BSc (Hons) Computing Science

Magnetic Field Indoor Positioning and Navigation for UWS Paisley Campus

Angelo Romel Lopez

B00285812

24th March 2017

Supervisor: Dr Ying Liang

Declaration

This dissertation is submitted in partial fulfilment of the requirements for the degree of BSc Computing Science (Honours) in the University of the West of Scotland.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

Name: ANGELO ROMEL LOPEZ

Signature:

Date: March 24, 2017

Library Reference Sheet

Surname Lopez	
First Name Angelo	**Initials** ARL
Borrower ID Number B00285812	
Course Code COMPSCI	
Course Description BSc (Hons) Computing Science	
Project Supervisor Dr Ying Liang	
Dissertation Title Magnetic Field Indoor Positioning and Navigation for UWS Paisley Campus	
Session 2016/2017	

PROJECT SPECIFICATION FORM

Project Title: Magnetic Field Indoor Positioning and Navigation for UWS Paisley Campus

Student: Angelo Romel Lopez **Banner ID:** B00285812

Supervisor: Dr Ying Liang

Moderator: Dr Mark Stansfield

Outline of Project:

The aim of this project is to develop a native Android app to assists visitors, students and staff that require directions/way-finding to a location or event inside buildings in UWS Paisley campus. The app will use the geomagnetic indoor positioning technology provided by IndoorAtlas. IndoorAtlas provides developers with SDKs and APIs delivered as a scalable cloud platform that allows them to create Android and IOS apps capable of accurately locating a device inside buildings and structures using geomagnetic indoor positioning technology.

Top level view of the project stages:

- Using an image editor, plot the different floor layout/plan of a building to make sure that the scale and alignment is accurate.
- Upload maps to IndoorAtlas server.
- Record magnetic field readings for every location and possible routes within buildings.
- Development of an Android application that displays the map of the campus and the internal layout of the buildings. The application will locate and pinpoint the exact location of the user and provide navigation route to the destination intended by the user.

A Passable Project will:

Using the geomagnetic sensor of the device, the app will accurately pinpoint the location of a user within the Oswald Building (R Block) and South Building (E Block) of the Paisley campus. The app must also detect the change in floor level and display the appropriate map layout. The app must provide navigation route to the intended destination within the two buildings.

A First Class Project will:

In addition to the Oswald and South Buildings, the app will also be able to determine a user's exact location in other buildings. The app must also provide and seamlessly combine both indoor and outdoor navigation throughout the campus and between buildings.

Reading List:

[1] IndoorAtlas “Android SDK/API 2.2.2”
Package com.indooratlas.android.sdk, com.indooratlas.android.sdk.resources
http://docs.indooratlas.com/android/2.2.2/

[2] IndoorAtlas “Mapping Bigger Areas”
https://www.indooratlas.com/2015/09/23/mapping-bigger-areas/

[3] Android Developers “Building Apps with Location & Maps”
(https://developer.android.com/training/building-location.html)

[4] Android Developers “Material Design for Android”
(https://developer.android.com/design/material/index.html)

[5] Android Developers “Geomagnetic Field”
(https://developer.android.com/design/material/index.html)

[6] Android Developers “Position Sensors”
(https://developer.android.com/guide/topics/sensors/sensors_position.html)

[7] Android Developers “Environment Sensors”
(https://developer.android.com/guide/topics/sensors/sensors_environment.html)

[8] Indoor Positioning System
(https://en.wikipedia.org/wiki/Indoor_positioning_system)

[9] A Realistic Evaluation and Comparison of Indoor Location Technologies: Experiences and Lessons Learned
(https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Lymberopoulos.pdf)

[10] Indoor Positioning Using Sensor Fusion in Android Devices
(http://hkr.diva-portal.org/smash/get/diva2:475619/FULLTEXT02)

[11] A Magnetic Field Based Lightweight Indoor Positioning System for Mobile Devices
(https://www.microsoft.com/en-us/research/wp-content/uploads/2015/10/huang.pdf)

Resources Required:

- PC (2 GB RAM minimum 8 GB recommended, 2 GB HDD space).
- Android Studio.
- Java Development Kit (JDK) 8.
- Inkscape.

- IndoorAtlas SDK and API, API Key.
- Android smart phone running Android version 4.3 or later, accelerometer, gyroscope and compass for calibration.

Marking Scheme:

	Marks
Introduction	10
Background, research, chosen solution	10
Data Gathering	10
Design	15
Implementation	30
Testing	5
Conclusion	10
Critical Self-Appraisal	10

Contents

Acknowledgments

I would like to express my sincerest gratitude to my project supervisor Dr Ying Liang for her continuous support, guidance and enthusiasm that provided me with the motivation to write this report. This report would not have been possible without her professional advice.

I would also like to express my gratitude to my project moderator Dr Mark Stansfield for his energetic lectures and determination that made sure everybody is on the right track. His humour and witty remarks made coming to class a real pleasure.

I would like to thank IndoorAtlas for making their awesome technology and APIs freely available to developers.

Last but not the least, my love and devotion goes to my two biggest fans; my daughters Lindsey and Leanne who helped a lot with the house chores when I am too busy with the project.

1. Introduction

Today, every part of the earth has already been mapped and it's no wonder that that the mobile marketplace is teeming with all kinds positioning and navigation apps. But it's a different story when it comes to indoor locations. GPS does not particularly work well inside buildings and Wi-Fi tends to be unreliable and inaccurate to be used for indoor positioning. While mainstream solutions rely on the installation of hardware such as beacons and RFID readers and writers, IndoorAtlas has found a solution by exploiting the unique magnetic fingerprint of buildings. IndoorAtlas claims that their unique technology can provide very accurate positioning within 1-2 meters inside buildings and that their solution requires no infrastructure beyond a smartphone. This honours project will utilize their technology to develop a mobile application that will be used for indoor positioning and navigation within the buildings of the university.

2. Background, Research and Chosen Solution

2.1 Background Context and Project Overview

This document is a final stage report for a software based IPS (Indoor Positioning/Mapping System) project to be implemented on an Android operating system/device using geomagnetic data gathered from the unique magnetic landscape of selected buildings/structures within the Paisley campus of the University of Scotland. The project will use the Android API and SDK provided by IndoorAtlas (www.indooratlas.com) to communicate with IndoorAtlas' cloud location service that estimates the device's indoor position and passes the information back to the application as GPS coordinates. In addition to the geomagnetic technology that IndoorAtlas uses as the foundation for its indoor positioning technology, IndoorAtlas can also leverage other device hardware sensors for further optimisation of its mapping and position data. Device sensors such as accelerometer, gyroscope, Wi-Fi and Bluetooth can also be utilized during mapping and positioning for better accuracy and performance.

2.2 Project Development Key Stages

The project development has three key stages and will be discussed in greater detail in the succeeding sections of this report.

1. **Mapping**
   The initial stage of the project starts with surveying the venue where mapping and positioning will be conducted. Floor plans for the selected building/s must be created and uploaded to IndoorAtlas web application. The floor plans must have the correct scale and orientation and must be aligned with corresponding geographic coordinates.

   After a venue is created and the floor plans uploaded, **the next step is to upload positioning data to the server** by collecting magnetic field readings of all the paths that will be covered by the positioning service. This is accomplished by using the IndoorAtlas Map Creator mobile application and walking along the planned routes and adding checkpoints, waypoints and paths.

   The IndoorAtlas mapping guide can be accessed from this website: http://docs.indooratlas.com/app/

   *Note:
   The map covers a total of seven floor plans from three different buildings within the university's Paisley campus.
2. **Positioning**

The first step in creating the mobile application is to integrate the IndoorAtlas service into the project. The SDK contains several APIs that provides regular indoor location updates to the application. **The application must then implement event listeners to listen to the location updates, and from there on it's up to the application's business logic to handle and implement the GPS coordinates received from the IndoorAtlas service.** The current version of the IndoorAtlas SDK is 2.2.2 and can be accessed from the following packages: **com.indooratlas.android.sdk** and **com.indooratlas.android.sdk.resources**. API documentation can be accessed from this webpage: **http://docs.indooratlas.com/android/2.2.2/**

[1] The *IALocationManager* class provides access to the IndoorAtlas location services. Documentation at: http://docs.indooratlas.com/android/2.2.2/com/indooratlas/android/sdk/IALocationManager.html

[2] The *IALocationListener* class is used to receive location updates when location has changed. Documentation at: http://docs.indooratlas.com/android/2.2.2/com/indooratlas/android/sdk/IALocationListener.html

3. **Navigation**

   Design and development is yet to be started on the navigation or way-finding part of the application. This service is not provided by IndoorAtlas. However, using IndoorAtlas as a foundational technology; the research for this project will cover the possibility of building the navigation and routing aspect of the application on top of IndoorAtlas location services and the mapping services provided by the Google Maps API. Several routing algorithms that implement the Shortest Path algorithm and its variants will be investigated. This possible solution is not set in stone and may change as the development of the project progresses.

   *Note:
   Testing and debugging of the application is done within the building/s where the positioning/location service will be provided. The IndoorAtlas API for location listeners will not work outside of the mapped areas of the building/s.

Indoor Positioning Systems (IPS)

There are currently no standard for implementing an IPS system as the technology is still young and continue to evolve. The companies that provide IPS solutions typically employ a combination of different technologies to achieve a decent level of precision with regards to locating a position indoors. The technologies that are currently being implemented in the market for indoor positioning include Wi-Fi, Low Energy Bluetooth (BLE) Beacons, GPS, RFID and geomagnetic field. Geomagnetic positioning is an IPS technology that does not require any hardware. IndoorAtlas, the company which implements a cloud based service for geomagnetic positioning holds several patents on the technology. The technology uses the magnetic sensor data from a smartphone to locate a position of a device indoors.

IndoorAtlas defines Indoor positioning systems as:

"Indoor positioning systems (IPS) locate people of objects inside a building using radio waves, magnetic fields, acoustic signals or other sensory information collected by a smartphone device or tablet. These systems are used to detect and track a position. Just think of it as GPS, but for indoors."

IndoorAtlas, (No Date), How It Works - What Is Indoor Positioning Systems [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed December 30, 2016]

2.3 Indoor Location Technologies

BLE Beacons

IPS provided by other companies relies on the use of different hardware and multiple systems such as Wi-Fi, beacons (Low Energy Bluetooth/BLE), Radio (RFID/NFC) and GPS to provide positioning services. Almost all require hardware to locate and determine the position of a device or object indoors. The use of BLE beacons is probably the most widely used in the market today. The problem with beacons is the cost involved with acquiring the hardware and the maintenance involved once the solution has been implemented. A single beacon does not provide the distance of a device or user from a sensor but can only detect the proximity. Multiple overlapping beacons must be in place if a device's or a person's indoor position is to be accurately determined. A BLE beacon depending on the make and model typically has a RSSI (Received Signal Strength Indicator) broadcast range of 3 to 50 meters and can locate a position of a device indoors within 5 to 8 meters (lighthouse.io, n.d.).

Wi-Fi

Using Wi-Fi networks alone to pinpoint a location of a device indoors is inaccurate and unreliable as signal strength tends to vary a lot and can drastically affect readings. The fixed nature of Wi-Fi systems makes them less flexible in comparison to other IPS technology like BLE beacons. There is also the issue of installation cost as multiple Wi-Fi access points is needed to be installed in order to provide at least 10 meter positioning accuracy consistently (IndoorAtlas, n.d.). The number of access points needed depends on the size of the area where positioning service is to be implemented. Wi-Fi networks that use trilateration together with BLE beacons are used to achieve a level of precision of 1 to 2 meters and less than 1 second refresh rates required for indoor positioning and navigation. However, this approach is expensive in terms of hardware acquisition, setup and maintenance.

GPS

GPS is highly accessible and accurate when used outdoors. According to GPS.gov, the FAA's (Federal Aviation Administration) high quality GPS receivers can attain results better than 2.168 meter horizontal accuracy, 95% of the time (GPS.gov, n.d.). However, as a GPS device relies on a series of satellites to determine its location; accurate positioning results can only be achieved if the device (smartphone with GPS receiver) has a clear line of sight to the sky. GPS is affected by atmospheric effects, the quality of the receiver and other physical barriers and interference like structures and buildings. This is the reason why GPS does not work well inside buildings and structures.

"A GPS works better when the device has a clear line of sight to the sky. The more GPS satellites that your personal device can access, the more accurate it is. When inside, there is often no direct line from the satellite signals to your device. The signal weakens or distorts as it travels through the buildings to your GPS, and the result is inaccurate operation."

Hans Fredrick, (No Date), Why Doesn't GPS Work Inside a Building? [Online] Tech in our everyday life, Available: http://techin.oureverydaylife.com/doesnt-gps-work-inside-building-18659.html [Accessed December 31, 2016]

GPS is used together with other indoor positioning technologies to provide a better accuracy for locating the position of a device inside a building or structure.

RFID

Radio Frequency Identification (RFID) uses radio waves to track objects (devices) and people wirelessly that can be used for indoor positioning. However, the hardware and infrastructure needed to implement such solution can be quite costly. This system is composed of tags (Active or Passive) that communicate via radio waves with a reader/writer device which in turn communicates with a controller which is usually a computer with a third party application software installed. Although the accuracy provided by a RFID solution can be very high, it is also one the most difficult solutions to implement. Implementing such solution requires a great deal of planning and preparation upfront with regards to hardware acquisition and setting up the infrastructure. This solution does not leverage the power and accessibility of smartphone and other mobile devices, as the RFID requires hardware and firmware that the current mobile devices in the market do not inherently possess (lighthouse.io, n.d.).

IndoorAtlas Geomagnetic Technology

Geomagnetic positioning is an IPS technology that uses a building's magnetic fingerprint to accurately locate a device's position inside a building or structure. The technology uses the hardware sensors typically inherent on smartphones and tablet devices that are currently on the market. This technology was invented by Professor Janne Haverinen (https://www.google.com/patents/US20130177208, July 11, 2013) who later founded the company IndoorAtlas that holds the patent (http://patents.justia.com/inventor/janne-haverinen, August 15, 2016) for measuring the earth's magnetic field indoors. IndoorAtlas describes how their geomagnetic technology for indoor positioning works:

"Modern buildings all have a unique magnetic landscape produced by the Earth's magnetic field that interacts with steel and other materials found in structures of buildings. By utilizing the built-in magnetic sensor (the compass) as well as other sensing technologies within a smartphone, our software is able to use the magnetic field inside the building as a map to accurately pinpoint and track a person's location indoors, producing a "blue dot" on a map. We know the start point, the path a person's taken, and their end point. This is all done in real time – just like GPS outdoors.

Geomagnetic technology is the foundation of our indoor positioning but we are able to leverage other sources of information such as Wi-Fi, for further optimization. Our cloud platform is the hub for application developers – we've made it easy to add support for indoor positioning in a building, manage the data and use our SDK to build location-based services within an app."

No Author, (No Date), How Does Our Geomagnetic Technology Work? – How It Works [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed January 02, 2017]

IndoorAtlas geomagnetic technology provides 1 to 2 meters of positioning accuracy (IndoorAtlas, n.d.) without the need for any costly hardware acquisition, setup and maintenance. Below are two different tables comparing IndoorAtlas' geomagnetic IPS to other indoor positioning technologies:

Technology	Accuracy	Infrastructure	Set-Up /Costs	Maintenance	Power Source	Developer Environment
Magnetic Positioning	< 6 feet	Software/ Cloud	API / minimal	Crowdsource	None	iOS/Android
WiFi	30 – 300 feet	Hardware/Soft-ware	Triangulation / $$$	Remap - Fingerprinting	Electricity/ Battery	Android only
BLE	6 – 100 feet (no blue dot)	Hardware/ Software	Configurable/ $$	Device Management	Battery (Degrades)	iOS/Android
PDR (pedestrian dead reckoning)	N/A - Complementary	Software/Cloud	API / minimal	Crowdsource	None	iOS/Android
Other (cameras, LED, sound)	Variable	Hardware/ Software	Mesh coverage / $$$	Device Man-agement	Varies	iOS/Android

Figure 1. Opus Research IPS comparison chart.

Greg Stirling, (June 2014), Magnetic Positioning, The Arrival of Indoor GPS [Online] Opus Research Report, Available: https://www.indooratlas.com/wp-content/uploads/2016/03/magnetic_positioning_opus_jun2014.pdf [January 02, 2017]

	WiFi	Beacon (BLE/iBeacon)	IndoorAtlas
Ability to scale	Low	Very low	Very high
Accuracy	Low	Very low	Very high
Hardware cost / venue (typ.)	> $1,000	> $1,000	$0
Supported platforms	Android	iOS / Android	iOS / Android
Real-time location	✓	✗	✓
Software-only solution	✗	✗	✓
Developer PaaS & SDK	✗	✗	✓

Figure 2. Comparison provided by IndoorAtlas (indooratlas.com).

IndoorAtlas, (No Date), How Do We Differ From Others? – How It Works [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed January 02, 2017]

IndoorAtlas also provides a comprehensive PaaS (Platform as a Service) for developers that includes a web based mapping tool, SDKs and APIs for both Android and IOS. Although the technology provided by IndoorAtlas does not need any additional hardware to purchase and install, their cloud service does not come completely free.

Pricing and Licensing

IndoorAtlas relies on their Platform as a Service (PaaS) business model to commercialize on their geomagnetic IPS technology.

The non-commercial license agreement can be found at: http://www.indooratlas.com/api-license/

The mobile license agreement can be found at: http://www.indooratlas.com/mobile-license/

The developer service terms and conditions can be found at: http://www.indooratlas.com/terms/

Commercial, revenue generating or enterprise license agreement is separately authorised by IndoorAtlas.

The project uses the IndoorAtlas free plan. The free plan allows unlimited use of their mapping tools, SDKs and APIs; provided that the app's monthly users does not exceed 100. Below is a more detailed list of their standard plans when the app exceeds more than 100 users per month:

Standard Plans

Unique users per month	Cost per month
101 – 1,000	$ 100
1,001 – 5,000	$ 300
5,001 – 10,000	$ 500
10,001 – 50,000	$ 1,700
50,001 – 100,000	$ 3,000
100,001 – 500,000	$ 11,000
500,001 – 1,000,000	$ 20,000
Over 1,000,000	Contact Sales

Figure 3. IndoorAtlas standard plans.

IndoorAtlas, (No Date), Pricing [Online] IndoorAtlas, Available: https://www.indooratlas.com/pricing/#all-plans [Accessed January 02, 2017]

2.4 Chosen Solution

IndoorAtlas has been awarded the “Most Innovative Start-Up” award at the Telecom Council Spiffy (https://www.telecomcouncil.com/spiffys/) Winner for 2015. Their geo-magnetic IPS technology (cloud based location service) truly delivers a scalable solution for indoor positioning with 1-2 meter accuracy without the need to install extra hardware. ABI Research describes the technology as the “Third Pillar in Indoor Location”.

“Its major advantage over infrastructure is its scalability. Over time, if IndoorAtlas can become commonplace in apps and on devices, supporting crowd sourced map building, it has huge potential to become a true alternative to Google and Apple. Suddenly, any retailer can partner with them and tap into their location data stream for their venues. Similarly, any company looking to create services around indoor mapped venues will look to license from IndoorAtlas”

Abiresearch.com. (2017). Will IndoorAtlas become the third pillar in Indoor Location in 2016? [online] Available at: https://www.abiresearch.com/blogs/will-indooratlas-become-third-pillar-indoor-location-2016/ [Accessed 20 Mar. 2017].

Companies from around the world like Baidu China, Yahoo Japan, ESRI UK and Mumbai Airport chose to use the IndoorAtlas platform for their indoor positioning solution because of its ability to scale regardless of the number of users.

The project for the UWS Indoor Positioning application will take advantage of the accessibility and scalability of the technology. Indoor positioning will be implemented by utilizing the IndoorAtlas cloud and software based location service without the need to purchase and install extra hardware like beacons and routers.

3. Data Gathering

3.1 Mapping

The first step in creating a map/location based mobile app using the IndoorAtlas service is to create a venue. The venue is the general location where mapping and positioning will be carried out. After the venue is created, individual floor plans for different buildings and levels/floors must be uploaded before any mapping activity can continue. An account is necessary to use IndoorAtlas location services. New accounts can be registered at app.indooratlas.com. Below is an image of the venue created with IndoorAtlas locations.

