channel, including SMS, GCM and email.
The system will make use of all pre-existing Android location methods	37, 24, 5	The application allows the user to customise the order of usage of the existing Android location providers when a location fix is requested by an external user.

Table 12. Confirmation of requirements satisfaction

15.1.9. *Acceptance Criteria Validation*

Acceptance Criteria	**Validated by test(s)**	**Reasoning**
The application interface will be aesthetic and user-friendly	1, Emulator Screen Size Testing Results, Website Cross-Browser	The fidelity of the application over multiple screen sizes and resolutions ensures the application is aesthetic and functional
The application should support the two fingerprinting methods previously disclosed	Fingerprinting Accuracy Tests	The application provides two fingerprinting methods, Wi-Fi and Bluetooth, the accuracy of which is attested within tests
The application should have a	32, 33, 34, 35, 36	The activities associated with

user-friendly and instructional training phase for both fingerprinting methods		fingerprinting are presented to the user with informative instructions
The application should include a positioning phase to complement the training phase	32, 33, 34, 35, 36	The application provides two training (calibration) activities, one for Wi-Fi and one for Bluetooth
Indoor resolution: the application should correctly report the location to a resolution of a standard room dimension	Fingerprinting Accuracy Tests	The resolution of Wi-Fi fingerprinting is defined to one alias (location), the resolution is therefore dependent on the user's training. Bluetooth is shown to be accurate to approximately 1cm.
Indoor accuracy: the application should correctly report the room the device resides within 90% of the time	Fingerprinting Accuracy Tests	Wi-Fi fingerprinting correctly classifies the room the device resides within ~97% of the time

Table 13. Acceptance criteria validation

Whilst the tests carried out in this report do not exhaustively test the implementations from all possible perspectives, they confirm that the core functional and non-functional requirements of the implementation have been achieved. These core functions coalesce to create an Android application with satisfies all the initial requirements and affords the user additional functionality, such as the accounts feature. Although not all possible tests have been completed, this system has been evaluated adequately as a summation of all subsystems that complement and enable features within the Android application, hence, intricate server load-testing and other extensive and verbose testing has been excluded.

15.1.10. *Discussion*

This section will discuss the test results presented within the previous section. This section provides an analysis of the tests performed and verification that the criteria for the project has been met.

The application tests verified that the application functions worked correctly when provided with acceptable information. This included tests measuring system response when triggers are received, but also certain GUI options and fingerprinting functionality, the accuracy of which is later tested. The results showed that the application responded as expected/desired on

the vast majority of tests, this is largely due to the comprehensive testing of these components on implementation. The only test which failed to produce a valid result was that of an expected SMS response containing a large amount of information. Although no exceptions or crashes were diagnosed making debugging a much harder and longer process, the fix involved splitting these large messages in segments such that the Android API would not refuse to send them, a mechanism assumingly in place to prevent excessive SMS usage by applications.

The second subsystem, the web interface, was a much easier implementation to test, not being a complex component of the system, much of this testing revolved around the webpage's handling of data retrieved (or not) from the database and the transmission of GC messages. This testing also extended to the additional fields that can be inserted into a custom GC message as well as confirmation from the device that the trigger sent caused an action to occur within the device. This testing ensured that the messages being sent to the device were parsed and subsequently elicited the correct response for that specific phrase. Also tested was the device's consideration of additional fields within the payload when appropriate, such as triggers that read saved presets from SharedPreferences. The tests adequately proved that the interface is able to correctly send fully customised messages to Android devices using the related Android API key and device token of the device. The response received as a consequence of sending response triggers to the device also affirmed that the listeners (scripts) within the webpage are capable of receiving messages sent to the token generated on the webpage load. This, in turn, allowed the tester to deduce that, as the web interface is able to receive responses, the Android application is able to parse the "sender" device token from the message payload to formulate a response and send using the fetched web API key. These tests were more informative out of all others performed, testing functionality of all subsystems.

The next stage of testing was done to test the website's fidelity of presentation across some of the currently popular browsers. Not only did this extend to finding errors within the webpage that would impact functionality, such as the non-transmission of messages, but the correct display of the web interface components. The website appeared consistent over browsers tested, this is mainly due to the fixed alignment of elements on the webpage, rather than absolute positioning that may produce different experiences at varying screen sizes and resolutions, a factor not considered during testing. In addition, HTTP warnings and errors were not discovered using the validator described. This affirms that the web page is free of systematic flaws and that any faults in the overall system can be largely be abstracted away from this subsystem.

Emulator testing was a repetitive but necessary phase to ensure a linear experience across multiple device types, screen sizes and resolutions. Fortunately, due to Android's layout scaling solution, density pixels, which considers pixel density and resolution, the activities within the application were consistent over all devices tested. This was, in part, due to the careful choice of default Android layouts used in the display of GUI components.

The Wi-Fi fingerprinting accuracy tests produced largely similar results to those within the dedicated chapters. Although the testing parameters deviated slightly so as to properly stress the algorithm over a larger number of calibrations and reports. The reports culminated in accuracy

exceeding that previously achieved experienced. It was observed in earlier tests that the accuracy converged at one hundred per cent around (N=10) calibrations. This test confirmed the assumption that equal or higher levels of accuracy are obtainable based on a higher number of samples. The same was apparent with Bluetooth fingerprinting, achieving an average of 97.6% accuracy, although, as previously stated, such results may only be obtainable with specialised Bluetooth devices, including Bluetooth Low Energy beacons employed within this testing phase.

