



**DUBLIN INSTITUTE
of TECHNOLOGY**

Institiúid Teicneolaíochta Bhaile Átha Cliath

Travel Assistant Final Year Project Report

**DT228
BSc in Computer Science**

Cillian McCabe

School of Computing
Dublin Institute of Technology

13/04/2018



Abstract

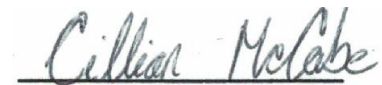
The aim of this project was to improve upon the various “realtime” and navigation Android applications. The reason behind selecting this idea originates from my college and work experience where my schedule each week was consistent, although not always the same for each day. The project determines a user’s schedule by using Google Fit data collected by an Android device and clusters the time and activities for each day using a machine learning algorithm.

By using machine learning and movement data, the schedule may provide accurate results instead of requiring the user to manually enter times and locations, which may be inaccurate. The application may then provide relevant information based on the current time and location of the user. This information consists of current transport times as well as user walking, cycling or running speeds. The project relies on the Google Fit API to record movement data as well as the Google Maps Directions API for creating routes between two locations and retrieving transport times.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A handwritten signature in dark ink, reading "Cillian McCabe", is written over a horizontal line.

Cillian McCabe

13/04/2018

Acknowledgements

I would like to thank my project supervisor Paul Bourke for weekly guidance to the goal of the project as well as advice for features and project management.

I would also like to thank my friends and family that tested the application and gave advice on design choices.

Table of Contents

1. INTRODUCTION	9
1.1 PROJECT BACKGROUND	9
1.2 PROJECT OBJECTIVES	9
1.3 PROJECT CHALLENGES	9
1.3.1 Using appropriate APIs	10
1.3.2 Machine Learning	10
1.3.3 Designing an Efficient Process of Creating the Schedule	10
1.3.4 Time Analysis	10
1.3.5 UI Design	10
1.3.6 Time Management	11
1.4 STRUCTURE OF THE DOCUMENT	11
1.4.1 Section 2: Research	11
1.4.2 Section 3: Design	11
1.4.3 Section 4: Architecture & Development	11
1.4.4 Section 5: System Validation	11
1.4.5 Section 6: Project Plan	12
1.4.6 Section 7: Conclusion	12
2. RESEARCH	13
2.1 BACKGROUND RESEARCH	13
2.2 ALTERNATIVE EXISTING SOLUTIONS	13
2.2.1 Nielsen's Heuristics	14
2.2.1.1 User Control and Freedom	14
2.2.1.2 Consistency and Standards	15
2.2.1.3 Recognition Rather Than Recall	18
2.2.1.4 Flexibility and Efficiency of Use	18
2.2.1.5 Aesthetic and Minimalist Design	18
2.2.2 Google Maps Schedule Update	20
2.3 TECHNOLOGIES RESEARCHED	22
2.3.1 Application Development Type	22
2.3.2 APIs	25
2.3.2.1 Fitness Recording	25
2.3.2.2 Mapping	26
2.3.3 Target Android SDK Version	27
2.3.4 Machine Learning	27
2.3.5 Data Storage	29
2.3.6 Data Visualisation	30
2.4 OTHER RELEVANT RESEARCH	31
2.4.1 Terms and Conditions	31
2.4.2 Data Privacy	31
2.4.3 User Feelings Consideration	32
2.5 FINDINGS	34
3. DESIGN	35
3.1 METHODOLOGY	35
3.1.1 Waterfall	35
3.1.2 Rapid Application Development	35
3.1.3 Agile	36
3.1.3.1 The Agile Methodology	36
3.1.3.2 Scrum	38
3.1.3.3 Extreme Programming	39
3.2 PROJECT COMPONENTS	41