Figure 4. UWS Map Venue. Source: https://app.indooratlas.com/locations

Venue:

University of the West of Scotland, Paisley Campus

Details:

Location ID - 1b44944d-deee-4ce4-a998-e90a0b8becaa

Created at - 2016-11-03 13:42 UTC

Address - Townhead Terrace, Renfrewshire, PA1 2BE, Scotland, United Kingdom

GPS Coordinates - 55.842717229947, -4.43033337593079

*Note – The GPS coordinates represents the location where the position marker was placed in the general vicinity of the venue. A location id is necessary as more than one venue can be registered.

3.2 Floor Plans

The floor plans (including the map icons) were created using an open source vector graphics drawing software called “Inkscape” (https://inkscape.org/en/). The images were uploaded to the IndoorAtlas server in the PNG (Portable Network Graphics) format. Seven floor plans were created and uploaded that covers seven different floors/levels for three buildings within the campus. The floor plans were aligned and scaled to the correct measurement as accurately as possible using the IndoorAtlas web application.

[1] Floor plan ID: 00329a79-da39-4a5c-8f4b-fefb6219a4f1:

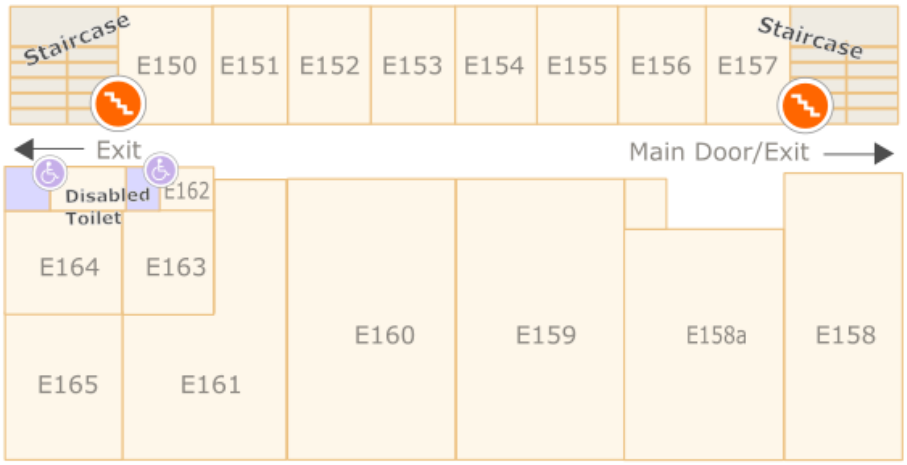

Figure 5. E Block South Building, Level 1

[2] Floor plan ID: 399708a3-1e6b-4b74-8251-8a5d6f2062c7:

Figure 6. E Block South Building, Level 2

[3] Floor plan ID: 64761fe2-dd26-46b7-aa3e-c95dd7c5b9ef:

Figure 7. E Block South Building, Level 3

[4] Floor plan ID: 64761fe2-dd26-46b7-aa3e-c95dd7c5b9ef:

Figure 8. E Block Henry Building East, Level 1

[5] Floor plan ID: 7544a9a2-2320-4fff-ba81-9d0a3a3cfc93:

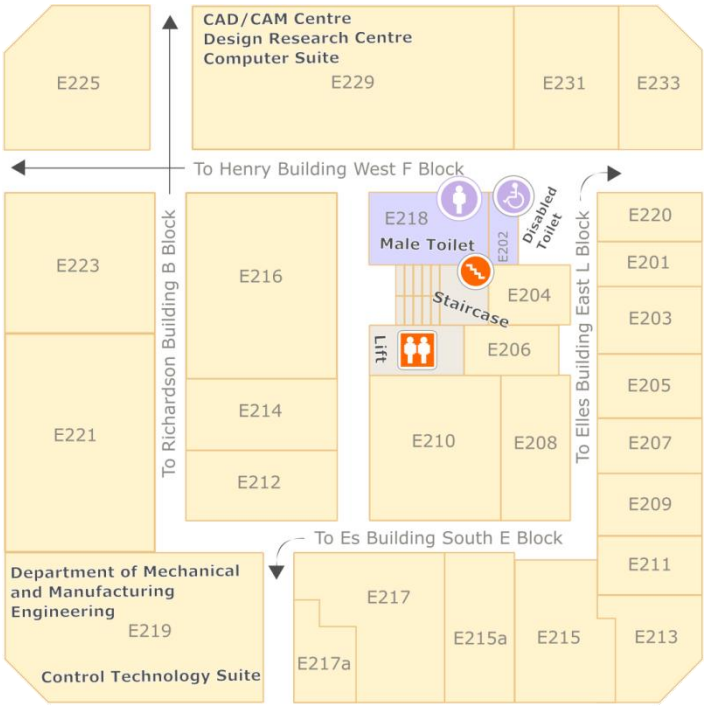

Figure 9. E Block Henry Building East, Level 2

[6] Floor plan ID: 2b120ce0-6c65-416a-b518-cf4b46fa76b1:

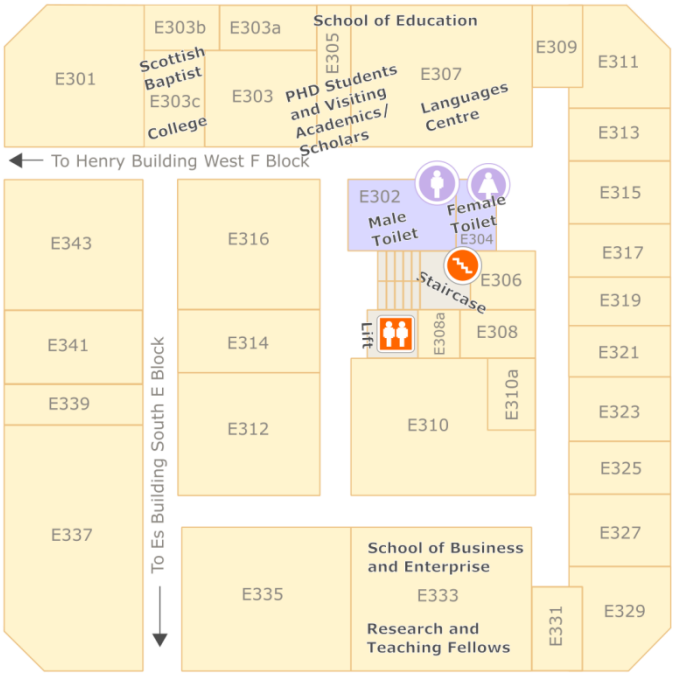

Figure 10. E Block Henry Building East, Level 3

[7] Floor plan ID: 4f869e74-5f7f-4d28-99d3-0427481a1f02:

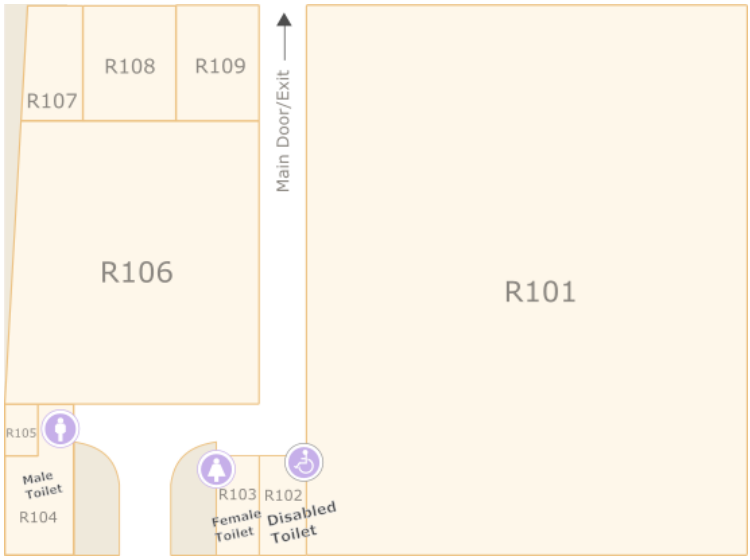

Figure 11. R Block Oswald Building

Inksape, nd. Inkscape. [computer program]. Inkscape Project. [Accessed January 5, 2017].

3.3 Mapping the Locations

The goal for mapping the locations using the Map Creator tool is to cover all the walkable paths and routes within the selected indoor location. Using a smartphone, magnetic readings will be recorded and uploaded to the IndoorAtlas server by walking along and setting checkpoints, waypoints and paths for every floor/level within the building or structure. Aside from the magnetometer (digital compass), additional sensor data like Wi-Fi; accelerometer and gyroscope are also utilized to achieve a more accurate and optimal reading.

It is extremely important that the smartphone that will be used to record the magnetic field readings must first be calibrated. Phone calibration is explained in the IndoorAtlas website: http://docs.indooratlas.com/app/.

The steps that were followed and undertaken for the mapping process are briefly outlined below:

- Step 1 – Create Location/Venue and Add Floor Plan
- 1-0 – Sign-in
- 1.1 – Create location
- 1.2 – Add floor plan
- Step 2 – Map Location/Venue
- Preparation for Mapping
- 2.0 - Download MapCreator 2
- 2.1 – Get started
- 2.2 – Calibrate phone
- 2.3 – Add waypoints
- 2.4 – Get the walk right before mapping
- 2.5 – Mapping
- 2.6 – Stop and save
- Step 3 – Manage Map Data

These steps are fully explained in greater detail in the IndoorAtlas website: http://docs.indooratlas.com/app/.

The mapping results were achieved using the magnetometer, Wi-Fi, accelerometer and gyroscope sensors of an LG Nexus 5 running on stock Android operating system smartphone device.

3.4 Mapping Results/Analytics

This section shows the results of the mapping process after all the magnetic readings and other sensor data were collected from the mapped locations. The generated map that includes the map analytics (overlay and details), paths, waypoints, convergence distance, positioning error and map coverage will be shown for every floor plan of the three buildings.

Key terms related to the map analytics and overlays:

[1] **Convergence distance** – The distance that the device must move before positioning is accurate. Note that the convergence distance may differ between map areas (positioning starting points), due to differences in the magnetic and radio landscapes.

[2] **Positioning error** – Estimates the distance from the actual position after convergence. Note that the positioning error may differ between map areas, due to differences in the magnetic and radio landscapes.

[3] **Map coverage** –This depicts the navigable area, i.e. the area that has been mapped and where the devices can be positioned. The area depends on the navigable area mask and the location of the map paths.

[4] **Magnetic mapping coverage** – The amount of magnetic field data that was collected during the mapping process.

[5] **Wi-Fi environment quality** – Depicts the quality of Wi-Fi signal within the mapped areas of a floor plan.

Map analytics presentation:

The map analytics for the convergence distance and positioning error will be shown in different colors that represent the distance in meters. Map coverage will be shown using two colors that represent the mapped area and the validated area. Waypoints that were manually placed (no more than 10 to 20 meters apart with 0.5 to 1 meter distance to walls as per recommendation) are represented on the map as circles with a dark blue outline. Paths that were undertaken and recorded are represented as purple lines.

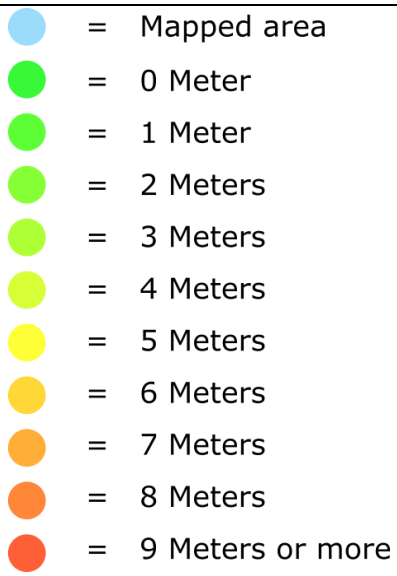

Figure 12. Convergence distance and positioning error

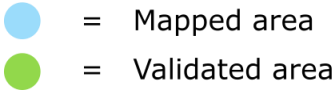

Figure 13. Map coverage.

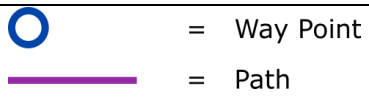

Figure 13. Waypoint and path.

Figure 14. Magnetic mapping coverage and Wi-Fi environment quality.

Inksape, nd. Inkscape. [computer program]. Inkscape Project. [Accessed January 5, 2017].

3.4 Map analytics for the E Block South Building

Figure 15. Convergence distance overlay for floor level 1

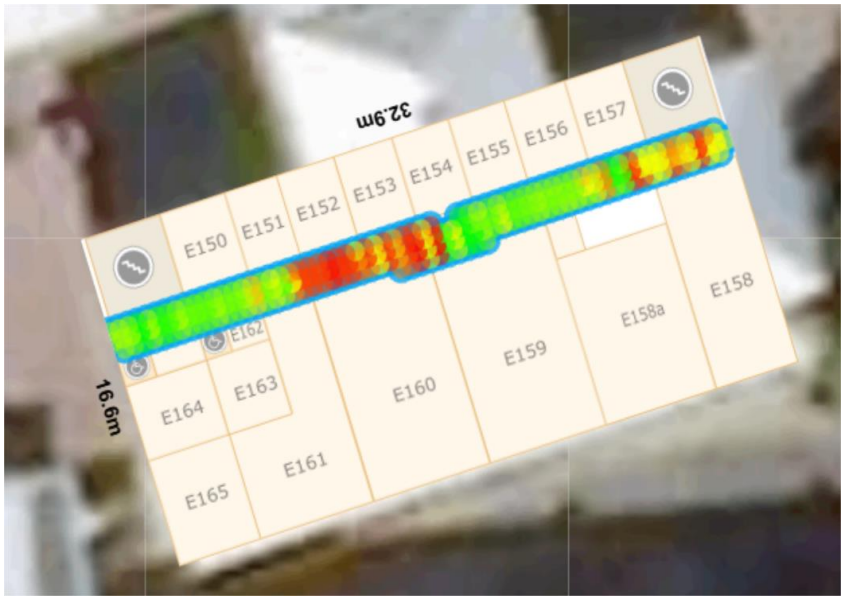

Figure 16. Positioning error overlay for floor level 1

Figure 17. Map coverage, waypoints and path/s overlay for floor level 1

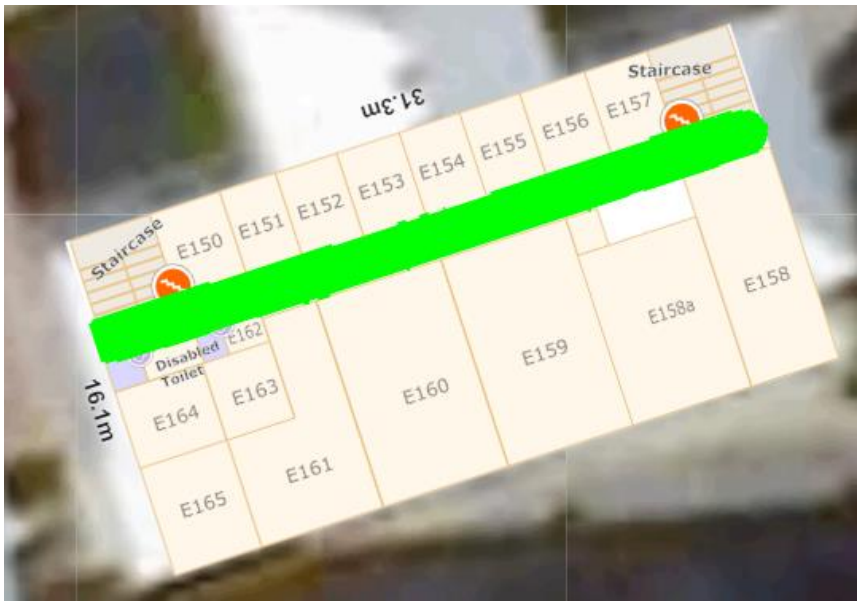

Figure 18. Magnetic mapping coverage overlay for floor level 1.

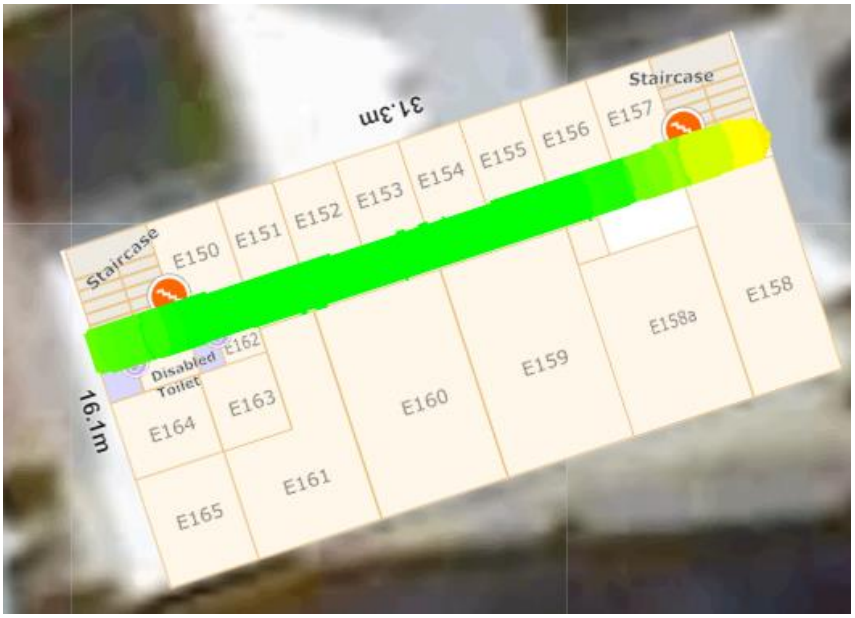

Figure 19. Wi-Fi environment quality overlay for floor level 1.

Figure 20. Convergence distance overlay for floor level 2

Figure 21. Positioning error overlay for floor level 2

Figure 22. Map coverage, waypoints and path/s overlay for floor level 2

Figure 23. Magnetic mapping coverage overlay for floor level 2.

Figure 24. Wi-Fi environment quality overlay for floor level 2.

Figure 25. Convergence distance overlay for floor level 3

Figure 26. Positioning error overlay for floor level 3

Figure 27. Map coverage, waypoints and path/s overlay for floor level 3

3.5 Map analytics for the E Block Henry Building East

Figure 28. Convergence distance overlay for floor level 1

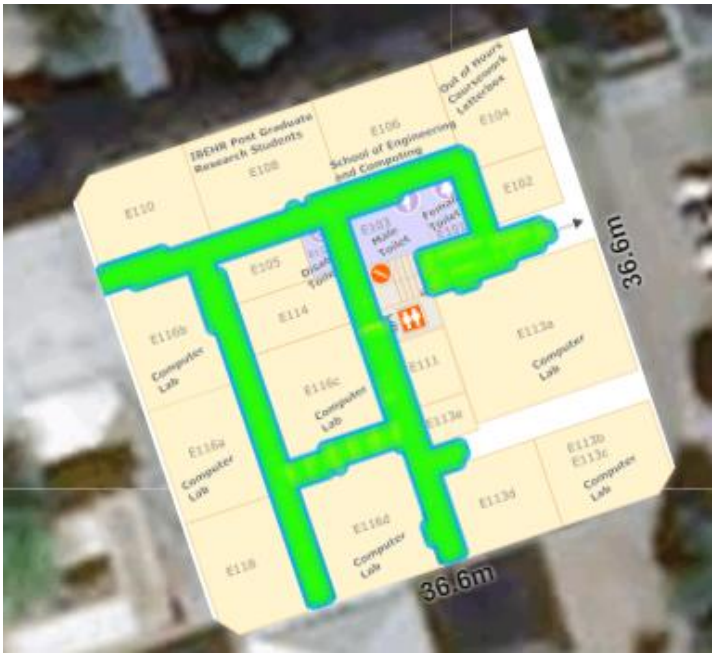

Figure 29. Positioning error overlay for floor level 1

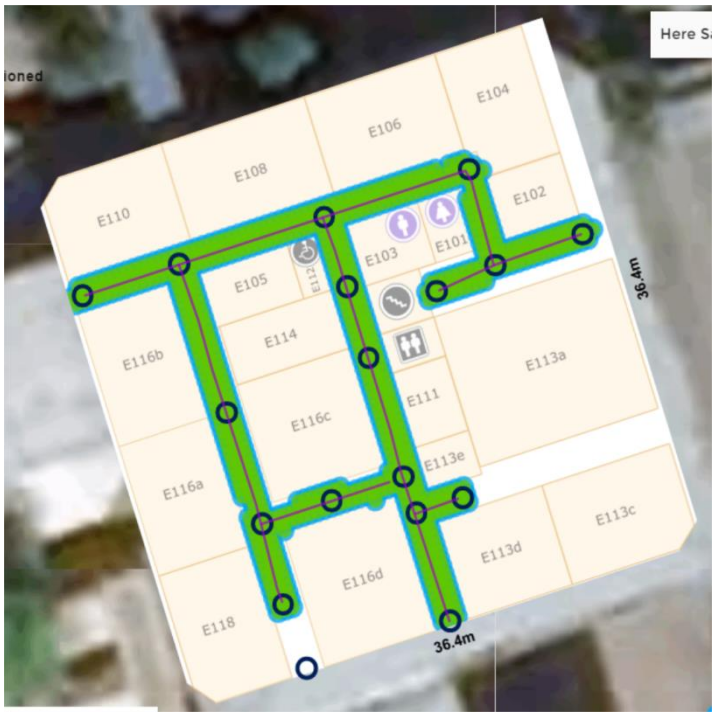

Figure 30. Map coverage, waypoints and path/s overlay for floor level 1

Figure 31. Magnetic mapping coverage overlay for floor level 1.