The rest of testing was done to conclude the satisfaction of the functional and non-functional requirements and the acceptance criteria outlined in the previous section.

16. Conclusion

The work conducted in this project has been successful in developing a capable Android application with numerous features to remotely cause the execution of device actions and obtain informative responses. This includes the unique feature of being able to send trigger messages from the web interface to a remote “lost” device from any physical location. The functional and non-functional objectives and criteria have been satisfied as highlighted within testing and discussion sections respectively.

With Find My Android, the project has resulted in a system, created from nothing, which combines multiple implementations and subsystems, such as the accompanying database and web interface, to form a comprehensive remote control and feedback application. These features include offering multiple communication methods for all Android device variants (SMS for cellular-enabled devices, GCM and email). The usage of common and standardised messaging protocols/services ensures compliance and overall effectiveness of the system over current and future devices. One of the primary outcomes from this project was the creation of an indoor localisation method. The Wi-Fi fingerprinting method is able to reduce the resolution (precision) to that of a room as defined in the initial requirements, although greater precision may be available with a different classification model developed over a large timescale. However, one disadvantage of the alternate indoor localisation method (Bluetooth) is that the classification is supervised (the user performs the virtual calibration of a device), the training data provided being the distance away from device. This method proved to work very well for calculating distances away from individual devices, especially with Bluetooth Low Energy Beacons. However, it is a time consuming and laborious task as a method of training, involving the wide experimentation in the number of samples. Moreover, due to the differences in signal strength with consumer Bluetooth devices, which prioritise signal strength and connectivity, it is largely ineffective at predicting distance away from these devices. A better solution would involve using a probabilistic model. Such a model would mitigate the ineffectiveness of dealing with these differing devices. However, to achieve the same fine-grain positional accuracy (in order of centimetres), the process would still need supervision to create the model. A more general approach would be to provide an additional unsupervised variant which would simply return the room in which the device resides within, with a resolution comparable to the previous fingerprinting method, although the benefit of using the method as a replacement would be minimal for a user desiring pinpoint accuracy indoors with an array of devices. For future

development, as there is a limit to the communication methods that can be employed, with the simplest and most convenient method probably mimicking something similar to the GCM solution already implemented. Other than the other algorithmic additions previously stated, extensions for the project would inevitably extend to the database's contents and data records. This could include a list "history" of locations declared by the device, such that a map of all visited locations could be produced from that list for the users to view, along with dates declared. This would be an addition to the existing web interface, which already facilitates communication to the database contents. This would be extremely useful with stolen devices, but would be largely redundant for a stationary device. Moreover, the inclusion of cloud messaging could extend to this feature, sending the new declared location updates from the database to any authenticated users, allowing the device owner to visualise location changes in real time. The inclusion of new features could permit "premium" marketing, selling desirable options as unlockable paid features to accompany the application. Although, due to some of the security features within Find My Android, such as the prevention of app uninstallation without the device password, the marketing and selling of the application is not permitted on the default Android application store (Google Play). Due to these policy restrictions it may be lucrative and advantageous to remove certain non-essential features in order to list the application on the store which, described within the background section of this report, is a popular and growing market.

While challenging, implementation of compatibility with the GCM format and paradigm has proved successful in providing Find My Android with a scalable communication solution that allows many different users to use the method simultaneously, both within the application and the web interface. The web interface being hosted on a managed and established platform (GitHub) also ensures a high level of traffic will not adversely affect the functionality of the overall system.

Find My Android has also demonstrated how communication methods and associated broadcast receivers can be used to covertly relay data and perform system actions without the viewer of the device being able to conceivably notice, useful in stolen-device scenario. Without this behaviour, the usage of this app would be extremely limited. For example, if certain actions required user interaction with the device, a lost device may never be recovered. This would be especially true if the device in question were to be stolen, in such a scenario the thief could deny system actions and uninstall the application. However, certain functions that were previously available for use through APIs, and are present in older relevant device-recovery applications, are now depreciated and require user interaction at runtime. A feature that was conceived for Find My Android was the capturing of pictures from the device's camera for relaying to an external user, providing a visual context to location reports. Unfortunately, such a process was found to not be possible due to the runtime permissions nuance present in later versions of Android (6.0+), requiring a user to be present to confirm events on the device. A future version of Find My Android could perhaps use external libraries (third party libraries) in an attempt to achieve this functionality in a streamlined manner and in accordance with Find My Android's silent operation.

Overall, Find My Android is a robust product with significant potential to be refined and enhanced further whilst still maintaining its focus on the primary use cases of the application, including locating a stationary lost device and performing covert and undetectable surveillance on a stolen one. To offer a broader range of compatibility, an IOS variant of the application could be developed in future, increasing the user base of the application and the scope of potentially introducing OS-specific features to widen the capabilities of the application where possible. However, in its current form, Find My Android offers a large amount of abstraction, as much as possible, from the complex mechanisms and underlying processing occurring in the application in presenting the system in a simple, aesthetic and easy to use form, with no advanced or laborious setup required.