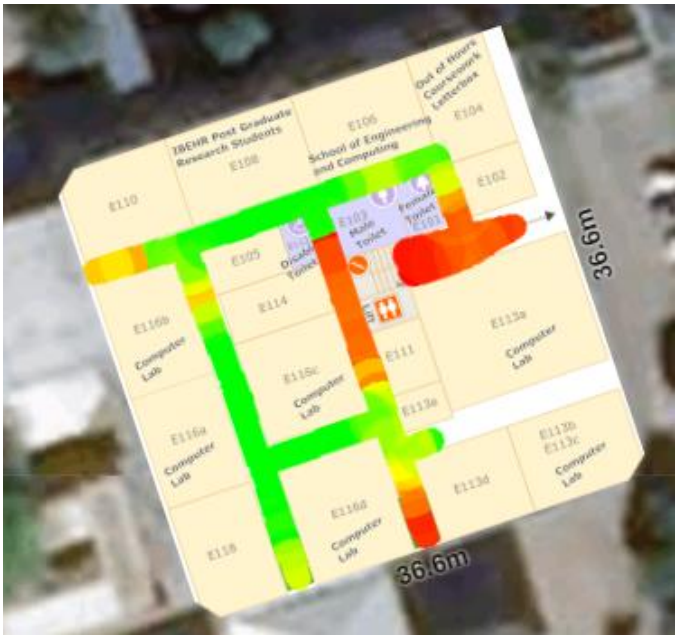

Figure 32. Wi-Fi environment quality overlay for floor level 1.

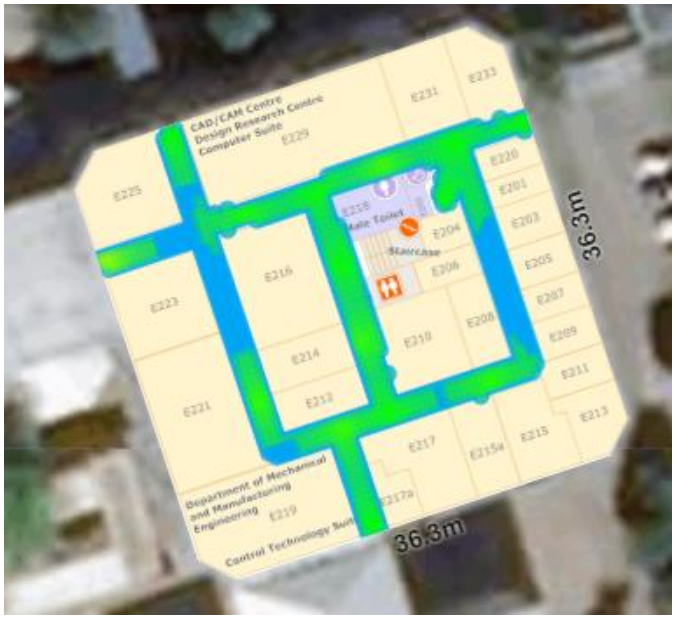

Figure 33. Convergence distance overlay for floor level 2

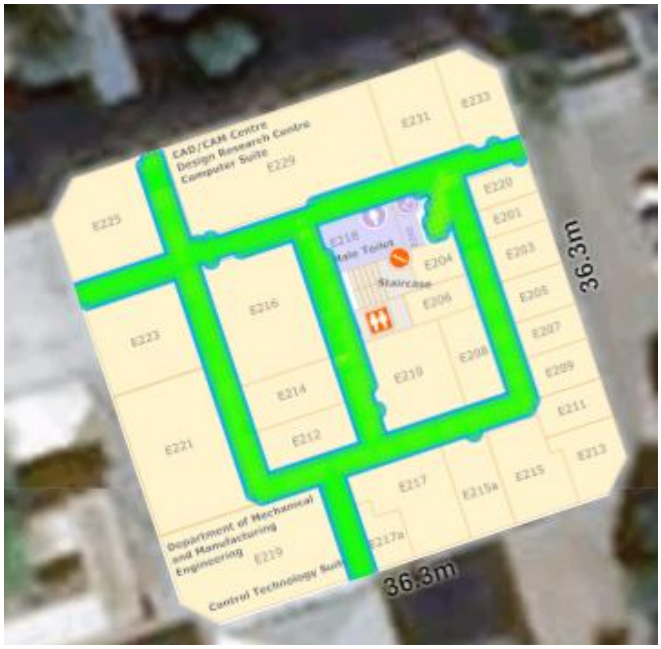

Figure 34. Positioning error overlay for floor level 2

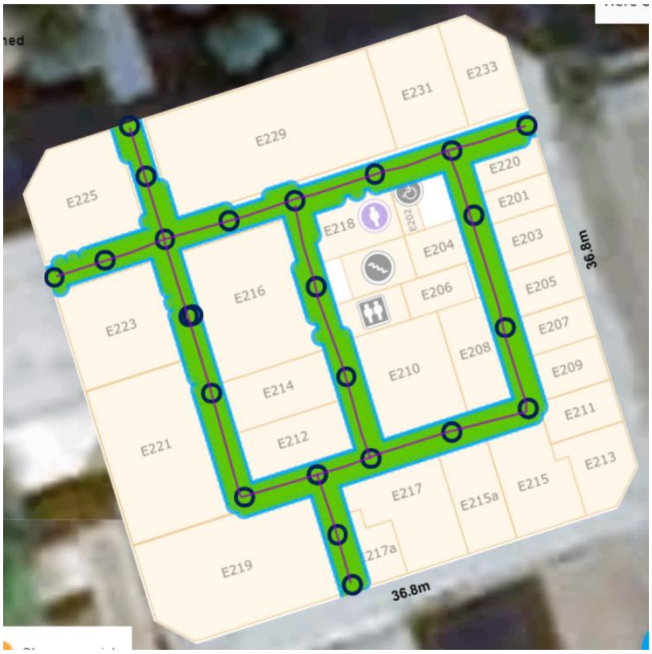

Figure 35. Map coverage, waypoints and path/s overlay for floor level 2.

Figure 36. Magnetic mapping coverage overlay for floor level 2.

Figure 37. Wi-Fi environment quality overlay for floor level 2.

Figure 38. Convergence distance overlay for floor level 3.

Figure 39. Positioning error overlay for floor level 3.

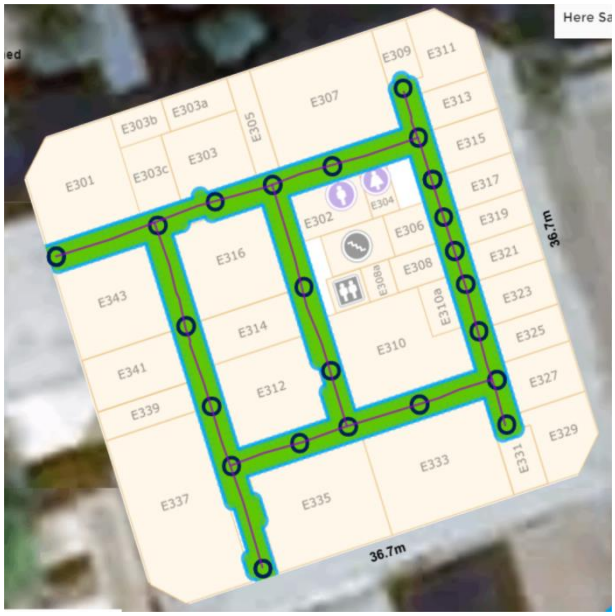

Figure 40. Map coverage, waypoints and path/s overlay for floor level 3.

Figure 41. Magnetic mapping coverage overlay for floor level 3.

Figure 42. Wi-Fi environment quality overlay for floor level 3.

3.6 Map analytics for the R Block Oswald Building

Figure 43. Convergence distance overlay.

Figure 44. Convergence distance overlay.

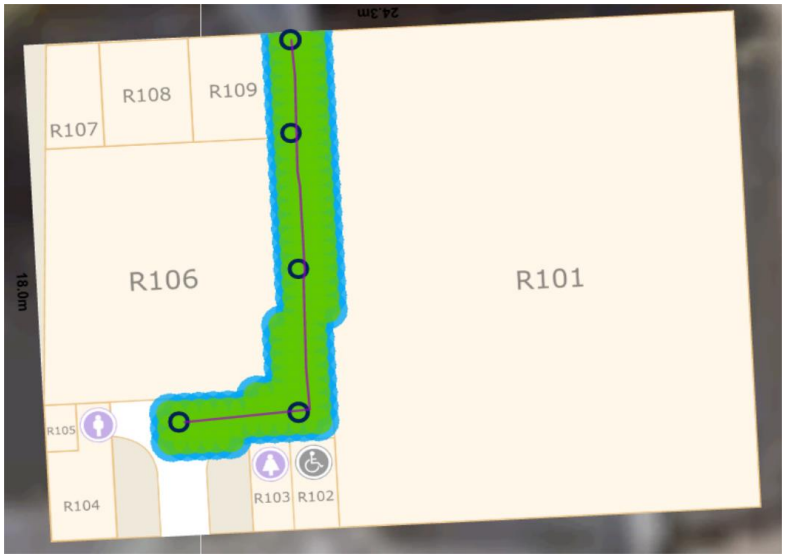

Figure 45. Map coverage, waypoints and path/s overlay.

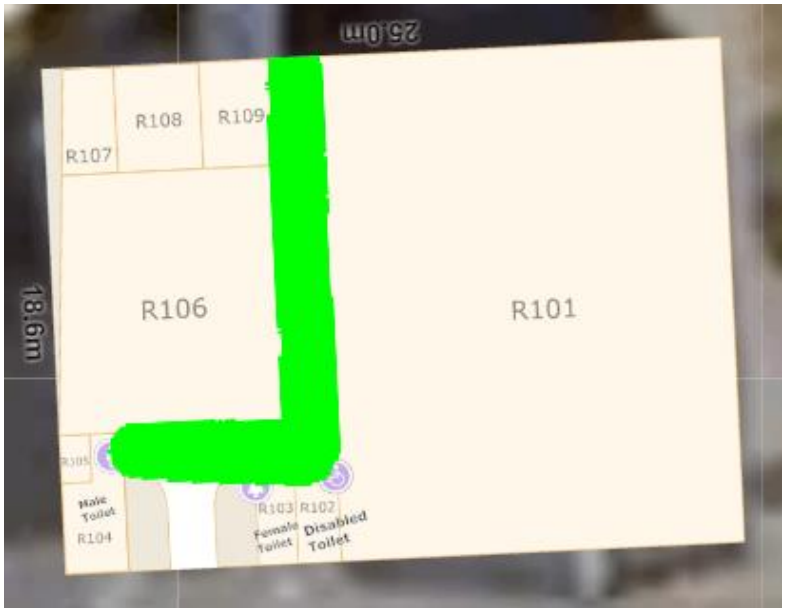

Figure 46. Magnetic mapping coverage overlay.

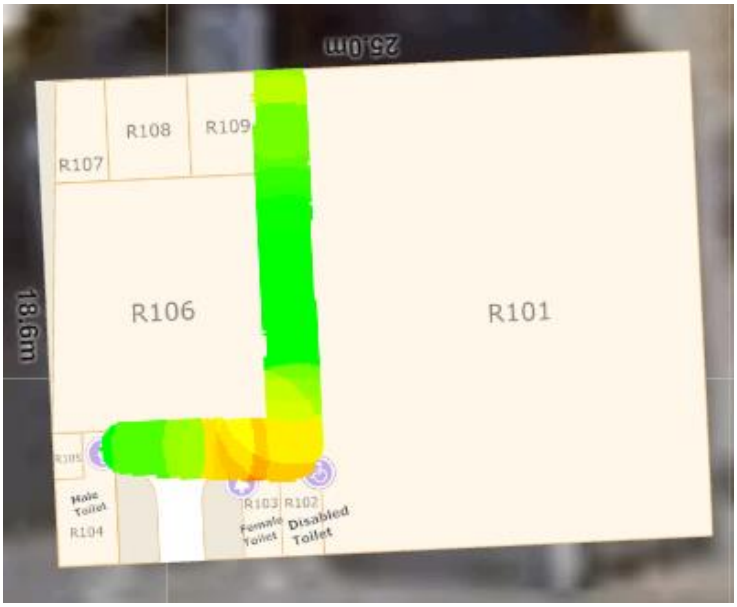

Figure 47. Wi-Fi environment quality overlay.

3.7 Mapping accuracy achieved:

[1] E Block South Building Level 1: 1 to 2 meter accuracy on certain points in the path. Positioning error of up to 9 meters or more on certain points in the path. Good magnetic mapping coverage throughout. Good Wi-Fi environment.

[2] E Block South Building Level 2: 1 to 2 meter accuracy throughout. Good magnetic mapping coverage throughout. Average/bad Wi-Fi environment quality.

[3] E Block South Building Level 3: 1 to 2 meter accuracy throughout. Good magnetic mapping coverage throughout. Good/average Wi-Fi environment quality

[4] E Block Henry Building East Level 1: 1 to 2 meter accuracy throughout. No positioning error. Good magnetic mapping coverage throughout. Bad Wi-Fi environment quality.

[5] E Block Henry Building East Level 2: 1 to 2 meter accuracy throughout. No positioning error. Good magnetic mapping coverage throughout. Good/Average Wi-Fi environment quality.

[6] E Block Henry Building East Level 3: 1 to 2 meter accuracy on most paths. Positioning error of up to 5 meters on certain points in the path. Good magnetic mapping coverage throughout. Bad Wi-Fi environment quality.

[7] R Block Oswald Building: 1 to 2 meter accuracy throughout. No positioning error. Good magnetic mapping coverage throughout. Good/Average Wi-Fi environment quality.

4. Design

4.1 Project Tool Chain

Below is a list of technologies (SDK/API) and tools (hardware/software) that is required to design and develop the UWS Indoor Positioning and Navigation using the IndoorAtlas technology on an Android device:

1. **IndoorAtlas Locations** – a web application used for creating a venue, submitting and aligning floor plans with geocoordinates where the magnetic field readings will be recorded and where mapping and positioning will be implemented. The web application can be accessed from https://app.indooratlas.com/locations.

2. **IndoorAtlas Map Creator** – a mobile app used to generate and gather the magnetic field readings and other sensor data for a map. Checkpoints, waypoints and paths can be set for a particular map and floor plans using this app. The Android app can be downloaded from the Google Play Store: https://play.google.com/store/apps/details?id=com.indooratlas.android.apps.jaywalker&hl=en

3. **Inkscape** – an open source graphics editor used to draw the maps and floor plans for the venue where the indoor positioning will be implemented. Inkscape can be downloaded from this site: https://inkscape.org/en/

4. **Java Development Kit (JDK) version 8** – a development platform for the Java programming language. The JDK Standard Edition contains the tools necessary to develop, compile, test and package programs written in the Java programming language. The product binaries required for the project are from the Oracle JDK 8 (http://www.oracle.com/technetwork/java/javase/downloads/index.html). The alternative open source OpenJDK (http://openjdk.java.net/) is not officially supported by the current version of the Android platform.

5. **Android Studio** – the official integrated development environment (IDE) for the Android platform. It contains end to end tools necessary to design, develop, debug, test, build, collaborate and deploy applications that run on every type of Android device. Android Studio's current version is 2.2.3 and can be download at: https://developer.android.com/studio/index.html

6. **IndoorAtlas Android SDK** – contains the API and Java classes for creating location aware Android apps. The API provides the event listeners that must be implemented by the app to listen to the location updates from the

IndoorAtlas location service. The classes are contained in the following packages:

- com.indooratlas.android.sdk
- com.indooratlas.android.sdk.resources

The SDK documentation can be found at:
http://docs.indooratlas.com/android/2.2.2/

7. **Development Computer** – a computer that will be used to interact with IndoorAtlas web application for venue creation and floor plan submission and alignment. The computer will also be used to run the software/packages required to create the maps/floor plans (Inkscape) and the IDE (Android Studio) that will be used to develop the Android app. The computer must meet the recommended system requirements to run the latest version of the Android Studio.

   Android Studio system requirements for the Windows operating system:
   "Windows
   - Microsoft® Windows® 7/8/10 (32- or 64-bit)
   - 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
   - 2 GB of available disk space minimum,
   - 4 GB recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
   - 1280 x 800 minimum screen resolution
   - For accelerated emulator: 64-bit operating system and Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality"

   No Author, (No Date), System Requirements [Online] Android Studio, Available: https://developer.android.com/studio/index.html [Accessed January 03, 2017]

   *Note – Android Studio also supports the Mac OS X and the Linux operating systems. Please visit the Android Studio official website for information regarding other operating system requirements (https://developer.android.com/studio/index.html). The Inkscape website does not specify any recommended system requirements for their software.

8. **Android Smartphone device** –a mobile phone used for the mapping of selected indoor locations/venue using the IndoorAtlas Map Creator Android app. The device is also used for the development, testing, debugging and deployment of the Indoor Mapping and Navigation (location aware) Android App. IndoorAtlas recommends the following devices for production quality maps:

“For production quality maps, we recommend using Nexus 5, Nexus 5X, Nexus 6, Nexus 6P, Honor 8, LG G4, Xiaomi Mi4, OnePlus 2 and OnePlus 3”

No Author, (No Date), Preparation for Mapping [Online] IndoorAtlas, Available: http://docs.indooratlas.com/app/ [Accessed January 03, 2017]

*Note – The recommended Android devices are used for mapping the selected indoor locations using the IndoorAtlas Map Creator Android app. The location aware Android application that will be developed for this project will run on any Android device (where Android 5.0 Lollipop is the minimum SDK) where the required hardware sensors are supported.

IndoorAtlas recommends that the smartphone must meet the following minimum requirements to implement and execute a location aware app using the IndoorAtlas SDK:

“Minimum Requirements:

- *SDK minimum API level 10 (Gingerbread)*
- *Physical Android device (emulator is not supported) with Wi-Fi connectivity.*
- *Gyroscope and Magnetometer are preferred.”*

IndoorAtlas, (No Date), Android SDK 2.2.4, Minimum Requirements [Online] IndoorAtlas, Available: http://docs.indooratlas.com/android/getting-started.html [Accessed January 03, 2017]

*Note – The app is currently being debugged and tested using USB-Debugging, as emulators are not fully supported (hardware sensors only works accurately on an actual physical device). Although the minimum Android API level is 10, the app for the project is set to have a minimum requirement of API level 21 (SDK Android 5.0 Lollipop) and a target API level of 23 (SDK Android 6.0 Marshmallow) to be compatible with the newer responsive UI layout and material design for Android.

9. **Android Realm Database** – an object type database architecture which was developed as an alternative to SQLite. Realm exposes the data as objects that can be directly queried and manipulated in code, thus eliminating the need to use any object relational mapping tool. It is 10x faster than SQLite, easier to use, supports full ACID transactions and thread safety.

   The project uses the Realm database to store and query database objects that hold information on different indoor locations. The database is modelled to store attributes such as type (i.e., room, computer lab, lift, toilet, stair, etc.), GPS coordinates, building, level, floor id and block. The database is loaded from a JSON file into the Realm database at runtime.

Below is an illustration of the **tool chain interaction**.

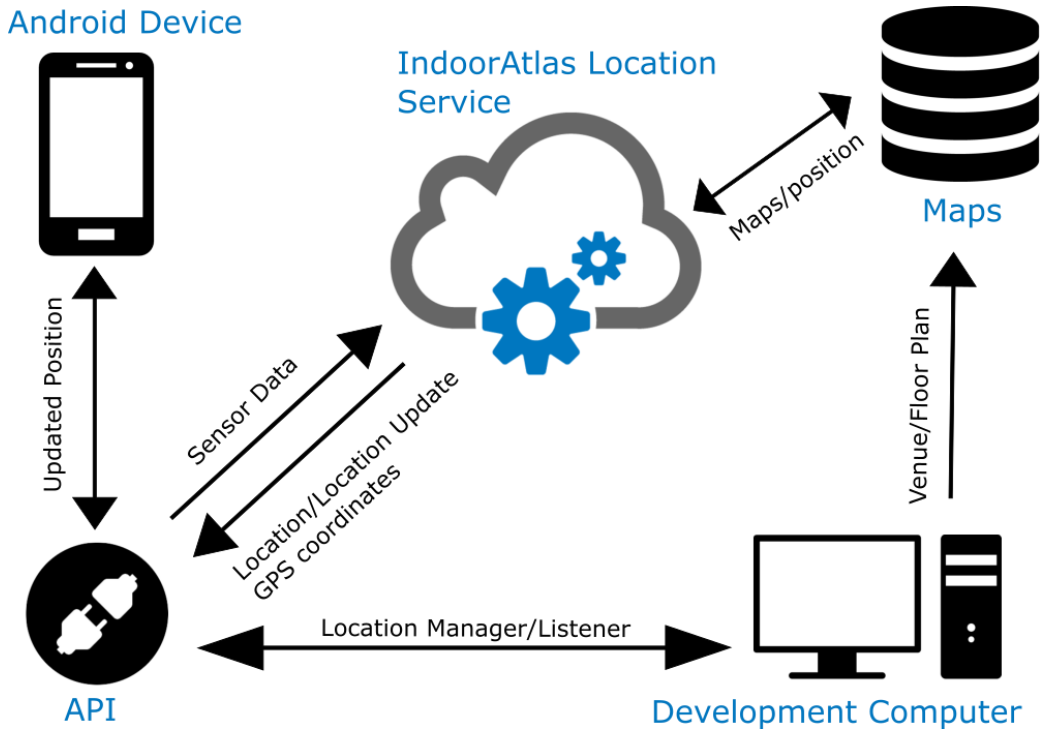

Figure 48. Project tool chain.

Inksape, nd. Inkscape. [computer program]. Inkscape Project. [Accessed January 3, 2017].

4.2 System Flowchart

The figure below shows a top level flow chart for the **Main activity**.

The MainActivity activity is the entry point of the application. It checks if the application has been granted the necessary permissions to function properly.

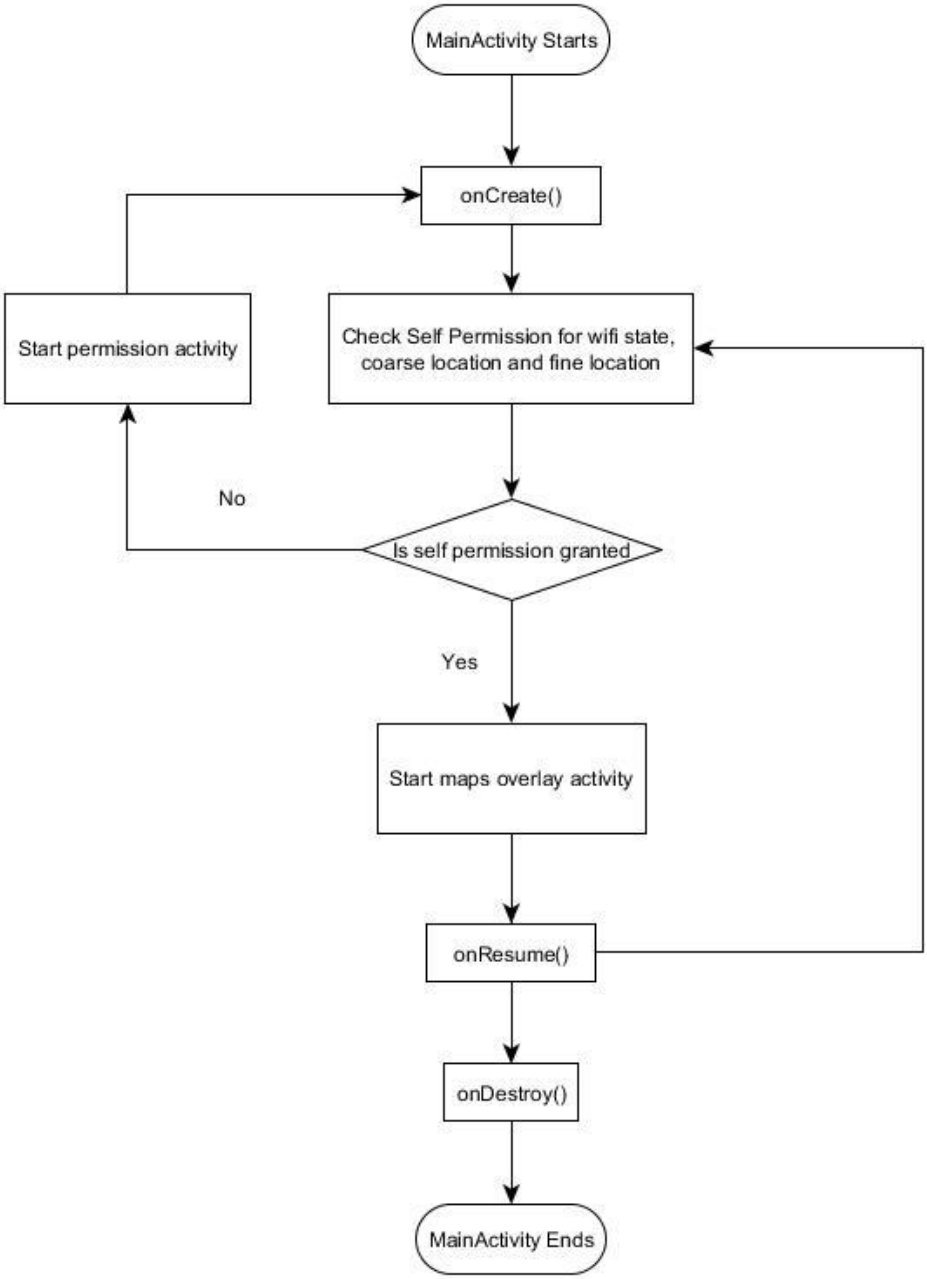

Figure 49. Main activity flow chart.

The figure below shows a top level flow chart for the **Permission activity**.

The PermissionActivity activity is started if the application needs to ask for the necessary permissions from the user. If the user declines to give the permissions, then the activity provides the user a Snackbar that can be started to launch the Android permission screen.

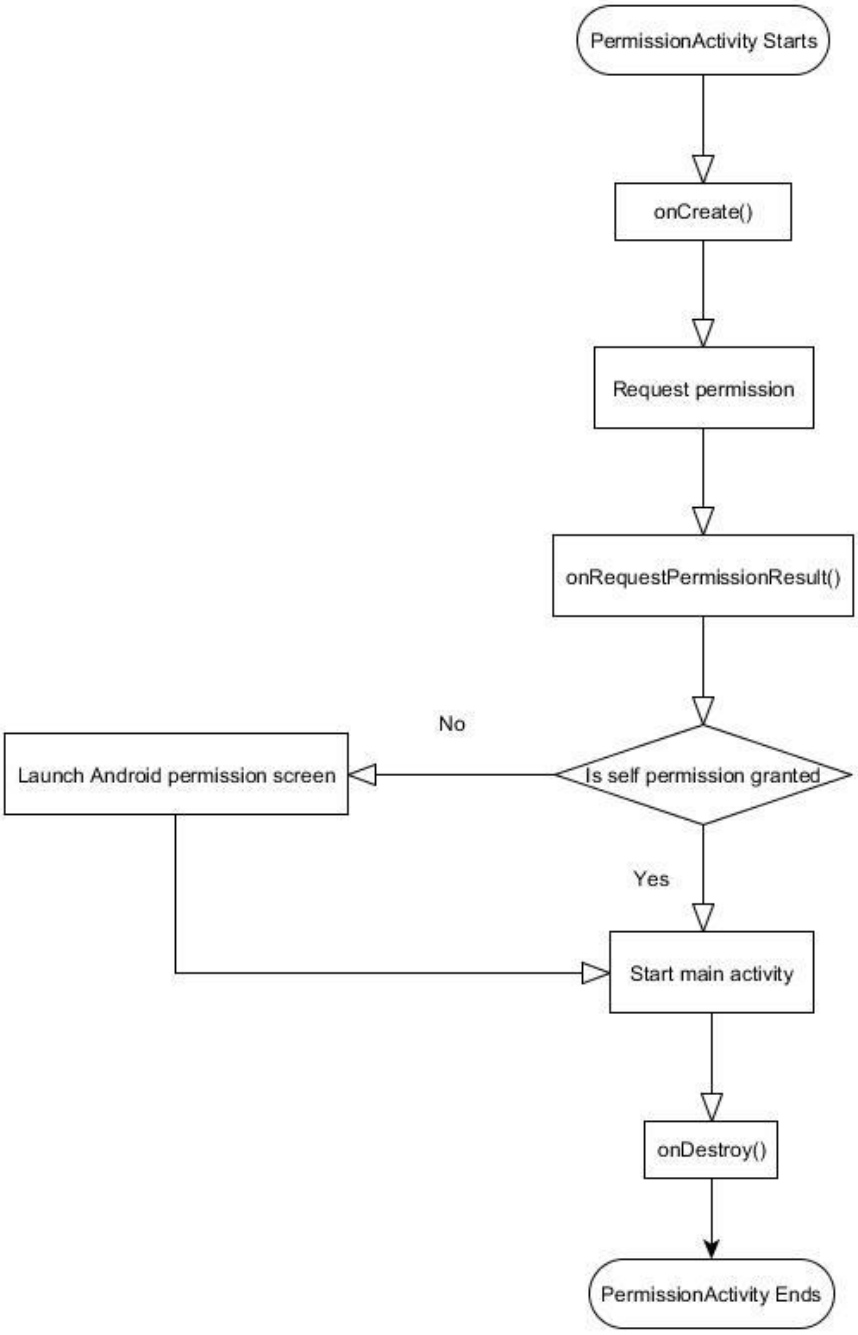

Figure 50. Permission activity flow chart.

The figure below shows a top level flow chart for the **Maps overlay activity**.

The MapsOverlayActivity activity implements the IndoorAtlas API for location and region listeners. The activity displays the appropriate floor plan on top of Google maps and displays the markers for the updated user location and destination in real time.

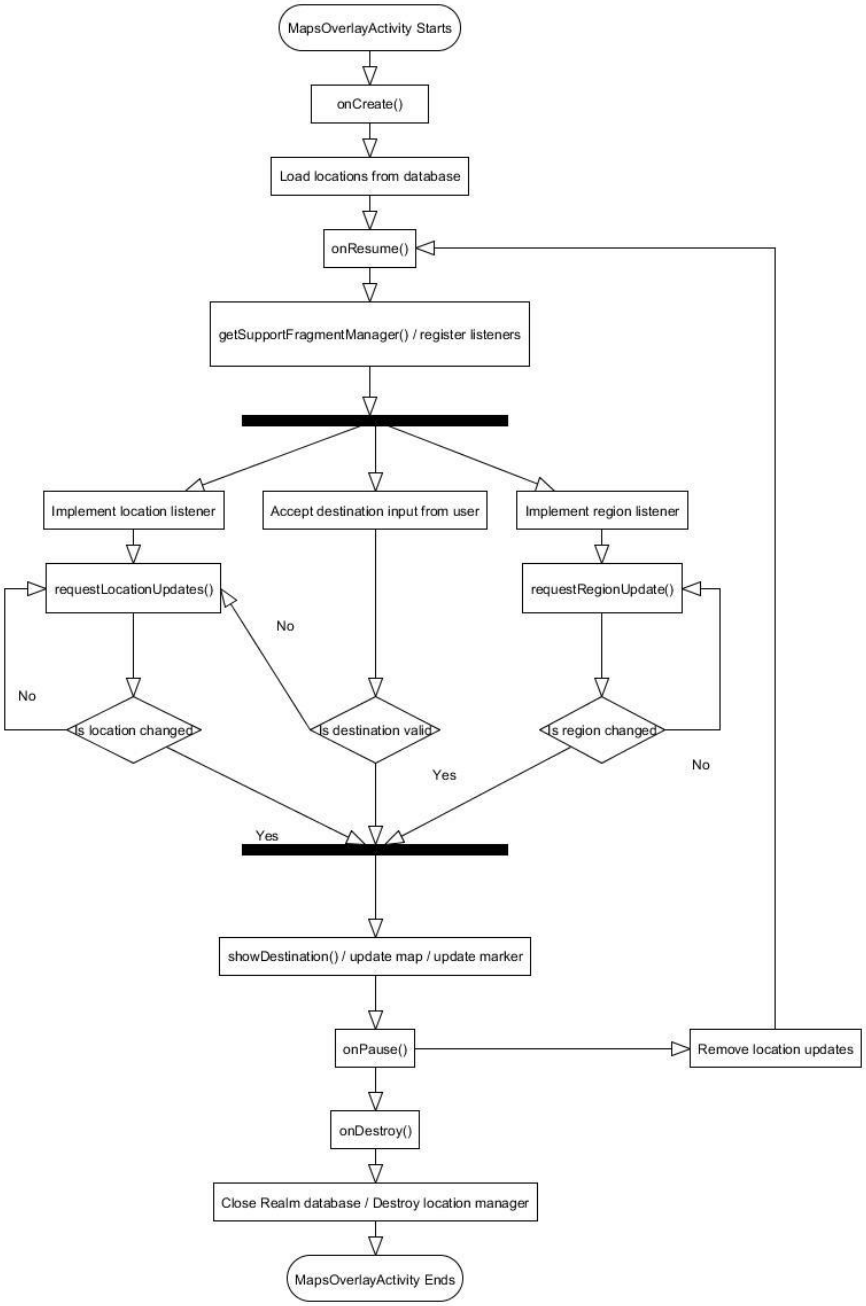

Figure 51. Maps overlay activity flow chart.

The figure below shows a top level flow chart for the **Search activity**.

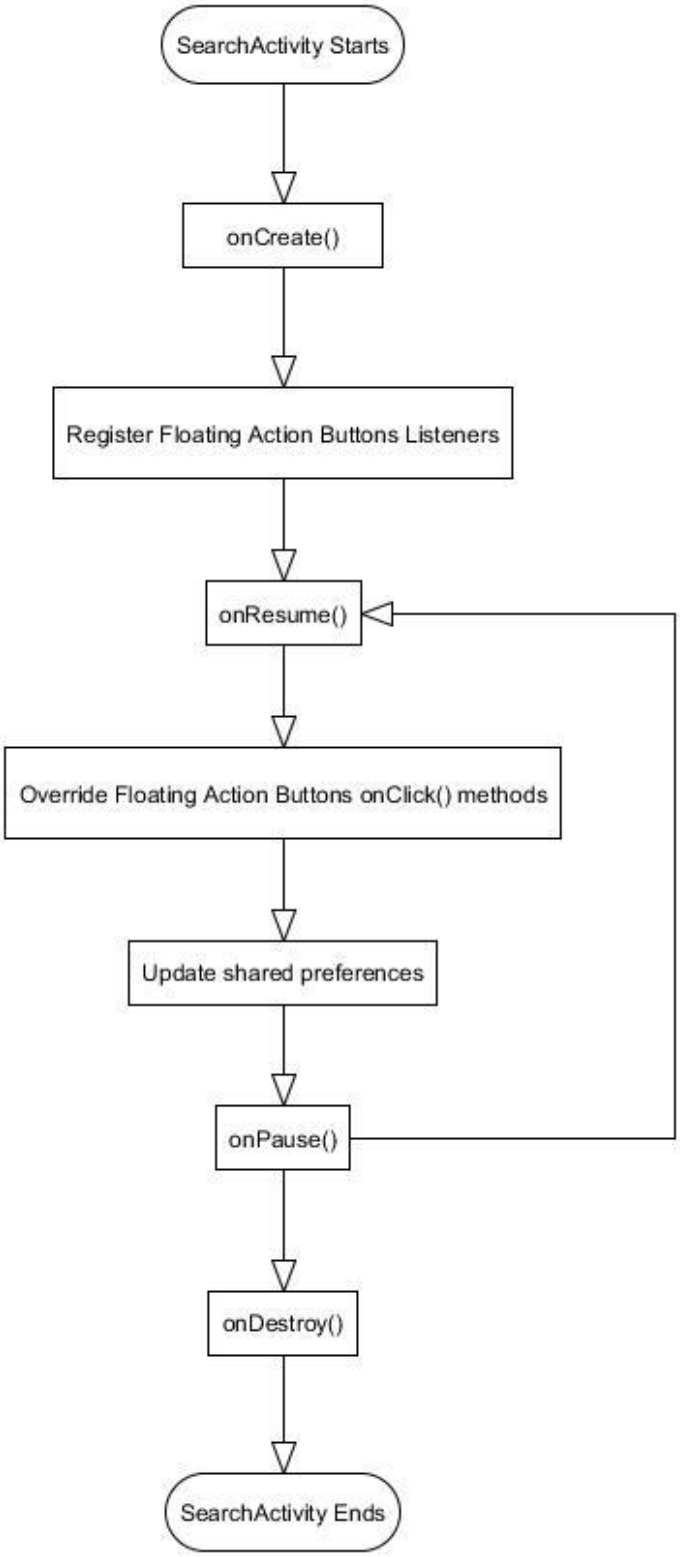

Figure 52. Search activity flow chart.

4.3 Class Diagram

The figure below shows the class diagram for the UWS Indoor Positioning application.