17. Project Commentary

The initial requirements to satisfy the completion of this project were outlined in the Project Initiation Document (PID), present in the Appendix, which was completed before project launch. Within this document, the key implementation and modules of the system were listed, and a desired timeline of what was to be required and completed at several phases of the project produced. This is known as the "waterfall" approach, in which a project is scrutinised for shortcomings and is planned well before it is undertaken. Whilst so called "Agile" development has its benefits, it does not translate well with development undertaken by an individual.

Fortunately, the development of the project was completed far ahead of schedule. The user interfaces/GUI and rudimental triggering methods, including email triggering, were designed and completed very early on in the academic year, ensuring that the bulk of the project implementations could be developed and inserted into the project much sooner than originally planned. This was advantageous from a time-management perspective because the additional implementations, including those that required algorithm design and independent thought took much longer than planned. However, there were no significant detrimental issues throughout this project and it exceeded the objectives quickly through the scheduled time periods. Testing all the subsystems produced as part of the cohesive solution took much longer than the implementation of some features themselves, however all key features were completed in good time. This time allowance permitted additional features to be added to the project, including the database of location data.

18. Reflection

This project proved to be a logical and time-consuming challenge at certain points, especially the response method employed to accompany the GCM triggering mechanism. However, the final system has achieved all the objectives previously noted as well as achieving additional functionality.

Throughout the undertaking of this project, I have developed many existing skills in software development and project management. My level of new and present knowledge of the Android

operating system has increased greatly, such as making aesthetic, fast, and error free applications, as well as furthering my Java programming language knowledge, including nuances belonging to newer versions of Java. Moreover, my project management skills, initially developer earlier in university education, have been developed immensely during this project. The inclusion of a logbook complemented the usage of source control to effectively track progress, changes and issues for documentation, this coupled with being a great backup solution for all related project files.

Many challenges were presented throughout this project which demanded large amounts of logical thinking, rather than developing. Although this was important to ensure that a large portion of time was not used implementing redundant or broken features, the time spend developing innovative and new features was extremely time consuming and often involved collating knowledge about entire systems and mechanisms needed to implement the solution. Due to limited existing examples of these features in the wild, there are frequent times when trial and error took precedence over properly diagnosing the solution, in the interest of time and achieving a quick fix. This notably occurred during the development of the aforementioned GCM response method, as there was no examples and extremely limited API documentation at the time of development. It was abandoned after a significant length of time, only to be revisited and a solution made after discovering the necessity of having two separate API keys.

If I were to undertake the project again, I would have attempted to observe more existing applications to improve upon frequently used methods and features. In addition to this, I would prepare more thoroughly in terms of which implementations would take the longest and which may not be feasible during the timescale of the project. Moreover, I'd endeavour to be more active during the development phase to develop and test features in a faster and concurrent manner rather than performing some tasks sequentially. I feel strongly that this approach would result in more features being developed, with the completion of the project being achieved in a timelier manner. This modularity I believe would also result in less optimisation being necessary at the end of the project, with less redundant code being produced.

Overall, I am satisfied with the success of this project and the final system and implementations which have been produced as a result of its undertaking. Find My Android has large potential to be deployed onto a market such as Google Play Store with minimal alterations and even sold commercially, with further work and additions that can be introduced in the near future.

19. Social, Legal, Health & Safety and Ethical Issues

This application should only be installed on a device that is to be operated solely with the owner's consent. Installation on an unknowing party's device to remotely wipe, relay sensitive information, or cause havoc, would constitute misuse under the Computer Misuse Act of 1990 and carries a maximum penalty of 10 years imprisonment [70]. The retrieval of user information from the database requires authentication via a username and password pair, passwords are stored in a secure, hashed form, and no data is sold or handled by any third parties. Additionally, no identifiable information is contained within the database that would violate the privacy of users and reveal identities. These considerations were not highlighted in the initial literature, including the Project Initiation Document, but have been consolidated as a consequence of the additional implementations developed.

20. References

[1] DONG NGO, "Celebrating 10 years of GPS for the masses," [Online]. Available: https://www.cnet.com/news/celebrating-10-years-of-gps-for-the-masses/. [Accessed 3 March 2018].

[2] S. B. X. X. Mingjing Guo, "Research on Location Arithmetic in Multipoint Cooperative Positioning Model about LBS of Mobile Campus," IEEE, Wuhan, Nanchang, 2017.

[3] G. B. T D'Roza, "An overview of location-based services," *BT Technology Journal* , vol. 21, no. 1, 2003.

[4] C. Steinfield, "The Development of Location Based Services in Mobile Commerce," Lansing.

[5] Berg Insight, "GPS AND MOBILE HANDSETS," 4th ed., 2009.

[6] M. Y. Mohamed Ibrahim, "A Hidden Markov Model for Localization Using Low-End GSM Cell Phones," IEEE Communications Society, Egypt , 2011.

[7] B. L. K. Z. C. R. Z. Z. Lina Chen, "An Improved Algorithm to Generate a Wi-Fi Fingerprint Database for Indoor Positioning," *Sensors,* vol. 13, no. 1, 2013.

[8] Z. D. Rajan Khullar, "Indoor Localization Framework with WiFi Fingerprinting," New York.

[9] S. C. Zihao Li, "An Adaptive Hybrid Indoor WiFi Fingerprinting and Propagation Parameter Estimation Using RANSAC LASSO Regression," Guangzhou.