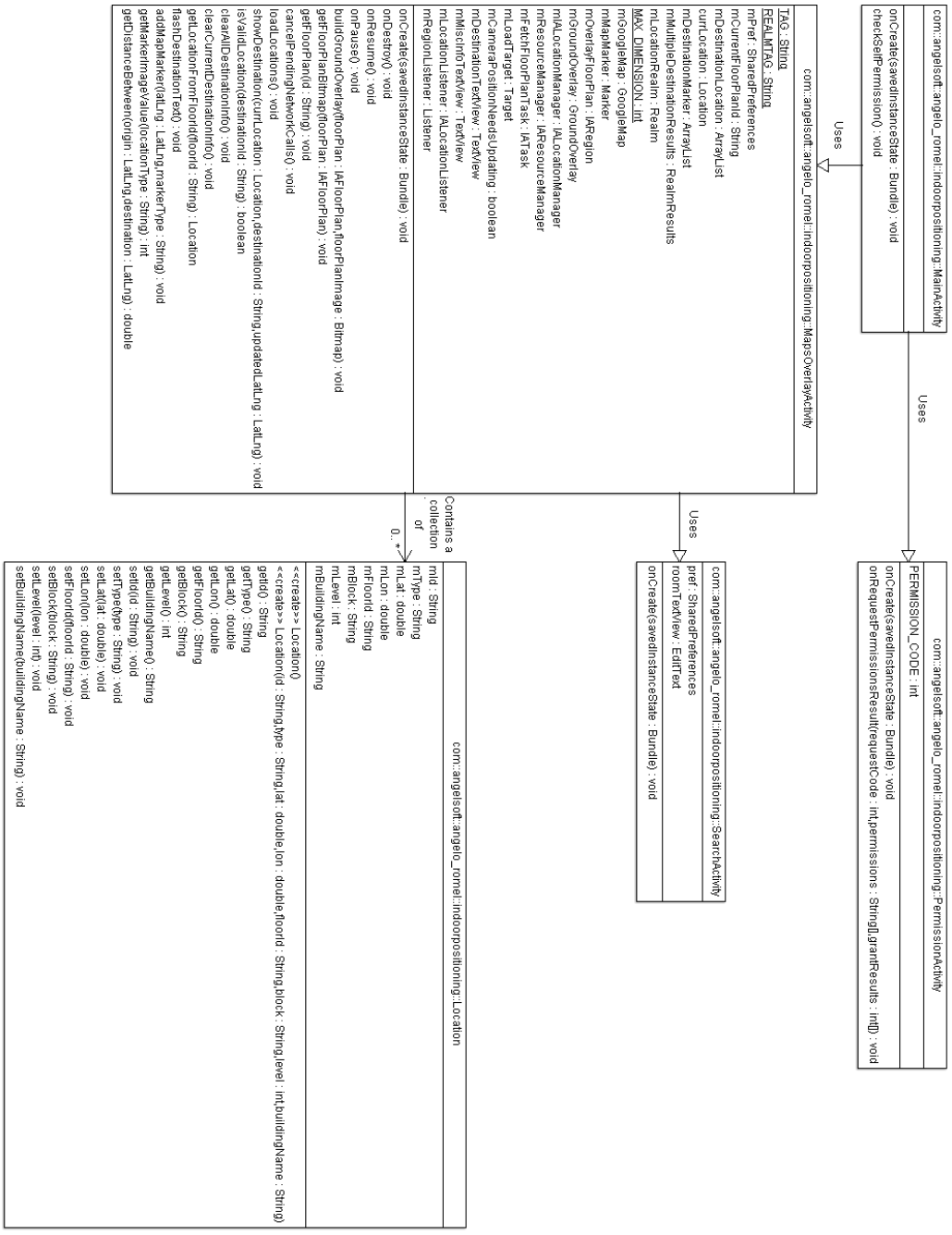

Figure 53. Class diagram.

4.4 Sequence Diagram
Class: MainActivity

[1] Method: onCreate()

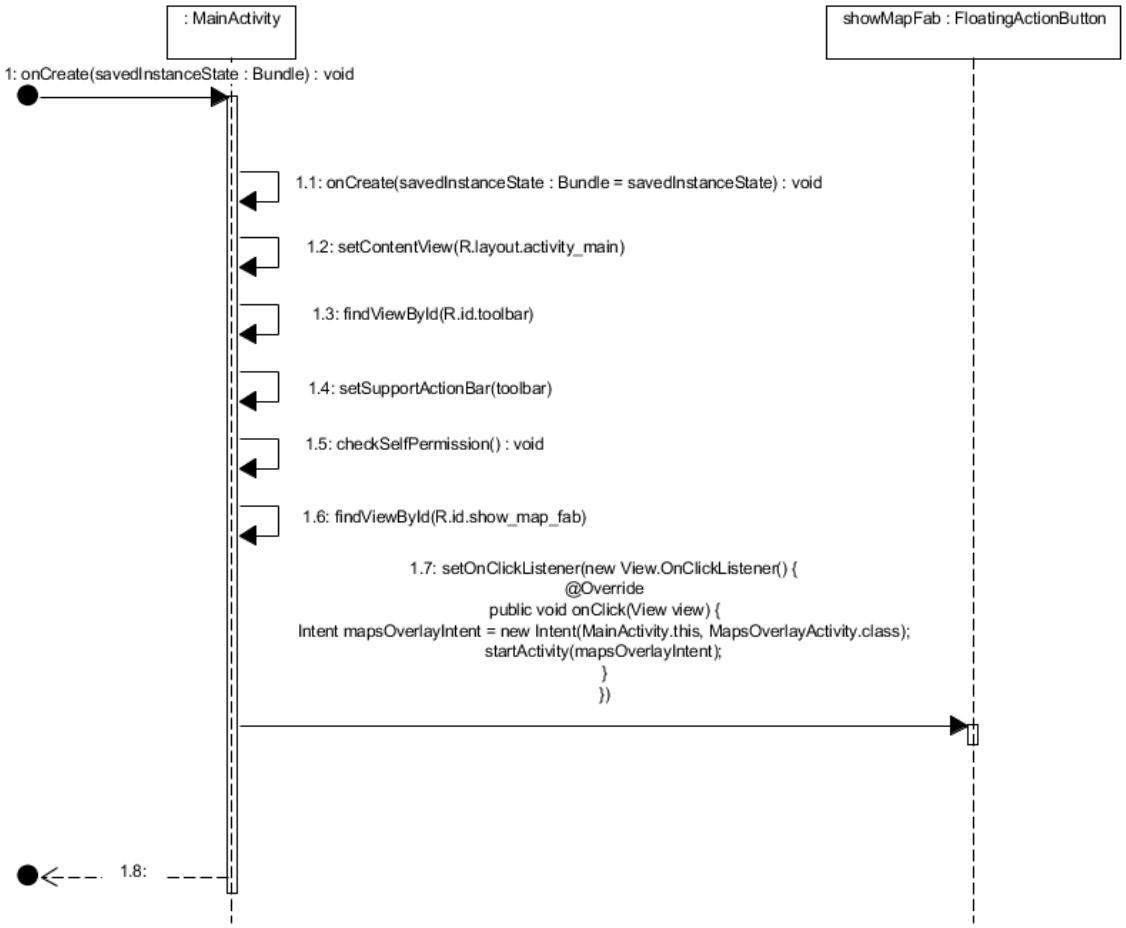

Figure 54. onCreate() sequence diagram.

[2] Method: checkSelfPermission()

Figure 55. checkSelfPermission() sequence diagram.

Class: PermissionActivity

[3] Method: onCreate()

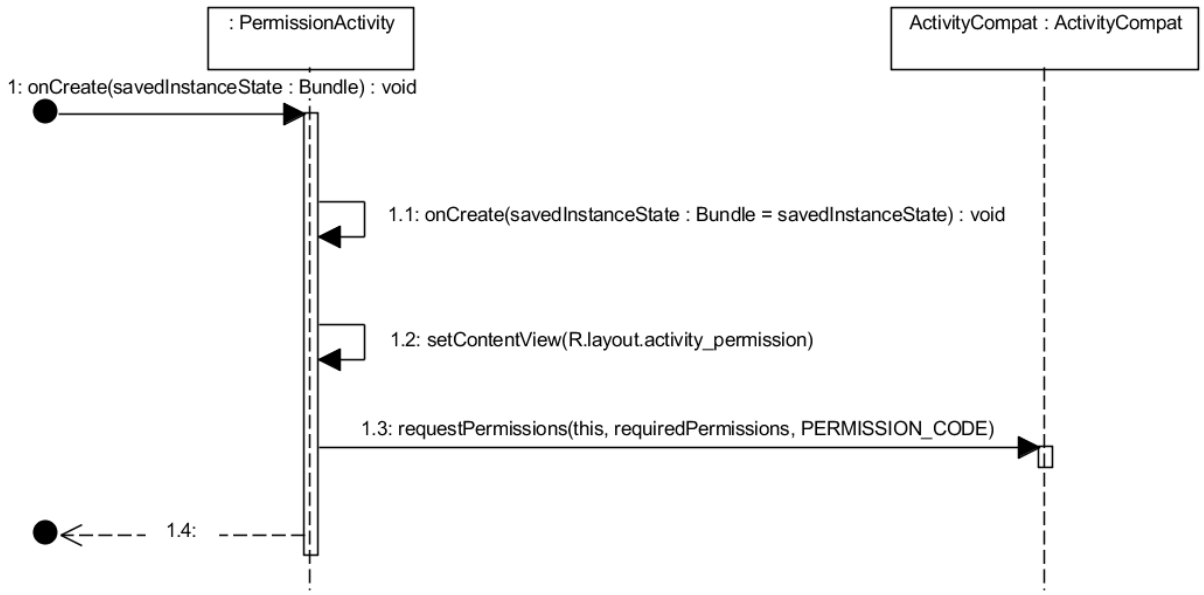

Figure 56. onCreate() sequence diagram.

[4] Method: onResquestPermissionResult()

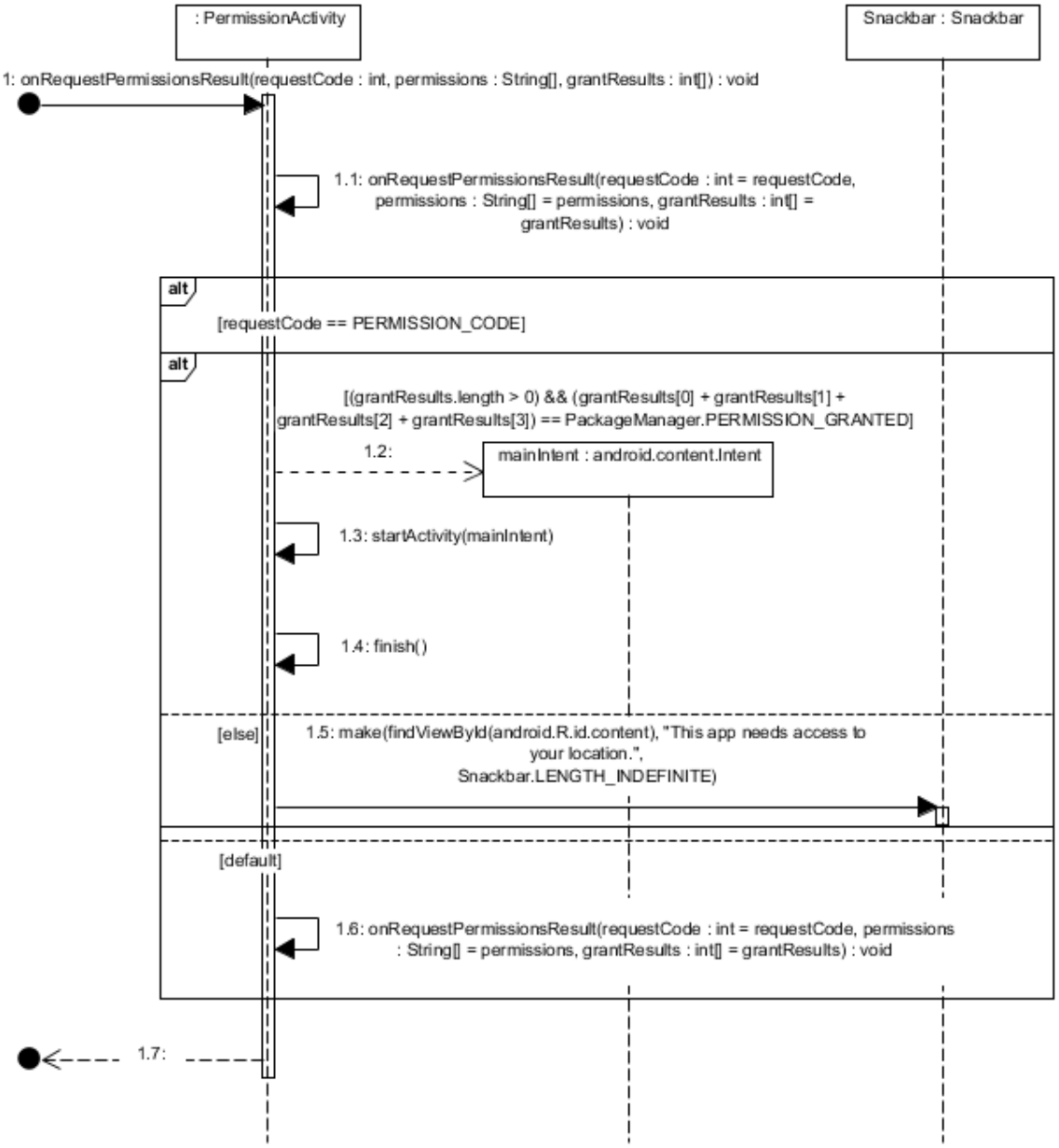

Figure 55. onRequestPermissionResult() sequence diagram.

Class: MapsOverlayActivity

[5] Method: onCreate()

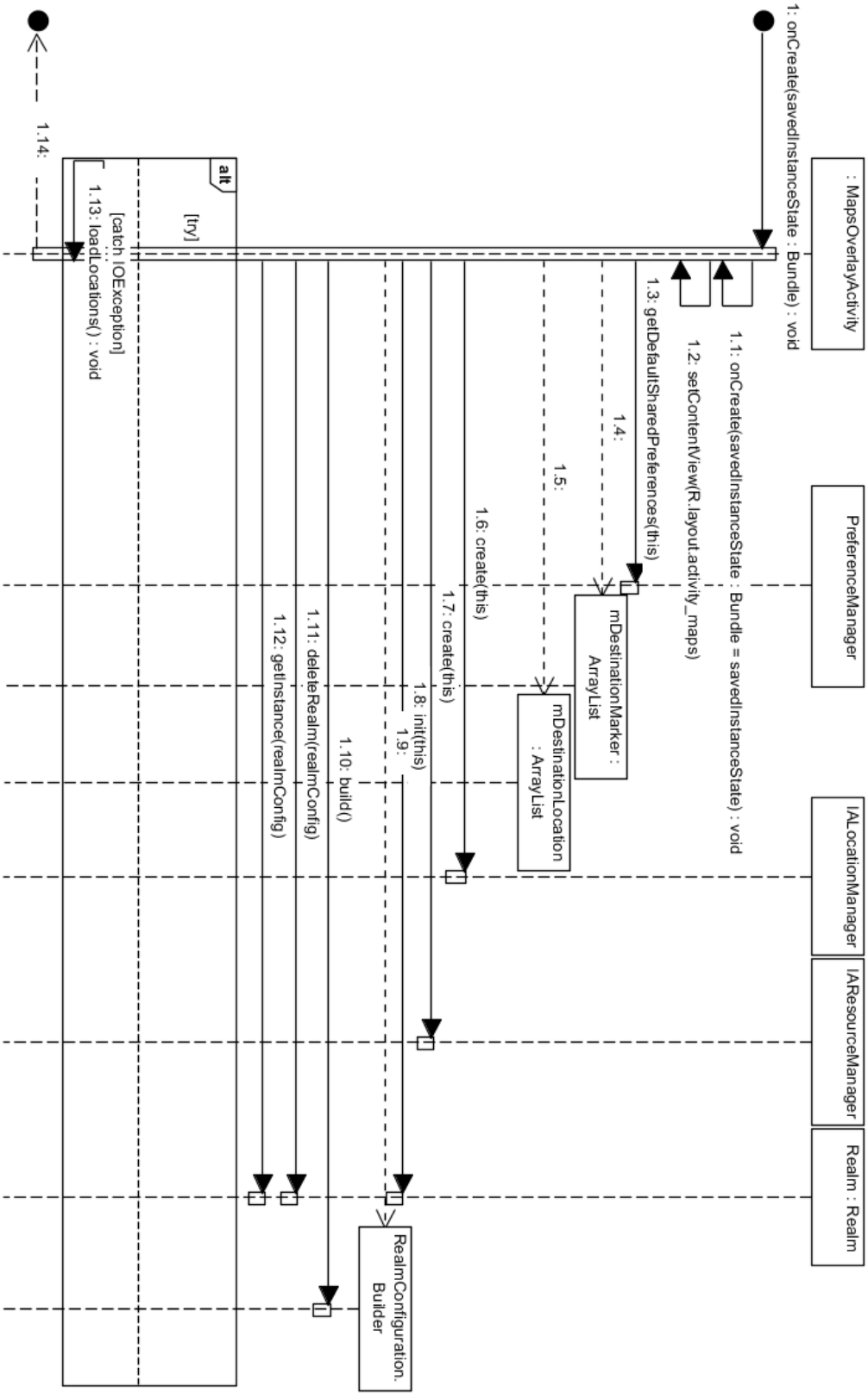

Figure 56. onCreate() sequence diagram.

[6] Method: loadLocations()

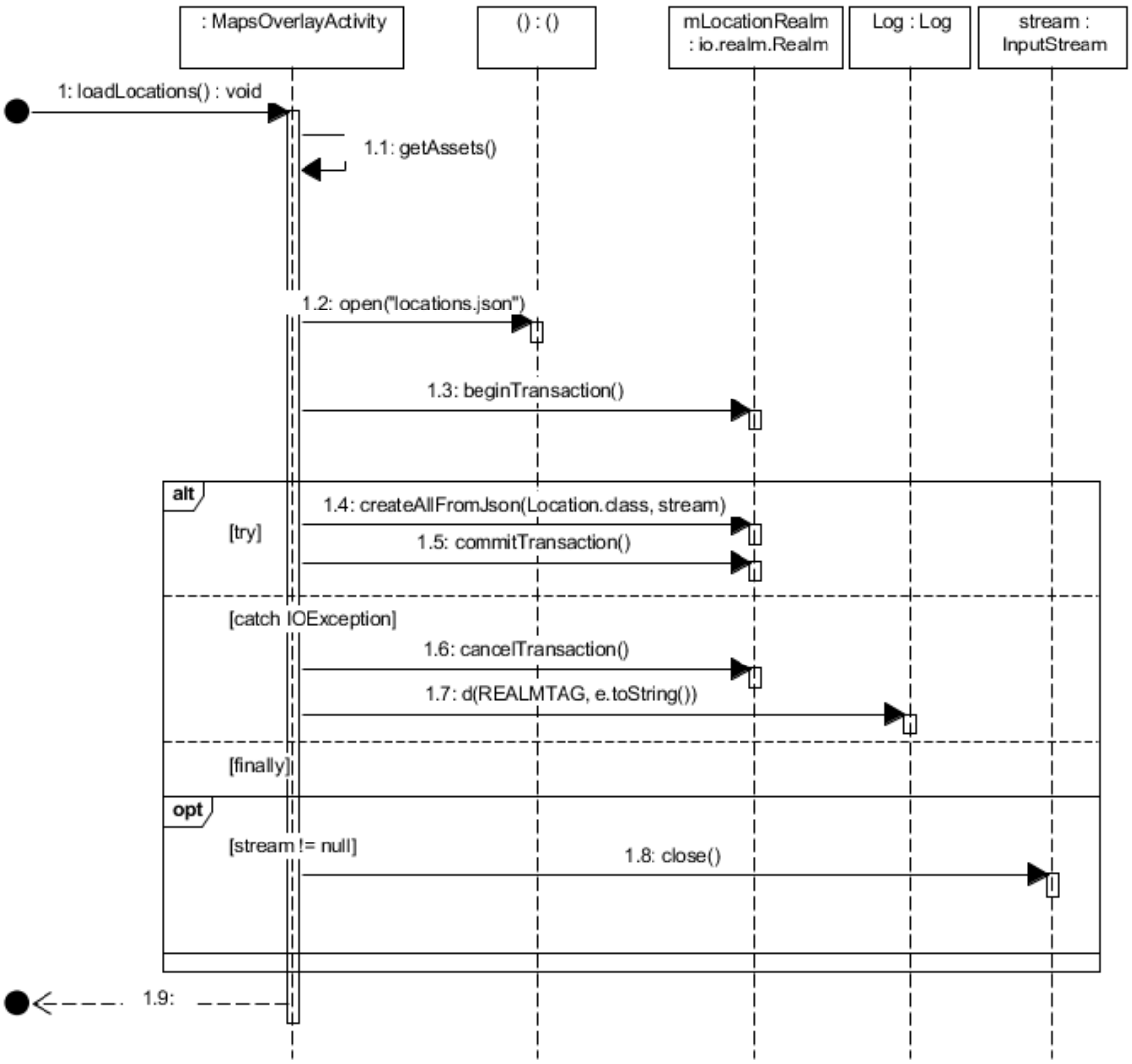

Figure 57. loadLocations() sequence diagram.

[7] Method: getFloorPlan()

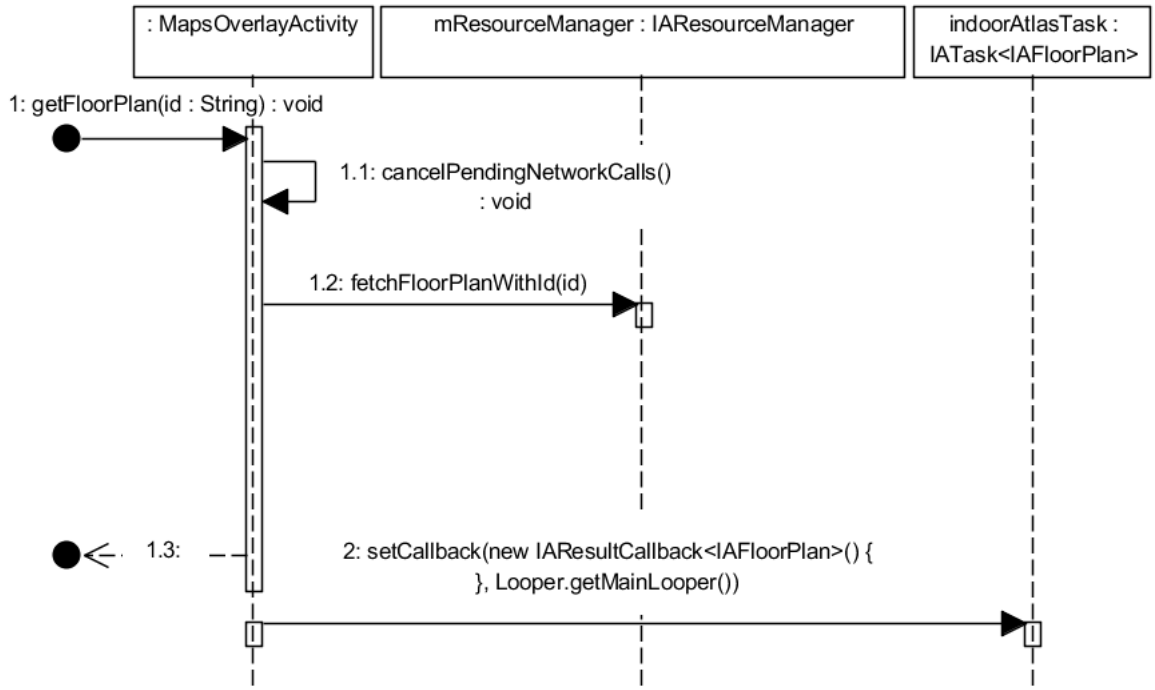

Figure 58. getFloorPlan() sequence diagram.

[8] Method: getDistanceBetween()

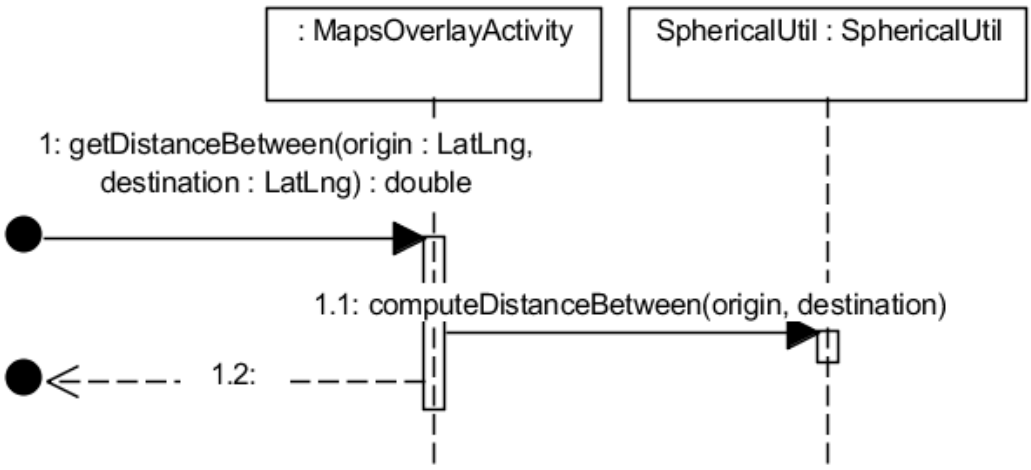

Figure 59. getDistanceBetween() sequence diagram

[9] Method: getLocationFromFloorId()

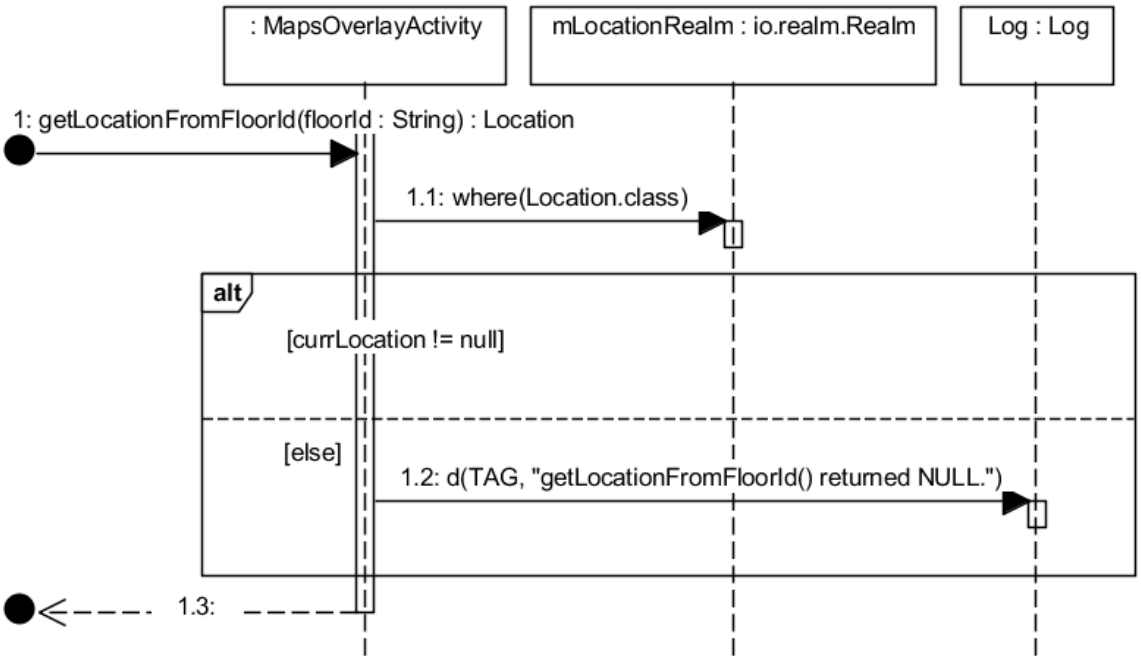

Figure 60. getLocationFromFloorId() sequence diagram.

[10] Method: addMapMarker()

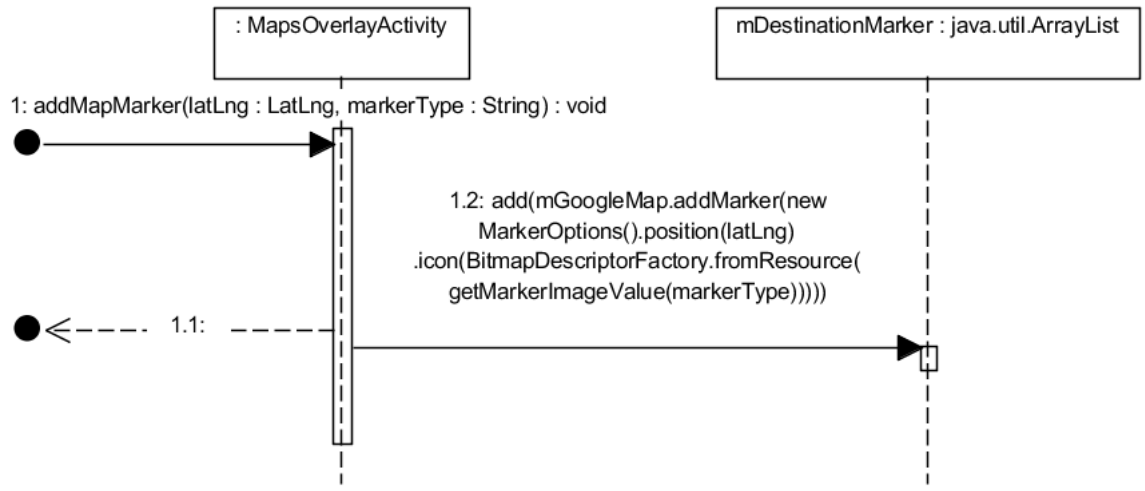

Figure 61. addMapMarker() sequence diagram.

Class: SearchActivity

[11] Method: onCreate()

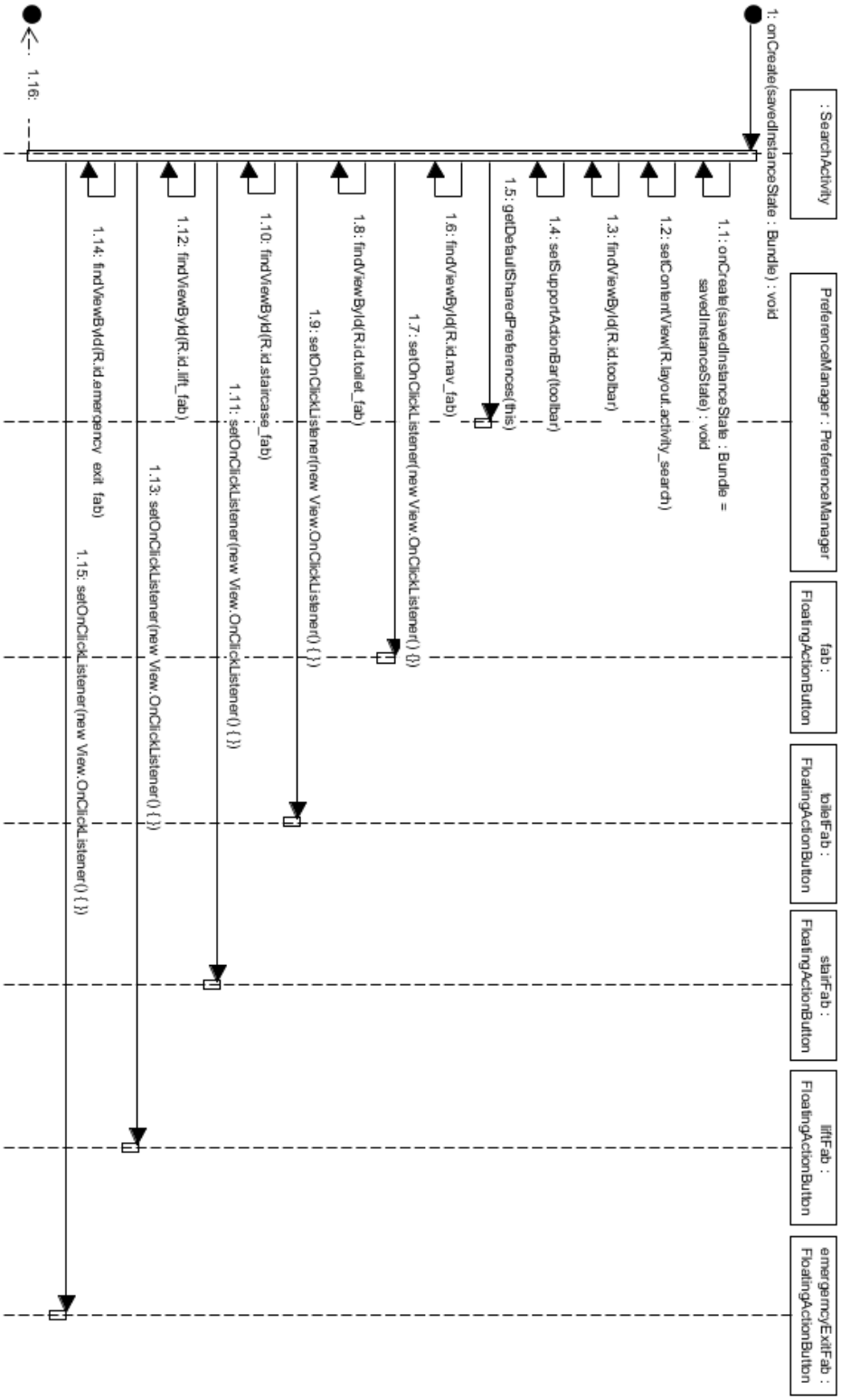

Figure 62. onCreate() sequence diagram.

4.5 User Interface Design

The figure below shows the user interface design of the Main (MainActivity.java) activity, which is the main entry point of the application.

Figure 63. Main activity UI.

The figure below shows the permission (PermissionActivity.java) activity screen requesting for user permission to access the device's location.

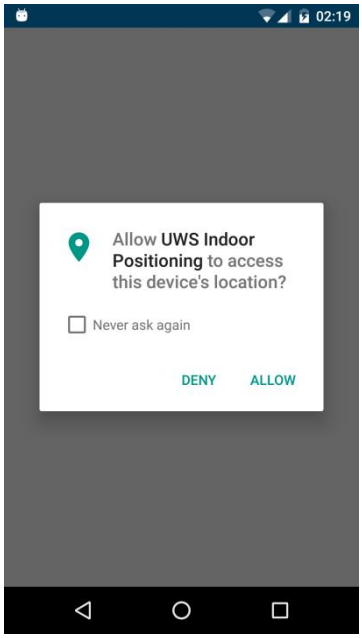

Figure 64. Permission Activity UI.

The figure below shows the permission (PermissionActivity.java) activity showing a Snackbar that will launch the Android permission screen, giving the user another chance at allowing the application to access the device's location.

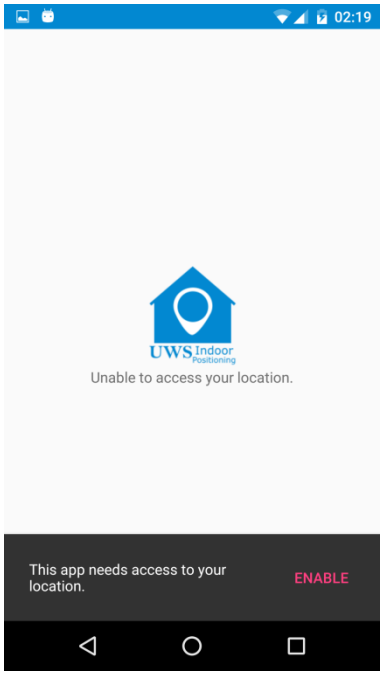

Figure 65. Permission Snackbar.

The figure below shows the application's Android permission screen.

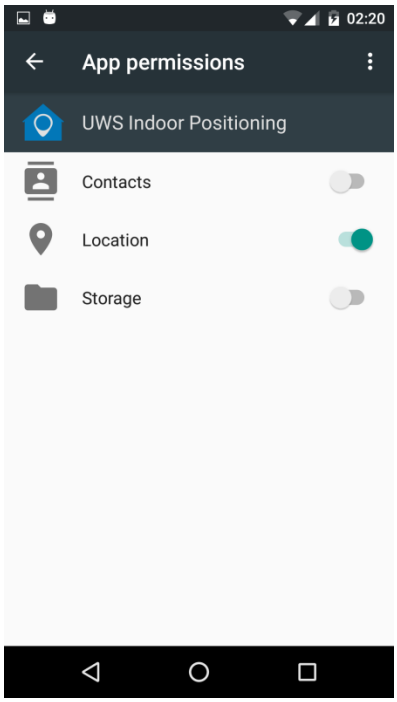

Figure 66. Android permission screen.

Figure below shows the screen for the maps overlay (MapsOverlayActivity.java) activity screen that shows the user's current location within the building.

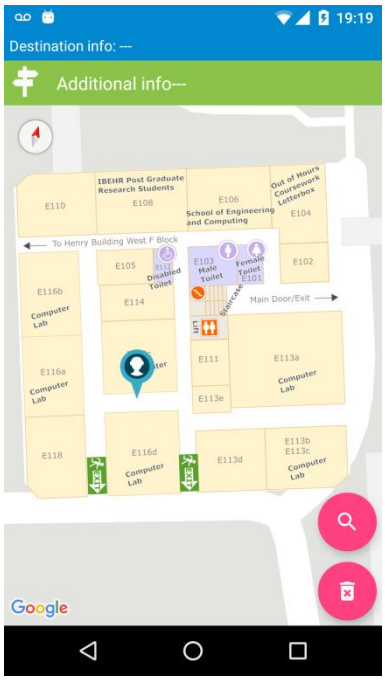

Figure 67. Maps overlay activity UI showing user's current location.

Figure below shows the screen for the search (SearchActivity.java) activity screen. The user has the option to type the destination as room number or use the quick search buttons to locate staircases, lifts, toilets and emergency exits within the building.

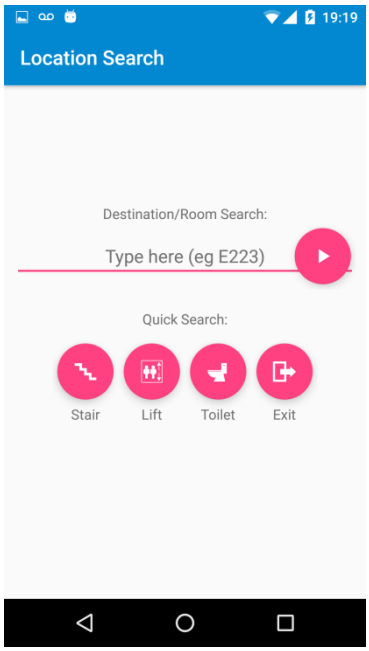

Figure 68. Search activity UI.

The use can manually search for a room number by typing into the input text as shown in the figure below.

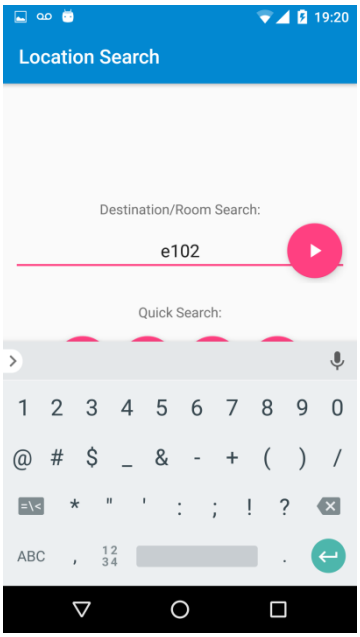

Figure 69. Input room destination.

The destination is shown as a red marker. Information about the destination such as room number, building name, block and floor/level is shown in the top most part of the screen. Other information is also displayed such as the floor/level within the building where the destination is located and the distance (in meters) from the user. If the destination is located in a different floor of level of the building, then the application will indicate which floor/level the destination is located and would show the direction/location and distance of the stairs.

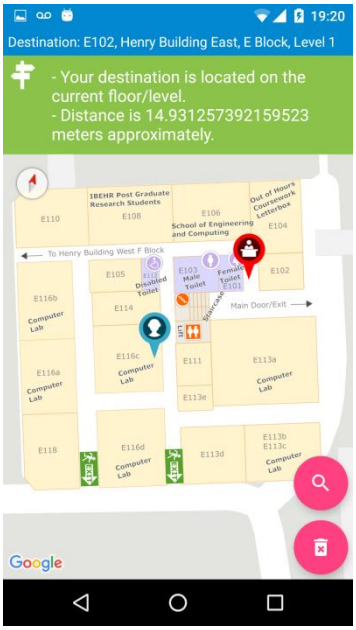

Figure 70. Destination marker.

The distance and position of the user is updated and shown to the user in real time.

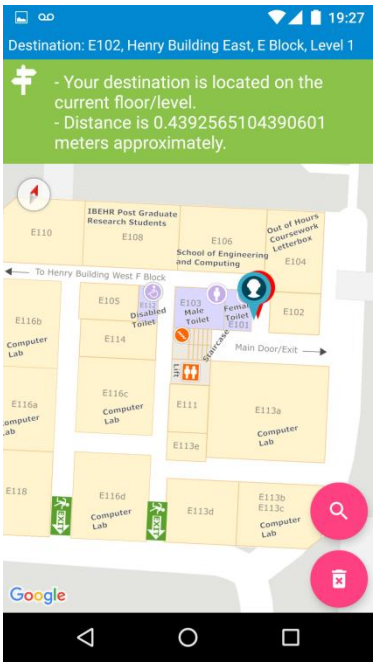

Figure 71. Updated distance.