[10] Google Inc., "SignalStrength," [Online]. Available: https://developer.android.com/reference/android/telephony/SignalStrength.html. [Accessed 27 March 2018].

[11] M. W. Appala Chekuri, "Automating WiFi Fingerprinting Based on Nano-scale Unmanned Aerial Vehicles," IEEE, Brookings, 2017.

[12] S. B. A. Y. A. F. M. Wondimu K. Zegeye, “WiFi RSS Fingerprinting Indoor Localization for Mobile Devices,” Baltimore.

[13] Google Inc., “Activity,” [Online]. Available: https://developer.android.com/reference/android/app/Activity.html. [Accessed 16 January 2018].

[14] Google Inc., “Fragments,” [Online]. Available: https://developer.android.com/guide/components/fragments.html. [Accessed 10 January 2018].

[15] Google Inc., “App Manifest Overview,” [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html. [Accessed 4 February 2018].

[16] Google Inc., “Intent,” [Online]. Available: https://developer.android.com/reference/android/content/Intent.html. [Accessed 24 February 2018].

[17] Google Inc., “<intent-filter>,” [Online]. Available: https://developer.android.com/guide/topics/manifest/intent-filter-element.html. [Accessed 1 February 2018].

[18] Google Inc., “AsyncTask,” [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask.html. [Accessed 24 January 2018].

[19] Google Inc., “SharedPreferences,” [Online]. Available: https://developer.android.com/reference/android/content/SharedPreferences.html. [Accessed 1 February 2018].

[20] ISOCPP, “Serialization and Unserialization,” [Online]. Available: http://isocpp.org/wiki/faq/serialization. [Accessed 23 January 2018].

[21] Google Inc., “"A Java serialization/deserialization library to convert Java Objects into JSON and back",” [Online]. Available: https://github.com/google/gson. [Accessed 23 January 2018].

[22] University of Ottawa, [Online]. Available: http://www.site.uottawa.ca/~rhabash/GPS1Introduction.

[23] Google Inc., “LocationManager,” [Online]. Available: https://developer.android.com/reference/android/location/LocationManager.html#NETWORK_PROVIDER. [Accessed 1 January 2018].

[24] “Number of available applications in the Google Play Store from December 2009 to December 2017,” [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [Accessed 19 Janurary 2018].

[25] “Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016,” Gartner, [Online]. Available: https://www.gartner.com/newsroom/id/3609817. [Accessed 15 January 2018].

[26] Google, “Dashboards,” [Online]. Available: http://developer.android.com/resources/dashboard/platform-

] versions.html. [Accessed 1 January 2018].

[27 Y. D. Liang, Intro to Java Programming, Comprehensive Version, Pearson, 2013.
]

[28 PHP, “What is PHP?,” [Online]. Available: https://secure.php.net/manual/en/intro-whatis.php. [Accessed 4
] January 2018].

[29 D. R. Brooks, An Introduction to HTML and JavaScript, Springer, 2010.
]

[30 L. Bassett, Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON, O'Reilly, 2015.
]

[31 . W. S. M. Elliotte Rusty Harold, XML in a Nutshell (In a Nutshell (O'Reilly)), O'Reilly, 2004.
]

[32 Google Inc., “LocationManager,” [Online]. Available:
] https://developer.android.com/reference/android/location/LocationManager.html. [Accessed 2 January 2018].

[33 Google Inc., “LocationManager, getLastKnownLocation,” [Online]. Available:
] https://developer.android.com/reference/android/location/LocationManager.html#getLastKnownLocation(java.l
ang.String). [Accessed 1 January 2018].

[34 “Tracking a suspect by mobile phone,” August 2005. [Online]. Available:
] http://news.bbc.co.uk/1/hi/technology/4738219.stm.

[35 OpenCelliD, [Online]. Available: https://opencellid.org.
]

[36 C. M. Magnus Olsson, “EPC and 4G Packet Networks 2nd Edition,” 2013, p. xxvii–xxxvi.
]

[37 Google Inc., “TelephonyManager,” [Online]. Available:
] https://developer.android.com/reference/android/telephony/TelephonyManager.html.

[38 Google Inc., “CellIdentityGsm,” [Online]. Available:
] https://developer.android.com/reference/android/telephony/CellIdentityGsm.html. [Accessed 28 December
2017].

[39 Bluetooth Inc., “Bluetooth Brand Guide & Logos,” [Online]. Available: https://www.bluetooth.com/develop-
] with-bluetooth/marketing-branding/brand-guide-logos. [Accessed 26 February 2018].

[40 C. Bisdikian, “An Overview of the Bluetooth Wireless Technology,” *IEEE Communications Magazine,* pp. 1-2,
] December 2001.

[41] S. S. M. M. M. H. a. D. D. U.L.Muhammed Rijah, “Bluetooth Security Analysis and Solution,” *International Journal of Scientific and Research Publications,* vol. 6, no. 4, pp. 1-3, April 2016.

[42] Google Inc., “Bluetooth,” [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth.html.

[43] Google Inc, “Bluetooth Low Energy,” [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html.

[44] D. K. C. B. Joonyoung Jung, “Distance Estimation of Smart Device using Bluetooth,” Deajeon, Korea, 2013.

[45] A. M. Petros Spachosa, “Energy Efficiency and Accuracy of Solar Powered BLE Beacons,” Guelph, 2017.