The quick search options would show the locations of staircases, lifts, toilets and emergency exits within the building.

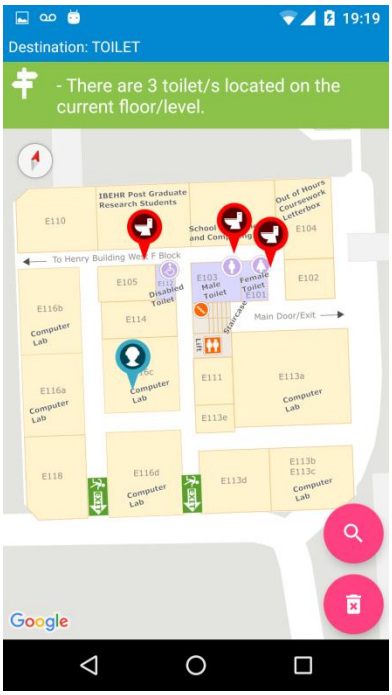

Figure 72. Toilet/s location.

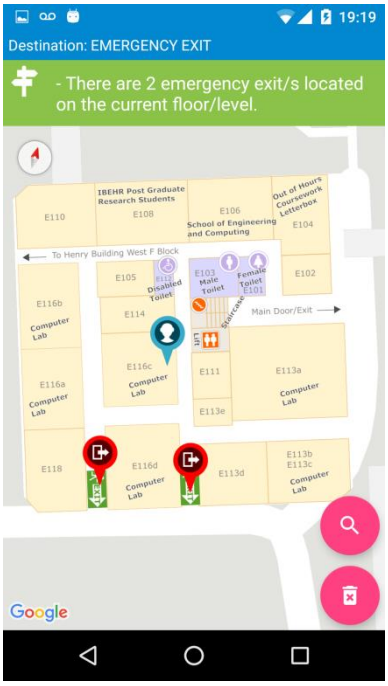

Figure 73. Emergency exit/s location.

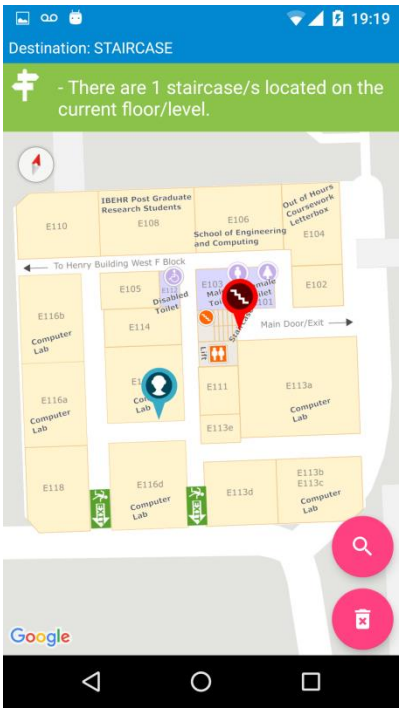

Figure 74. Staircase location.

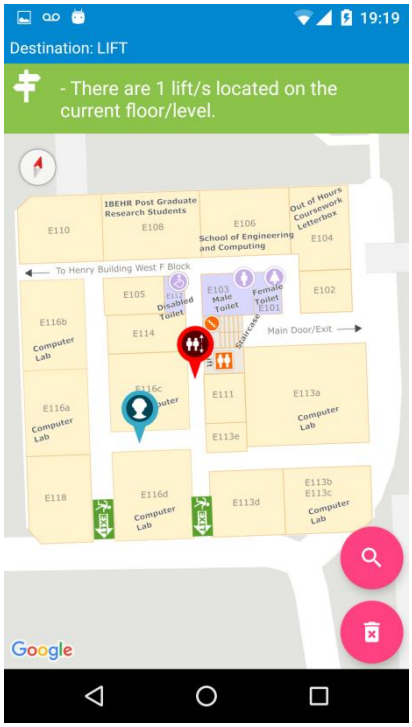

Figure 75. Lift location.

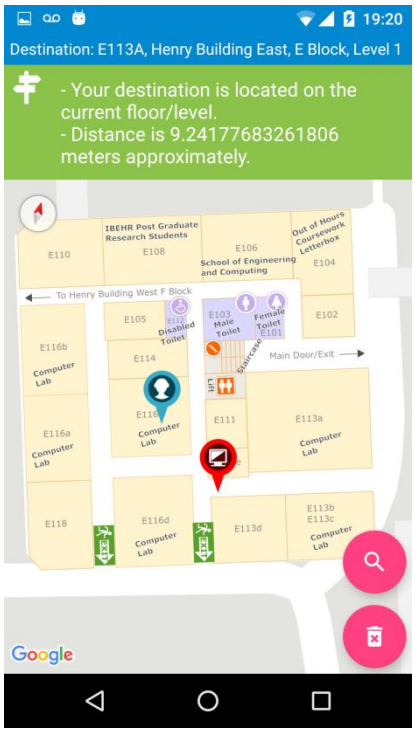

Figure 76. E113 lab.

4.6 Map Marker Design

The different designs for the map markers are shown in the figures below. The markers were designed using Inkscape (inkscape.org) and are stored in the /res/drawable folder of the Android Studio project structure.

Figure77. User

Figure 78. Room/Classroom

Figure 79. Computer lab

Figure 80. Lift

Figure 81. Staircase

Figure 82. Toilet

Figure 83. Emergency exit

5. Implementation

5.1 Naming convention

The following naming convention is followed:

- Class names start with an uppercase letter, then follows the camel case notation.
- Non-public and non-static field names start with a lowercase m, then follows the camel case notation.
- Methods start with a lowercase letter, then follows the camel case notation.
- Constants are all uppercase letters. If the variable name consists of more than one word, then they are separated with an underscore.
- XML names are all lowercase letters. If the name consists of more than one word, then they are separated with an underscore.

5.2 Source Code

Project configuration options in Gradle files.

build.gradle (Project level)

The code listing below shows the top-level build file containing configuration options common to all modules.

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.3.0'
        classpath 'io.realm:realm-gradle-plugin:3.0.0'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

build.gradle (Module level)

The build file for specific modules (dependencies, build types, flavours, etc.) is shown below.

```
apply plugin: 'com.android.application'
apply plugin: 'realm-android'

android {
    compileSdkVersion 23
    buildToolsVersion '25.0.0'
    defaultConfig {
        applicationId "com.angelsoft.angelo_romel.indoorpositioning"
        minSdkVersion 21
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    productFlavors {
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.android.support:design:23.4.0'
    compile 'com.indooratlas.android:indooratlas-android-sdk:2.3.1'
    compile 'com.google.android.gms:play-services:8.1.0'
    compile 'com.squareup.picasso:picasso:2.5.2'
    compile 'com.davemorrissey.labs:subsampling-scale-image-view:3.2.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.google.maps.android:android-maps-utils:0.3.4'
    testCompile 'junit:junit:4.12'
}

repositories {
    maven {
        url "http://indooratlas-ltd.bintray.com/mvn-public"
    }
}
```

Manifest file

AndroidManifest.xml

Manifest file that contains all the important information about the application such as required permissions, features it needs, activity and meta-data descriptions.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.angelsoft.angelo_romel.indoorpositioning">

    <!-- Required permissions. Manifest declaration Prior to Android 6. -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission
android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <!--
    The SDK already uses three hardware sensors plus Wi-Fi when available, so the
declaration and use of
    these features in the manifest are optional. However declaring these features
in the manifest will improve
    the application's performance, but will result in incompatibility issues with
device not supporting
    these features.
    -->
    <uses-feature
        android:name="android.hardware.sensor.accelerometer"
        android:required="true" />
    <uses-feature
        android:name="android.hardware.sensor.compass"
        android:required="true" />
    <uses-feature
        android:name="android.hardware.sensor.gyroscope"
        android:required="true" />
    <uses-feature
        android:name="android.hardware.wifi"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- Define application credentials for IndoorAtlas. Values are in
strings.xml file. -->
        <meta-data
            android:name="com.indooratlas.android.sdk.API_KEY"
            android:value="@string/indooratlas_api_key" />
        <meta-data
            android:name="com.indooratlas.android.sdk.API_SECRET"
            android:value="@string/indooratlas_api_secret" />
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="@string/googlemaps_api_key" />
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
```

```
        <!-- Activities. -->
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_map"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="portrait"
            android:configChanges="keyboardHidden|orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".PermissionActivity"
            android:label="@string/title_activity_map"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity
            android:name=".MapsOverlayActivity"
            android:label="@string/title_activity_map"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="portrait"
            android:configChanges="keyboardHidden|orientation"/>
        <activity
            android:name=".SearchActivity"
            android:label="@string/title_activity_search"
            android:theme="@style/AppTheme.NoActionBar" />
    </application>

</manifest>
```

locations.json

JSON file (local assets folder) that contains the data to be loaded into the Location class and the Realm database.

```
[
  {"id":"E116B",
    "type":"computer lab",
    "lat":55.84302740463703,
    "lon":-4.430496197562462,
    "floorId":"1e773449-0c7e-4a5d-b571-0128ff6403c6",
    "block":"E",
    "level":1,
    "buildingName":"Henry Building East"},

  {"id":"E114",
    "type":"room",
    "lat":55.84302740463703,
    "lon":-4.430496197562462,
    "floorId":"1e773449-0c7e-4a5d-b571-0128ff6403c6",
    "block":"E",
    "level":1,
    "buildingName":"Henry Building East"},
      .

      .

      .

      .

]
```

Model

Location.java

The model for the location object.

```
public class Location extends RealmObject {

    @PrimaryKey//sets id to be the primary key of the Location object of
"location_realm" database.
    private String mId;
    private String mType;
    private double mLat;
    private double mLon;
    private String mFloorId;
    private String mBlock;
    private int mLevel;
    private String mBuildingName;

    //default constructor
    public Location() {}

    //custom constructor
    public Location(String id, String type, double lat, double lon, String floorId,
                    String block, int level, String buildingName) {
        this.mId = id;
        this.mType = type;
        this.mLat = lat;
        this.mLon = lon;
        this.mFloorId = floorId;
        this.mBlock = block;
        this.mLevel = level;
        this.mBuildingName = buildingName;
    }

    //standard getters
    public String getId(){
        return this.mId;
    }

    public String getType() {
        return this.mType;
    }

    public double getLat() {
        return this.mLat;
    }

    public double getLon() {
        return this.mLon;
    }

    public String getFloorId() {
        return this.mFloorId;
    }

    public String getBlock() {
        return this.mBlock;
    }

    public int getLevel() {
        return this.mLevel;
    }

    public String getBuildingName() {
        return this.mBuildingName;
```

```
    }

    //standard setters
    public void setId(String id) {
        this.mId = id;
    }

    public void setType(String type) {
        this.mType = type;
    }

    public void setLat(double lat) {
        this.mLat = lat;
    }

    public void setLon(double lon) {
        this.mLon = lon;
    }

    public void setFloorId(String floorId) {
        this.mFloorId = floorId;
    }

    public void setBlock(String block) {
        this.mBlock = block;
    }

    public void setLevel(int level) {
        this.mLevel = level;
    }

    public void setBuildingName(String buildingName) {
        this.mBuildingName = buildingName;
    }

}
```

Activities

MainActivity.java

The main activity and entry point of the application. The permissions that the application needs to run are checked in this activity. The permission activity is launched if permission is required; otherwise the maps overlay activity is launched instead.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        checkSelfPermission();

        FloatingActionButton showMapFab = (FloatingActionButton)
findViewById(R.id.show_map_fab);
        showMapFab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```
                Intent mapsOverlayIntent = new Intent(MainActivity.this,
MapsOverlayActivity.class);
                startActivity(mapsOverlayIntent);
            }
        });
    }//end onCreate()

    private void checkSelfPermission() {
        if(ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_WIFI_STATE) +
                ContextCompat.checkSelfPermission(this,
android.Manifest.permission.CHANGE_WIFI_STATE) +
                ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) +
                ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            Intent permissionIntent = new Intent(this, PermissionActivity.class);
            startActivity(permissionIntent);
            this.finish();
        }
    }
}
```

PermissionActivity.java

Displays the request permission dialog box. Also displays a Snackbar at the bottom of the screen with a link to start the Android permission settings if the user has denied the permission required to run the application properly.

```
public class PermissionActivity extends AppCompatActivity {
    private final int PERMISSION_CODE = 123;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_permission);

        String[] requiredPermissions = {
                Manifest.permission.CHANGE_WIFI_STATE,
                Manifest.permission.ACCESS_WIFI_STATE,
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_FINE_LOCATION
        };
        ActivityCompat.requestPermissions(this, requiredPermissions,
PERMISSION_CODE);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        switch (requestCode) {
            case PERMISSION_CODE:

                if ((grantResults.length > 0) && (grantResults[0] + grantResults[1]
+
                        grantResults[2] + grantResults[3]) ==
PackageManager.PERMISSION_GRANTED) {
                    Intent mainIntent = new Intent(this, MainActivity.class);
                    startActivity(mainIntent);
```

```
                    finish();
                }
                else {
                    Snackbar.make(findViewById(android.R.id.content), "This app
needs access to your location.",
                            Snackbar.LENGTH_INDEFINITE).setAction("ENABLE",
                            new View.OnClickListener() {
                                @Override
                                public void onClick(View v) {
                                    Intent intent = new Intent();

intent.setAction(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
                                    intent.addCategory(Intent.CATEGORY_DEFAULT);
                                    intent.setData(Uri.parse("package:" +
getPackageName()));
                                    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);

intent.addFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
                                    startActivity(intent);
                                }
                            }).show();
                }
                break;
            default:
                super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        }

    }
}
```