[46] O. H. a. Y. C. Yan Luo, “Enhancing Wi-Fi fingerprinting for indoor positioning using human-centric collaborative feedback,” in *Human-centric Computing and Information Sciences*, Springer.

[47] M. Sauter, From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband, John Wiley & Sons, 2011.

[48] G. Retscher, “FUSION OF LOCATION FINGERPRINTING AND TRILATERATION BASED ON THE EXAMPLE OF DIFFERENTIAL WI-FI POSITIONING,” ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vienna, 2017.

[49] Carnegie Mellon University, “Concept of Hashing,” [Online]. Available: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Hashing/hashing.html. [Accessed 27 February 2018].

[50] M. Rouse, “redundancy,” [Online]. Available: http://whatis.techtarget.com/definition/redundancy. [Accessed 3 March 2018].

[51] Wikipedia, “Phoning home,” [Online]. Available: https://en.wikipedia.org/wiki/Phoning_home. [Accessed 3 March 2018].

[52] PHP, “password_hash,” [Online]. Available: https://secure.php.net/manual/en/function.password-hash.php. [Accessed 3 March 2018].

[53] The Institute of Electrical and Electronics Engineers, Inc, “IEEE Standard for Binary Floating-Point Arithmetic,” The Institute of Electrical and Electronics Engineers, Inc, New York, 1985.

[54] PHP, “mysqli_fetch_row,” [Online]. Available: https://secure.php.net/manual/en/mysqli-result.fetch-row.php. [Accessed 3 March 2018].

[55] PHP, “password_verify,” [Online]. Available: https://secure.php.net/manual/en/function.password-verify.php. [Accessed 3 January 2018].

[56] JQuery, “jQuery.get(),” [Online]. Available: https://api.jquery.com/jquery.get/. [Accessed 20 January 2018].

[57] Google Inc., “ACTION_BATTERY_CHANGED,” [Online]. Available: https://developer.android.com/reference/android/content/Intent.html#ACTION_BATTERY_CHANGED. [Accessed 13 February 2018].

[58] Google Inc., “AsyncTask,” [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask.html. [Accessed 12 January 2018].

[59] Oracle, “Package javax.mail,” [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/mail/package-summary.html. [Accessed 3 March 2018].

[60] PHP, “mail,” [Online]. Available: https://secure.php.net/manual/en/function.mail.php. [Accessed 3 March 2018].

[61] S. B. S. R. S. D. B. Suman Bhattacharjee, “Best-effort delivery of emergency messages in post-disaster scenario with content-based filtering and Priority-enhanced PRoPHET over DTN,” International Conference on Communication Systems and Networks (COMSNETS), Kolkata, INDIA, 2016.

[62] H. S. G. P. A. S. C. PAUL GRIFFITHS, “The Global System for Mobile Communications Short Message Service,” DIALOGUE COMMUNICATIONS LTD., SHEFFIELD, 2000.

[63] “Time to Confirm some Mobile User Numbers: SMS, MMS, Mobile Internet, M-News,” 13 January 2011. [Online]. Available: http://communities-dominate.blogs.com/brands/2011/01/time-to-confirm-some-mobile-user-numbers-sms-mms-mobile-internet-m-news.html. [Accessed 23 January 2018].

[64] “SMS Introduction,” [Online]. Available: http://www.smssolutions.net/tutorials/misc/intro/. [Accessed 3 March 2018].

[65] Google Inc., “<intent-filter>,” [Online]. Available: https://developer.android.com/guide/topics/manifest/intent-filter-element.html#priority. [Accessed 12 January 2018].

[66] Google Inc., “createFromPdu,” [Online]. Available: https://developer.android.com/reference/android/telephony/SmsMessage.html#createFromPdu(byte[],%20java.lang.String). [Accessed 2 January 2018].

[67] Google Inc., “SmsManager,” [Online]. Available: https://developer.android.com/reference/android/telephony/SmsManager.html. [Accessed 12 January 2018].

[68] Google Inc., “Google Cloud Messaging: Overview,” [Online]. Available: https://developers.google.com/cloud-messaging/gcm. [Accessed 3 March 2018].

[69] W3C, “Markup Validation Service,” [Online]. Available: https://validator.w3.org. [Accessed 30 March 2018].

[70] "Computer Misuse Act 1990," [Online]. Available: http://www.legislation.gov.uk/ukpga/1990/18/contents. [Accessed 24 Feburary 2018].

21. Appendix

21.1. *PID (Project Initiation Document)*

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**24025024**
Student Name	**Oliver Bathurst**
Email	**ft025024@live.reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**Computer Science BSc**
Supervisor Name	**Patrick Parslow**
Supervisor Signature	
Date	**24/09/2017**

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	250
1.2	**Project Title**
	The usage of multiple positioning techniques to better ascertain an individual's cellular location
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	Utilising the multiple positioning techniques available for Android devices as well as more unconventional approaches to improve accuracy over the standard.

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Oliver Bathurst, BSc CS, o.bathurst@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
	Pat Parslow, P.Parslow@reading.ac.uk
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
1.8	**Company Address**
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**

2

SECTION 2 – Project Description

2.1	**Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**
	Background research will include in-depth analysis regarding the accuracy and limitations of all positioning techniques such as WiFi, GPS, etc as well as the underlying mechanisms and some primary research when applicable. The rest of the research will involve exploring other methods of positioning that may or may not use additional hardware other than the mobile phone to improve accuracy. Other research may involve attempting to solve the notoriously inaccurate indoor positioning problem via data mining WiFi fingerprinting.
2.2	**Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.
	Tasks: -app that utilises at least GPS, WiFi, Passive positioning -add at least two more positioning techniques that will potentially improve accuracy and provide more information about the mobile's whereabouts -responds to SMS/Emails and relays position -app will create alerts also and send warning SMS/Email messages Milestones: -SMS/Email listeners added to relay positions when keywords are intercepted -google maps API added to the app -at least one unique positioning method incorporated into the app -app created Deliverables -extensive documentation and report -one Android application, and potentially a piece of hardware if applicable to the objective
2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.

3

2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval?**
	N/A
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	N/A
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**

4

	N/A

5

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1	**Background Research**		
1.1	Research	3	Extensive knowledge gained, understanding of techniques used in Android OS
1.2	Creation of visual aids	2	graphs and diagrams for the final report
2	**Analysis and design**		
2.1	GUI and app design	4	Sketches and app layout
2.2	Code design	3	What services, listeners, and classes will be responsible for what
2.3	Listeners	1	Creation of Broadcast Receivers (listeners) to listen for key phrases in SMS and emails to relay back the devices position
3	**Develop prototype**		
3.1	Developing app	4	App created
4	**Testing, evaluation/validation**		
4.1	unit testing	1	
4.2	Extensive testing on multiple devices (emulators)	1	Bug reports (if any), fixes, noted in the report
			…
5	**Assessments**		
5.1	write-up project report	2	Project Report
5.2	produce poster	0.5	Poster
TOTAL	**Sum of total effort in weeks**	**21.5**	

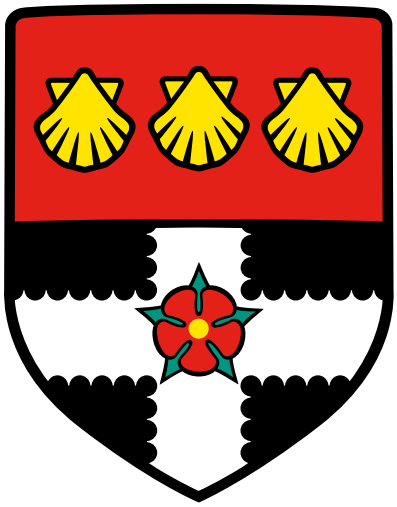
6

SECTION 4 - Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

	START DATE: ../../…. **<enter the project start date here>**												
	Project Weeks												
Project stage	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
Research													
Visual Aids													
2 Analysis/Design													
GUI and app design													
Code design													
Listeners													
3 Develop prototype.													
Developing app													
4 Testing, evaluation/validation													
unit testing													
Extensive testing on multiple devices													
5 Assessments													
write-up project report													
produce poster													

21.2. *Logbook*



SCHOOL OF MATHEMATICAL, PHYSICAL AND COMPUTATIONAL SCIENCES

UNIVERSITY OF READING

OLIVER BATHURST, BSc. Computer Science

Find My Android (The usage of multiple positioning techniques to ascertain an individual's device location and methods of recovery through remote control)

o.bathurst@student.reading.ac.uk

github.com/OliverBathurst/Individual-Project (App)

oliverbathurst.github.io/web (Web Interface)

Contents

WEEK 1 25/09/17 - 2/10/17

- Created starter application, basic template
- Researched APIs within the Android OS
- GPS, Wi-Fi, Passive APIs noted for use
- Researched permissions needed for use
- Location code taken from earlier geolocation project
- Sorted techniques via accuracy and implemented a class with snippets
- Configured Android Studio project, including permissions
- Created main fragment for gMaps integration
- Added fragment for device information
- Added fragment for preferences/settings
- Added permissions check on start up
- Researched Google geofencing API for consideration

WEEK 2 2/10/17 - 9/10/17

- Added beacon activity, added locator preference option
- Added reordering of locators (locator preference)
- Started to add GCM (now FCM)
- Added BLE scanning (Started to)
- Added BLE/Bluetooth scanning, results added to screen
- Added differentiation between Bluetooth types
- Fixed camera not repositioning on location change
- Added local webpage hosting
- Fixed bug where connected Wi-Fi with no internet caused a crash when queried for a location
- Fixed error with location not updating on GUI

WEEK 3 9/10/17 - 16/10/17

- App now can act as a server, serves a google maps page accessible from any computer on the phone's network, cleaned up code, optimized and fixed a random bug.
- Added more information to local server page

- Fixed Device Admin bug not activating device administrator
- Added option to prevent uninstall (Lock screen password/method required to uninstall app)
- Small code optimizations, shedding off lines and useless variables
- Fixed bug with app crashing on null context, fixed bug where it was declaring the wrong provider used with hardcoded text, location preference reflected on GUI
- Fixed potential bug, optimized some code
- Added Bluetooth beacon information to the scanning segment
- Fixed bug where "null" Bluetooth devices were being added