MapsOverlayActivity.java

Registers and implements the IndoorAtlas location and region listeners. The updated location is displayed on the floor plan which sits on top of Google Map. The Realm database is populated from a model class and a json file that contains the GPS coordinates of different locations within a building.

```
public class MapsOverlayActivity extends FragmentActivity {

    private static final String TAG = MapsOverlayActivity.class.getSimpleName();
    private static final String REALMTAG = "Realm Database Log:";
    private SharedPreferences mPref;

    //current location info
    private String mCurrentFloorPlanId;

    //destination location info
    private ArrayList<Location> mDestinationLocation;
    //destination markers
    private ArrayList<Marker> mDestinationMarker;

    //object that will hold realm database default instance.
    private Realm mLocationRealm;

    // Maximum dimension of the floor plan image
    private static final int MAX_DIMENSION = 2048;

    private GoogleMap mGoogleMap;
```

```
    private Marker mMapMarker;
    private IARegion mOverlayFloorPlan = null;
    private GroundOverlay mGroundOverlay = null;
    private IALocationManager mIALocationManager;
    private IAResourceManager mResourceManager;
    private IATask<IAFloorPlan> mFetchFloorPlanTask;
    private Target mLoadTarget;
    private boolean mCameraPositionNeedsUpdating = true;

    private TextView mDestinationTextView;
    private TextView mMiscInfoTextView;

    /**
     * Listener that handles location change events.
     * Note - this method definition is based on IndoorAtlas example code.
     */
    private IALocationListener mLocationListener = new IALocationListenerSupport()
{
        Location mCurrLocation;//current location
        @Override
        public void onLocationChanged(IALocation location) {
            mCurrLocation = getLocationFromFloorId(mCurrentFloorPlanId);
            if(!mPref.getString("destinationId", "none").equalsIgnoreCase("none")
&& mCurrentFloorPlanId != null) {
                LatLng updatedLatLng = new LatLng(location.getLatitude(),
location.getLongitude());
                showDestination(mCurrLocation, mPref.getString("destinationId",
"none"), updatedLatLng);
            }

            if (mGoogleMap == null) {
                return;
            }

            LatLng latLng = new LatLng(location.getLatitude(),
location.getLongitude());
            if (mMapMarker == null) {
                try {
                    mMapMarker = mGoogleMap.addMarker(new
MarkerOptions().position(latLng)

.icon(BitmapDescriptorFactory.fromResource(getMarkerImageValue("main marker"))));
                }
                catch(Exception e) {
                    Log.d(TAG, "location is null");
                }
            } else {
                mMapMarker.setPosition(latLng);
            }

            // update camera position on location change
            if (mCameraPositionNeedsUpdating) {
                mGoogleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng,
17.5f));
                mCameraPositionNeedsUpdating = false;
            }
        }
    };

    /**
     * Method that listens for region changes and displays the appropriate overlay
if needed
     * Note - this method definition is based on IndoorAtlas example code.
     */
    private IARegion.Listener mRegionListener = new IARegion.Listener() {

        @Override
        public void onEnterRegion(IARegion region) {
```

```
            if (region.getType() == IARegion.TYPE_FLOOR_PLAN) {
                final String newId = region.getId();
                mCurrentFloorPlanId = region.getId();
                if (mGroundOverlay == null || !region.equals(mOverlayFloorPlan)) {
                    mCameraPositionNeedsUpdating = true; // move camera to new
floor plan.
                    if (mGroundOverlay != null) {
                        mGroundOverlay.remove();
                        mGroundOverlay = null;
                    }
                    mOverlayFloorPlan = region;
                    getFloorPlan(newId);
                } else {
                    mGroundOverlay.setTransparency(0.0f);
                }
                flashDestinationText();
            }
        }

        @Override
        public void onExitRegion(IARegion region) {
            if (mGroundOverlay != null) {
                mGroundOverlay.setTransparency(0.5f);
            }
        }

    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        mPref = PreferenceManager.getDefaultSharedPreferences(this);
        mDestinationMarker = new ArrayList<Marker>();
        mDestinationLocation = new ArrayList<Location>();
        findViewById(android.R.id.content).setKeepScreenOn(true);
        mIALocationManager = IALocationManager.create(this);
        mResourceManager = IAResourceManager.create(this);
        mDestinationTextView = (TextView)findViewById(R.id.destination_textview);
        mMiscInfoTextView = (TextView)findViewById(R.id.misc_info_textview);

        Realm.init(this);
        RealmConfiguration realmConfig = new RealmConfiguration.Builder().build();
        Realm.deleteRealm(realmConfig);
        mLocationRealm = Realm.getInstance(realmConfig);
        try {
            loadLocations();
        }
        catch(IOException e) {
            Toast.makeText(this, getString(R.string.load_from_json_error),
Toast.LENGTH_SHORT).show();
        }

        FloatingActionButton searchFab =
(FloatingActionButton)findViewById(R.id.search_fab);
        searchFab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent searchIntent = new Intent(MapsOverlayActivity.this,
SearchActivity.class);
                startActivity(searchIntent);
            }
        });

        FloatingActionButton clearFab =
(FloatingActionButton)findViewById(R.id.clear_fab);
        clearFab.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                clearAllDestinationInfo();
            }
        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mLocationRealm.close();
        mIALocationManager.destroy();
        mPref.edit().putString("destinationId", "none").apply();
        mDestinationLocation = null;
        mDestinationMarker = null;
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (mGoogleMap == null) {
            mGoogleMap = ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(map))
                    .getMap();
        }

        mIALocationManager.requestLocationUpdates(IALocationRequest.create(),
mLocationListener);
        mIALocationManager.registerRegionListener(mRegionListener);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mIALocationManager.removeLocationUpdates(mLocationListener);
        mIALocationManager.registerRegionListener(mRegionListener);
    }


    /**
     * Sets floor plan image retrieved from IndoorAtlas as ground overlay on Google
Maps
     * Note - this method definition is based on IndoorAtlas example code.
     */
    private void buildGroundOverlay(IAFloorPlan floorPlan, Bitmap floorPlanImage) {

        if (mGroundOverlay != null) {
            mGroundOverlay.remove();
        }

        if (mGoogleMap != null) {
            BitmapDescriptor bitmapDescriptor =
BitmapDescriptorFactory.fromBitmap(floorPlanImage);
            IALatLng latLngIndoorAtlas = floorPlan.getCenter();//Get coords of
floor plan's center.
            //pass coords to Google's LatLng
            LatLng latLngCenterFloorPlan = new LatLng(latLngIndoorAtlas.latitude,
latLngIndoorAtlas.longitude);
            GroundOverlayOptions floorPlanOverlay = new GroundOverlayOptions()
                    .image(bitmapDescriptor)
                    .position(latLngCenterFloorPlan, floorPlan.getWidthMeters(),
floorPlan.getHeightMeters())
                    .bearing(floorPlan.getBearing());

            mGroundOverlay = mGoogleMap.addGroundOverlay(floorPlanOverlay);
        }
    }

    /**
```

```
     * Download floor plan image from IndoorAtlas using the Picasso library.
     * Note - this method definition is based on IndoorAtlas example code.
     */
    private void getFloorPlanBitmap(final IAFloorPlan floorPlan) {

        final String floorPlanUrl = floorPlan.getUrl();

        if (mLoadTarget == null) {
            mLoadTarget = new Target() {

                @Override
                public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from)
{
                    buildGroundOverlay(floorPlan, bitmap);
                }

                @Override
                public void onPrepareLoad(Drawable placeHolderDrawable) {
                }

                @Override
                public void onBitmapFailed(Drawable placeHolderDraweble) {
                    Toast.makeText(MapsOverlayActivity.this, "Failed to load
bitmap",
                            Toast.LENGTH_SHORT).show();
                    mOverlayFloorPlan = null;
                }
            };
        }

        RequestCreator requestFloorPlan = Picasso.with(this).load(floorPlanUrl);

        final int imageWidth = floorPlan.getBitmapWidth();
        final int imageHeight = floorPlan.getBitmapHeight();

        if (imageHeight > MAX_DIMENSION) {
            requestFloorPlan.resize(0, MAX_DIMENSION);
        } else if (imageWidth > MAX_DIMENSION) {
            requestFloorPlan.resize(MAX_DIMENSION, 0);
        }

        requestFloorPlan.into(mLoadTarget);
    }


    /**
     * Get floor plan data from IndoorAtlas server.
     * Note - this method definition is based on IndoorAtlas example code.
     */
    private void getFloorPlan(String id) {

        // if there is already running task, cancel it
        cancelPendingNetworkCalls();

        final IATask<IAFloorPlan> indoorAtlasTask =
mResourceManager.fetchFloorPlanWithId(id);

        indoorAtlasTask.setCallback(new IAResultCallback<IAFloorPlan>() {

            @Override
            public void onResult(IAResult<IAFloorPlan> result) {

                if (result.isSuccess() && result.getResult() != null) {
                    getFloorPlanBitmap(result.getResult());
                } else {
                    if (!indoorAtlasTask.isCancelled()) {
                        Toast.makeText(MapsOverlayActivity.this,
                                "Error loading floor plan: " + result.getError(),
```

```
Toast.LENGTH_LONG)
                                    .show();
                            mOverlayFloorPlan = null;
                        }
                    }
                }
            }, Looper.getMainLooper());
            mFetchFloorPlanTask = indoorAtlasTask;

        }

        /**
         * Method to cancel current task.
         * Note - this method definition is based on IndoorAtlas example code.
         */
        private void cancelPendingNetworkCalls() {
            if (mFetchFloorPlanTask != null && !mFetchFloorPlanTask.isCancelled()) {
                mFetchFloorPlanTask.cancel();
            }
        }

        /**
         * Load database from json file in local assets folder into realm.
         */
        private void loadLocations() throws IOException{
            InputStream stream = getAssets().open("locations.json");
            mLocationRealm.beginTransaction();
            try {
                mLocationRealm.createAllFromJson(Location.class, stream);
                mLocationRealm.commitTransaction();
            }
            catch(IOException e) {
                mLocationRealm.cancelTransaction();
                Log.d(REALMTAG, e.toString());
            }
            finally {
                if(stream != null) {
                    stream.close();
                }
            }
        }

        /**
         * Show destination/s as marker on Google Maps overlay.
         */
        private void showDestination(Location currLocation, String destinationId,
LatLng updatedLatLng) {
            String miscDestInfo = "";//Builds the string to display additional location
information.
            RealmResults<Location> mMultipleDestinationResults;//multiple destinations
locations
            if (destinationId.equalsIgnoreCase("toilet") ||//destination as result of
quick search.
                    destinationId.equalsIgnoreCase("lift") ||
                    destinationId.equalsIgnoreCase("staircase") ||
                    destinationId.equalsIgnoreCase("emergency exit")) {
                mMultipleDestinationResults = mLocationRealm.where(Location.class)
                        .equalTo("floorId", currLocation.getFloorId())
                        .equalTo("type", destinationId.toLowerCase())
                        .findAll();
                Log.d(REALMTAG, destinationId + " destination results: " +
mMultipleDestinationResults.size());
                if(mMultipleDestinationResults.size() > 0) {
                    clearCurrentDestinationInfo();
                    mDestinationTextView.setText("Destination: " +
destinationId.toUpperCase());
                    miscDestInfo += "- There are " + mMultipleDestinationResults.size()
+ " " +
```

```
                        destinationId + "/s located on the current floor/level.";

                for(int i = 0; i < mMultipleDestinationResults.size(); i ++) {
                    LatLng latLng = new
LatLng(mMultipleDestinationResults.get(i).getLat(),
                            mMultipleDestinationResults.get(i).getLon());
                    addMapMarker(latLng, destinationId.toLowerCase());
                }
            }
            else {
                miscDestInfo += "- There are no " + destinationId + "/s located on
the current floor/level";
            }
            mMiscInfoTextView.setText(miscDestInfo);
        }
        else {//destination as result of user input.
            if (isValidLocation(destinationId)) {
                clearCurrentDestinationInfo();
                mDestinationLocation.add(mLocationRealm.where(Location.class)
                        .equalTo("id", destinationId.toUpperCase())
                        .findFirst());
                miscDestInfo = "- Your destination is";
                for (int i = 0; i < mDestinationLocation.size(); i++) {

                    mDestinationTextView.setText("Destination: " +
                            mDestinationLocation.get(i).getId().toUpperCase() +
                            ", " + mDestinationLocation.get(i).getBuildingName() +
                            ", " + mDestinationLocation.get(i).getBlock() + "
Block, " +
                            "Level " + mDestinationLocation.get(i).getLevel());

                    //destination is on the same building
                    if
(mDestinationLocation.get(i).getBuildingName().equalsIgnoreCase(currLocation.getBui
ldingName())) {
                        if
(mDestinationLocation.get(i).getFloorId().equalsIgnoreCase(currLocation.getFloorId(
))) {
                            //destination is on the same floor/level
                            miscDestInfo += " located on the current floor/level.";
                            LatLng latLng = new
LatLng(mDestinationLocation.get(i).getLat(),
                                    mDestinationLocation.get(i).getLon());
                            try {
                                for (int j = 0; j < mDestinationLocation.size();
j++) {
                                    addMapMarker(latLng,
mDestinationLocation.get(i).getType());

                                }
                            } catch (Exception e) {
                                Log.d(TAG, e.toString());
                            }
                        } else {
                            //destination is on another level/floor of the same
building
                            if ((mDestinationLocation.get(i).getLevel() -
currLocation.getLevel()) < 0) {
                                //destination is on a level below current location
                                miscDestInfo += " located "
                                        +
Math.abs((mDestinationLocation.get(i).getLevel() - currLocation.getLevel()))
                                        + " level/s below you.\n- Please use the
stairs.";
                            } else {
                                //destination is on a level above current location
                                miscDestInfo += " located "
                                        + (mDestinationLocation.get(i).getLevel() -
```

```
currLocation.getLevel())
                                        + " level/s above you.\n- Please use the
stairs.";
                            }
                            Location stairLocation =
mLocationRealm.where(Location.class).equalTo("floorId",
                                    currLocation.getFloorId()).equalTo("type",
"staircase").findFirst();
                            if (stairLocation != null) {
                                LatLng latLng = new LatLng(stairLocation.getLat(),
stairLocation.getLon());
                                addMapMarker(latLng, stairLocation.getType());
                            }
                        }
                    } else {
                        //destination is on a different building
                    }

                    //Display the distance (in meters) between current location and
destination
                    LatLng destinationLatLng = new
LatLng(mDestinationLocation.get(i).getLat(),
                            mDestinationLocation.get(i).getLon());
                    miscDestInfo += "\n- Distance is " +
getDistanceBetween(updatedLatLng, destinationLatLng) +
                            " meters approximately.";
                    mMiscInfoTextView.setText(miscDestInfo);

                }//end for
            } else {
                Toast.makeText(MapsOverlayActivity.this,
getString(R.string.destination_not_recognized),
                        Toast.LENGTH_SHORT).show();
                Log.d(TAG, getString(R.string.destination_not_recognized));
            }
        }
    }

    /**
     * Method that checks if a destination exists.
     */
    private boolean isValidLocation(String destinationId) {
        Location location = mLocationRealm.where(Location.class).equalTo("id",
destinationId.toUpperCase()).findFirst();
        if(location == null) {
            return false;
        }
        else {
            return true;
        }
    }

    /**
     * clear map data (marker/s) on destination change
     */
    private void clearAllDestinationInfo() {
        mDestinationTextView.setText("Destination info: ---");
        mMiscInfoTextView.setText("---");
        mPref.edit().putString("destinationId", "none").apply();
        if(mDestinationLocation != null) {
            mDestinationLocation.clear();
        }
        if(mDestinationMarker != null) {
            for(int i = 0; i < mDestinationMarker.size(); i ++) {
                mDestinationMarker.get(i).remove();
            }
            mDestinationMarker.clear();
        }
```

```
    }

    /**
     * Clear map data (marker/s) on floor change
     */
    private void clearCurrentDestinationInfo() {
        if(mDestinationMarker != null) {
            for(int i = 0; i < mDestinationMarker.size(); i ++) {
                mDestinationMarker.get(i).remove();
            }
            if(mDestinationLocation != null) {
                mDestinationLocation.clear();
            }
            mDestinationMarker.clear();
        }
    }

    /**
     * Gets the Location object based on floor id.
     */
    private Location getLocationFromFloorId(String floorId) {
        Location currLocation = mLocationRealm.where(Location.class)
                .equalTo("floorId", floorId)
                .findFirst();
        if(currLocation != null) {
            return currLocation;
        }
        else {
            Log.d(TAG, "getLocationFromFloorId() returned NULL.");
            return null;
        }
    }

    /**
     * Flash ui textview object on destination update.
     */
    private void flashDestinationText() {
        Animation anim = new AlphaAnimation(0.0f, 1.0f);
        anim.setDuration(100);
        anim.setStartOffset(20);
        anim.setRepeatCount(15);
        mMiscInfoTextView.startAnimation(anim);
    }

    /**
     * Display specific marker/s type on top of Google Map.
     */
    private void addMapMarker(LatLng latLng, String markerType) {
        mDestinationMarker.add(mGoogleMap.addMarker(new
MarkerOptions().position(latLng)
                .icon(BitmapDescriptorFactory.fromResource(
                        getMarkerImageValue(markerType)))));
    }

    /**
     * Get image resource value from the res folder.
     */
    private int getMarkerImageValue(String locationType) {
        int imageRes = R.drawable.ic_main_marker;
        switch(locationType) {
            case "main marker":
                imageRes = R.drawable.ic_main_marker;
                break;
            case "computer lab":
                imageRes = R.drawable.ic_comp_lab_marker;
                break;
            case "room":
                imageRes = R.drawable.ic_room_marker;
```

```
                break;
            case "toilet":
                imageRes = R.drawable.ic_toilet_marker;
                break;
            case "lift":
                imageRes = R.drawable.ic_lift_marker;
                break;
            case "staircase":
                imageRes = R.drawable.ic_staircase_marker;
                break;
            case "emergency exit":
                imageRes = R.drawable.ic_exit_marker;
                break;
        }
        return imageRes;
    }

    /**
     * Calculates and returns the distance in meters between 2 geo-coordinates.
     */
    private double getDistanceBetween(LatLng origin, LatLng destination) {
        return SphericalUtil.computeDistanceBetween(origin, destination);
    }

}
```

6. Testing

6.1 Functional Testing – Test Cases

Test Case: UWS Indoor Positioning and Navigation

Prepared by: Angelo Romel Lopez

Test Item: MainActivity

Test Case	Description	Expected Result	Actual Result	Remarks
1	Refuse Location Permission	Permission activity starts and shows Snackbar	Same as expected	
2	Allow Location Permission	Main Activity resumes	Same as expected	
3	Allow permission from Snackbar	Android permission settings starts	Same as expected	

Test Item: MapsOverlayActivity

Test Case	Description	Expected Result	Actual Result	Remarks
1	Press Search FAB	Search Activity starts	Same as expected	
2	Press Clear Destination FAB	Destination markers are removed if any	Same as expected	
3	Move to trigger location change	GPS coordinates are updated. Map marker is updated.	Same as expected	
4	Move between floors to trigger region change	Appropriate floorPlan is displayed. Location is updated	Same as expected	
5	Press Search FAB	Search Activity starts	Same as expected	
6	Press Clear Destination FAB	Destination markers are removed if any	Same as expected	

Test Item: SearchActivity

Test Case	Description	Expected Result	Actual Result	Remarks
1	Input valid room number	Destination marker is displayed on floor plan	Same as expected	
2	Input invalid room number	No destination marker is displayed on floor plan	Same as expected	
3	Leave room search blank	Will prompt for input	Same as expected	
4	Press quick search FAB for stair	Destination marker is displayed on floor plan	Same as expected	
5	Press quick search FAB for Lift	Destination marker is displayed on floor plan	Same as expected	
6	Press quick search FAB for toilet/s	Destination marker/s is displayed on floor plan	Same as expected	
7	Press quick search FAB for emergency exit/s	Destination marker/s is displayed on floor plan	Same as expected	

6.2 Unit Testing – Test Cases

Test Case: UWS Indoor Positioning and Navigation

Prepared by: Angelo Romel Lopez

Test Item: MapsOverlayActivity

Test Case	Item being tested	Test value	Actual Result	Remarks
1	loadLocations()	locations.json	ok	
2	loadLocations()	null	null	
3	loadLocations()	Malformed json	error	
4	showDestination()	Valid location	ok	
5	showDestination()	Invalid location	error	
6	isValidLocation()	Valid destination id	true	
7	isValidLocation()	null	error	
8	loadLocations()	Invalid location	false	
9	getLocationFromFloorId()	Valid floor id	ok	
10	getLocationFromFloorId()	Invalid floor id	error	
11	getLocationFromFloorId()	null	error	
12	addMapMarker()	Valid lat, lon, valid marker type	ok	
13	addMapMarker()	Valid lat, lon, invalid marker type	error	
14	addMapMarker()	invalid lat, lon, valid marker type	error	
15	addMapMarker()	invalid lat, lon, invalid marker type	error	
16	getMarkerImageValue()	Valid location type	ok	
17	getMarkerImageValue()	invalid location type	error	
18	getDistanceBetween()	Valid origin, valid destination	ok	
19	getDistanceBetween()	Valid origin, invalid destination	error	
20	getDistanceBetween()	Invalid origin, valid destination	error	

Conclusion

This project has served as evidence that geo-magnetic field data in indoor locations can be used to pinpoint the location of a device inside buildings and structures. The need for extra hardware like beacons, routers and RFID is not necessary to achieve accurate positioning in the context of an indoor location. The IndoorAtlas IPS technology has proven that the unique geo-magnetic fingerprint of a building can be used to achieve 1-2 meters or less in positioning accuracy.

However, using the IndoorAtlas service has its downside as it relies on delivering the location service as a cloud platform. Connection to the cloud location service still relies on the strength of the building's Wi-Fi infrastructure or cellular coverage. The positioning accuracy degrades if the Wi-Fi environment quality or cellular coverage is poor.

The IndoorAtlas technology is still the most scalable solution for indoor localisation and has huge potential application for way-finding, indoor geo-fencing, proximity advertising, public venues and retail. The UWS Indoor Positioning application can be modified to work on museums, hospitals, hotels, supermarkets, shopping malls, airports and other indoor venues. The API and SDK is free and readily available, making it the most accessible indoor positioning solution. The only hardware investment is a smartphone and a laptop.

Critical Self-Appraisal

The project has been a great learning experience. I have improved my software development and coding skills further than I had ever hoped. It has been an experience full of obstacles and challenges, but I have overcome those through sheer drive and determination.

The project still needs a lot of work and improvement, but I am very confident that I can take it to the next level through my newfound knowledge and skills.

References

[1] IndoorAtlas, (No Date), How It Works - What Is Indoor Positioning Systems [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed December 30, 2016]

[2] Hans Fredrick, (No Date), Why Doesn't GPS Work Inside a Building? [Online] Tech in our everyday life, Available: http://techin.oureverydaylife.com/doesnt-gps-work-inside-building-18659.html [Accessed December 31, 2016]

[3] IndoorAtlas, (No Date), How Does Our Geomagnetic Technology Work? – How It Works [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed January 02, 2017]

[4] Greg Stirling, (June 2014), Magnetic Positioning, The Arrival of Indoor GPS [Online] Opus Research Report, Available: https://www.indooratlas.com/wp-content/uploads/2016/03/magnetic_positioning_opus_jun2014.pdf [January 02, 2017]

[5] IndoorAtlas, (No Date), How Do We Differ From Others? – How It Works [Online] IndoorAtlas, Available: http://www.indooratlas.com/how-it-works/ [Accessed January 02, 2017]

[6] IndoorAtlas, (No Date), Pricing [Online] IndoorAtlas, Available: https://www.indooratlas.com/pricing/#all-plans [Accessed January 02, 2017]

[7] No Author, (No Date), System Requirements [Online] Android Studio, Available: https://developer.android.com/studio/index.html [Accessed January 03, 2017]

[8] IndoorAtlas, (No Date), Preparation for Mapping [Online] IndoorAtlas, Available: http://docs.indooratlas.com/app/ [Accessed January 03, 2017]

[9] IndoorAtlas, (No Date), Android SDK 2.2.4, Minimum Requirements [Online] IndoorAtlas, Available: http://docs.indooratlas.com/android/getting-started.html [Accessed January 03, 2017]

[10] No Author, (No Date), Requesting Permissions at Runtime [Online] Android Developers, Available: https://developer.android.com/training/permissions/requesting.html [Accessed January 06, 2017]

[11] http://docs.indooratlas.com/android/2.2.2/ (accessed December 30, 2016)

[12] http://docs.indooratlas.com/android/2.2.2/com/indooratlas/android/sdk/IALocationManager.html (accessed December 30, 2016)

[13] http://docs.indooratlas.com/android/2.2.2/com/indooratlas/android/sdk/IALocationListener.html (accessed December 30, 2016)

[14] https://lighthouse.io/indoor-location-technologies-compared (accessed December 30, 2016)

[15] http://www.gps.gov/ (accessed December 30, 2016)

[16] https://www.google.com/patents/US20130177208, July 11, 2013 (accessed December 30, 2016)

[17] http://patents.justia.com/inventor/janne-haverinen, August 15, 2016 (accessed December 30, 2016)

[18] http://opusresearch.net/wordpress/ (Accessed January 02, 2017)

[19] http://www.indooratlas.com/api-license/ (Accessed January 02, 2017)

[20] http://www.indooratlas.com/mobile-license/ (Accessed January 02, 2017)

[21] http://www.indooratlas.com/terms/ (Accessed January 02, 2017)

[22] Inksape, nd. Inkscape. [computer program]. Inkscape Project. [Accessed January 3, 2017].

[23] https://developer.android.com/guide/components/fragments.html. (Accessed January 3, 2017)

[24] https://developer.android.com/guide/components/activities/activity-lifecycle.html. (Accessed January 3, 2017)

[24] Abiresearch.com. (2017). Will IndoorAtlas become the third pillar in Indoor Location in 2016? [online] Available at: https://www.abiresearch.com/blogs/will-indooratlas-become-third-pillar-indoor-location-2016/ [Accessed 20 Mar. 2017].

[25] Visual Paradigm International, nd. Visual Paradigm. [computer program]. Visual Paradigm International. Available at: <http://www.visual-paradigm.com/> [Accessed April 29, 2015].

[26] Jason E. Robbins, nd. ArgoUML. [computer program]. Tigris.org. Available at: <http://sourceforge.net/projects/argouml/> [Accessed April 29, 2015].