WEEK 4 16/10/17 - 23/10/17

- Fixed potential bug, optimized some code
- Added Bluetooth beacon information to the scanning segment
- Fixed bug where "null" Bluetooth devices were being added
- Lots of code stripping/optimizations
- Lots of code stripping/optimizations part 2
- Code stripping/optimizations part 3
- Bluetooth beacon objects are now saved to shared preferences for later reference
- Optimized imports and altered Bluetooth beacon toolbar (drawer)
- Added beacon deletion selection
- Deleted unnecessary code/ imports
- Fixed bug with null Bluetooth devices
- Turned off notifications by default, added SMS trigger to report stolen
- Cleaned up code
- Flagging device as stolen by email now possible
- Removed useless GPS elevation text
- Cleaned beacon layout
- SMS trigger to report stolen, 'un-flag as stolen' option, geofence and battery flare only now fire when device is marked stolen
- Fixed bug where the Bluetooth array in shared prefs returns null causing errors, added checks
- Added beacon configure activity, auto cancelling of reorder activity
- Finished beacon configure page design
- Finished ‘calibrate’ button
- Added scanning of RSSI and error handling
- Finished beacon activity and cleaned code, added ability to reset a beacon's calibration.

WEEK 4 23/10/17 - 30/10/17

WEEK 5 30/10/17 - 6/10/17

- Fixed potential null pointer exceptions
- Fixed potential null pointer exceptions pt2
- Cleaned and reduced code
- Converted network tasks on main thread to Asynctasks
- Code refinements
- Moved one method for other classes to use
- Added remote beacon information relay
- Finished beacon information relay from email/SMS
- Abstracted email credential checking to email helper class
- Generally reducing object creation
- Added method to get app version
- Removed unused code/imports
- Combined two classes to reduce object creation/general object creation reductions
- Fixed bug in Asynctasks with looper, fixed null context in email sender
- Added option to delete email after an action has been triggered (Gmail triggering)
- Edited login screen to prevent leak, added to info page
- Fixed bug with display colours, Bluetooth beacon scanning now faster with reduced operations/ object creation, email-triggered beacon scanning and reporting working and tested
- Removed useless methods/code, optimised imports
- Added sign-up page and web interface button
- Added GCM
- Added GC message receiving ready for triggers
- Started making web interface
- Edited theme
- Added manual latitude/longitude lookup
- Added more information to front page, cleaner aesthetic
- Cleaned up unused code
- Separated CSS files
- Fixed map bug

WEEK 6 30/10/17 - 6/11/17

- Code refinements
- Moved one method for other classes to use
- Added remote beacon information relay
- Finished beacon information relay from email/SMS
- Abstracted email credential checking to email helper class
- Generally reducing object creation
- Added method to get app version
- Removed unused code/imports
- Combined two classes to reduce object creation/ general object creation

- Fixed bug in AsyncTask with looper, fixed null context in email sender
- Added option to delete email after an action has been triggered
- Edited login screen to prevent leak, added to info page
- Fixed bug with display colours, bluetooth beacon scanning now faster
- Removed useless methods/code, optimised imports
- Added sign-up page and web interface button
- Added GCM
- Added GC message receiving ready for triggers
- Added method and user option to get the current GCM token

WEEK 7 6/11/17 – 13/11/17

- Added GCM Handler
- Sending messages from browser are now received
- App now validates GC message before considering anything else
- Optimised string resources, added the first GCM trigger option
- Optimised more string resources, added ring trigger for GCM
- Deleted duplicated code, finished extracting strings
- Fixed exception when scanning for bluetooth devices
- Now allows user to reject turning bluetooth on for beacon activity
- Shortened SettingsFragment code
- Reordered and neatened fragment
- Optimised settings fragment, reduced code usage, neatened onCreate
- Changed commits on the fragment manager to commitNow, no white flashes on fragment change on devices running Android 7.0 or above

WEEK 8 13/11/17 – 20/11/17

- Fixed bug when navigation item was checked before fragment finished loading
- Added sender to send upstream messages
- GCM handler can now parse bundles for extra data
- Added email and SMS sending from GC message, can accept extra data fields

WEEK 9 20/11/17 – 27/11/17

- Added cell info analyser activity
- Added cell tower map activity
- Added support for neighbouring networks
- Cell tower locator now works
- Neatened code and imports
- Added support for CDMA
- Added helper class for cell tower location and information retrieval for email and SMS bodies.
- Cut out unnecessary code in settings fragment, added torch class
- Implemented torch toggle, added cell tower information to email/SMS toggle

WEEK 10 27/11/17 - 4/12/17

- Added torch toggle for SMS, retrieval of GCM token by email/SMS

- Added GCM response class
- (Website) Added service worker registration bug fix
- (Website) Parses GC message payload sent to webpage
- Custom broadcast sent on receipt of certain GCMs to make receiver reply to GC messages
- Added alternate mail sender server (PHP) and added class to post data to it, will replace Gmail dependency
- Completed rollover to new PHP mail system, more reliable and saves lines of code
- Added exception handling
- Formatted strings for better web display
- Fixed bug where only one GCMRelay Asynctask could be executed per launch of the app.

WEEK 11 4/12/17 - 11/12/17

- Removed direct connection to mail server and added connection to redirect/resolver service, mail server can be changed without updating app.
- Added helper classes
- Added more information in login page
- Added spaces for better email display
- Added cell tower info to SMS response
- Added GCM triggers to get call and contacts logs, added log class, can now specify how many calls to retrieve for the 4kb limit of gcm
- Deleted useless trigger, neatened GCM class
- Fixed bug with battery flare feature, can now send multiple emails if the battery is recharged and drained below the level again
- Removed unused resources, added new logo, removed notification option
- Removed unused preferences
- Added location broadcast receiver for updating of database
- Removed unnecessary trigger
- Added user sign up to app, gets response from server and prints to user, added background location sending to server.
- Added protected character for web parsing
- Fixed typo
- Reordered preference fragment to avoid confusion
- Added accuracy to URL update params
- Added cell tower GCM relay
- Added new URL syntax, added username length requirement
- Updated mail server URL
- Created database updater class
- Added auto-updater
- Added Wi-Fi Fingerprinter and helper class

- Fixed data structure syntax that only allowed one pair to exist in Fingerprint helper class
- Finished fingerprinting helper class pending review and testing
- Added reset buttons to fingerprinting results
- Added fingerprinting to UI options
- (Website) Testing json parse payload
- (Website) Added PHP mail script
- (Website) Added new line regex replacer
- (Website) Added trimming to reduce user input errors
- (Website) Added new line filtering for better web display
- (Website) Added API key lookup
- (Website) Added API key lookup on start-up

WEEK 12 11/12/17 - 18/12/17

- Reduced general code usage
- Reduced general code usage, removed useless methods, updated Wi-Fi Fingerprinter UI
- Removed unnecessary exception catch blocks, added consistency and fixed potential future user errors
- Fixed bug where activity didn't start unless BT was disabled
- Changed inaccuracy where BT devices were initialised to a default 1 dBm/CM reading instead of 0.
- Reduced object creation
- Renamed classes for accuracy
- Fixed bug where Bluetooth wasn't enabling when needed
- (Website) Removed validator, user can now send from another GCM sender website

WEEK 13 18/12/17 - 25/12/17

- BT Deletion now deletes RSSI/CM keys

WEEK 14 25/12/17 - 01/01/18

- Neatened SMS receiver
- Added "update location on battery level change", updated dependencies
- Changed Manifest, removed useless intent filter
- Added more accurate email deletion
- Added comments, restructured email fetcher class
- Added more conditions for database updater class to execute

WEEK 15 01/01/18 - 08/01/18

- Fixed bug allowing null values to be passed in URL, unnecessary server load
- Removed dependency on useless authentication script

WEEK 16 08/01/18 - 15/01/18

- Updated versions
- Started BT Fingerprinting chapter
- Started writing Cell-ID chapter

WEEK 17 15/01/18 - 22/01/18

- Continuing to complete chapters

WEEK 18 22/01/18 - 29/01/18

- Updated Wi-Fi fingerprinting algorithms, now includes a HashMap and other speed optimisations, added break-early condition for Alias and fingerprint retrieval, updated to lambda expressions, deleted unnecessary imports
- Reduced overall code usage, app now opens to settings page, removed fewer calls in Wi-Fi finger printer method and removed potential error when saving
- Fixed a few logical errors that prevent execution, cleaned code
- Fixed a logical error that halved the reported dBm/CM for a BT device
- General moving around, removed unused resources
- Fixed error message displaying incorrect info

WEEK 19 29/01/18 - 05/02/18

- Added comments, updated API key
- Fixed bug where GCM-triggered BT scanning wasn't replicable
- Added manual location update button for database updating
- Fixed BT not discovery bug, relay delay shortened
- Added Background section to report

WEEK 20 05/02/18 - 12/02/18

- Continuing to complete chapters

WEEK 21 12/02/18 - 19/02/18

- Fixed potential null pointer exception
- Added salt to default triggers
- Added additional chapters
- CellID additions
- Added retrieval info
- CellID conclusion
- Added parsing information
- Added network
- Added location request information
- Added subchapter (Android implementation)
- Added pseudo code to provider choice

WEEK 22 19/02/18 - 26/02/18

- Added function requirements
- Started writing up full report
- Added chapter to dissertation

WEEK 23 26/02/18 - 05/03/18

- Merged the references for BT chapter
- Added chapters into final report
- Added introduction
- Added definition, more testing
- Fixed sub-heading
- Added design chapter

- Added fingerprint intro
- Fixed a bug where large SMS messages weren't being sent
- Added images

WEEK 05/03/18 - 12/03/18

- Added images
- Removed useless files, added to project poster
- Added to poster
- Reorganized, finished poster
- Added 'Share GCM Token' option

WEEK 12/03/18 - 19/03/18

- Added to poster
- Reorganized, finished poster

WEEK 19/03/18 - 26/03/18

- Fixed formatting, added header numbers
- Added figure formatting
- Fixed figure captions
- Added literature review
- Added appendixes
- Formatted figure text

WEEK 26/03/18 - 02/04/2018

- Updated dependencies
- Updated suppressions
- Removed redundant casts
- Reordered GUI items
- Fixed register receiver crash
- Fixed permissions bug
- Android 8.0 breaks broadcasts for some reason, converted a broadcast receiver into a service
- Made public inner classes package-private
- Added testing section
- Added more test cases
- Merged literature review
- Added to literature review
- Added reflection
- Added conclusion

WEEK 02/04/2018 - 09/04/2018

- Finished conclusion
- Added discussion
- Finished report