3.2.1 UI	41
3.2.2 Source Code Layout	44
3.2.3 Features and Use Cases	45
3.2.3.1 View user's schedule	46
3.2.3.2 View route	47
3.2.3.3 Display visuals	48
3.3 USER SCENARIOS	48
3.3.1 User A Scenario	49
3.3.2 User B Scenario	49
3.3.3 Scenario conclusion	49
4. ARCHITECTURE & DEVELOPMENT	51
4.1 SYSTEM ARCHITECTURE	51
4.1.1 Diagram	51
4.1.2 Execution Environment	51
4.1.3 Realm Database	52
4.1.4 Third Party APIs	52
4.1.4.1 Weka	52
4.1.4.2 Google Fit	53
4.1.4.3 Google Maps	53
4.2 PROJECT COMPONENTS	53
4.2.1 Clustering	53
4.2.1.1 Scheduler.java	54
4.2.1.2 StartClusterService.java	54
4.2.1.3 ClusterService.java	56
4.2.1.4 ActivityTime.java	56
4.2.1.5 Location.java	57
4.2.1.6 ClusterInstance.java	57
4.2.1.7 Clusterer.java	57
4.2.1.8 Cluster.java	61
4.2.1.9 RawCluster.java	62
4.2.1.10 ActivityInformation.java	62
4.2.1.11 Activity.java	62
4.2.1.12 CollectAddressTask.java	62
4.2.1.13 ParseGeocodingJSON.java	65
4.2.1.14 Schedule.java	65
4.2.1.15 CreateNotification.java	65
4.2.2 Scheduling and Notifications	66
4.2.2.1 Scheduler.java	66
4.2.2.2 JobSchedulerReceiver.java	66
4.2.2.3 NotificationPublisher.java	67
4.2.2.4 NotificationsReceiver.java	68
4.2.3 Maps	68
4.2.3.1 MapsActivity.java	69
4.2.3.2 CollectRouteTask.java	71
4.2.3.3 ParseMapsJSON.java	72
4.2.3.4 Route Objects	72
4.2.4 Visuals	72
4.2.4.1 GraphActivity.java	73
4.2.4.2 BarChartFragment.java	73
4.2.4.3 SpeedChartFragment.java	74
4.2.4.4 DistanceChartFragment.java	75
4.2.5 UI	76
4.2.5.1 MainActivity.java	77
4.2.5.2 ViewPagerAdapter.java	78
4.2.5.3 ScheduleActivity.java	79
4.2.5.4 ScheduleAdapter.java	82
4.2.5.5 GraphActivity.java	82
4.2.5.6 MapsActivity.java	82
4.2.6 Time	86

5. SYSTEM VALIDATION	88
5.1 TESTING	88
5.1.1 White Box Testing	88
5.1.1.1 Clustering Process	88
5.1.1.2 Testing Notifications	89
5.1.1.3 Realm Database Testing	90
5.1.2 User Validation	91
5.1.2.1 UI Design	91
5.1.2.2 Visuals	92
5.1.2.3 Other Devices	92
5.2 DEMONSTRATION	92
6. PROJECT PLAN.....	94
7. CONCLUSION.....	95
7.1 PROJECT SUMMARY	95
7.2 FURTHER DEVELOPMENT	96
8. BIBLIOGRAPHY.....	97

Table of Figures

Figure 1 – User Control	15
Figure 2 - Navigation Drawer	16
Figure 3 - Bottom Navigation Menu.....	17
Figure 4 - TabLayout	17
Figure 5 - Google Fit.....	19
Figure 6 - Google Maps	20
Figure 7 - Google Maps Schedule Notification	21
Figure 8 - Google Maps Schedule Notification	22
Figure 9 - Google Fit Components	26
Figure 10 - Realm Speeds Compared to SQLite.....	30
Figure 11 - Google Maps Calorie Update.....	33
Figure 12 - Waterfall Diagram.....	35
Figure 13 - RAD Diagram	36
Figure 14 - Agile Diagram.....	37
Figure 15 - Scrum Diagram	38
Figure 16 - XP Diagram.....	40
Figure 17 - Schedule Layout.....	42
Figure 18 - Ordered List	43
Figure 19 - Basic Map Design	44
Figure 20 - Source Code Layout.....	45
Figure 21 - Use Cases	46
Figure 22 - User's Schedule	46
Figure 23 - Route	47
Figure 24 - Visuals.....	48
Figure 25 - System Architecture	51
Figure 26 - Clustering Class Diagram	54
Figure 27 - Schedule Update Notification	55
Figure 28 - Schedule Updated Notification	55
Figure 29 - Clustering Results	59

Figure 30 - Improved Clustering Results	59
Figure 31 - Empty Constructor	61
Figure 32 - Setting Values	62
Figure 33 - Geocoding JSON Results	64
Figure 34 - Sample Notification	65
Figure 35 - Scheduling and Notifications Class Diagram	66
Figure 36 - Boot Completed Receiver	67
Figure 37 - Available Update Notification	67
Figure 38 - Android O+ Notification Channels	68
Figure 39 - Boot Completed Receiver	68
Figure 40 - Maps Class Diagram	69
Figure 41 - Google Maps Directions API URL	70
Figure 42 - Directions JSON Results	71
Figure 43 - Visuals Class Diagram	72
Figure 44 - BarChartFragment	74
Figure 45 - SpeedChartFragment	75
Figure 46 - DistanceChartFragment	76
Figure 47 - UI Class Diagram	76
Figure 48 - Splashscreen	77
Figure 49 - getItem()	78
Figure 50 - getCount() & getPageTitle()	79
Figure 51 - Comparison Between Original and Current	80
Figure 52 - Cluster Day & Time Comparison	81
Figure 53 - Cluster Removal	82
Figure 54 - Google Maps Comparison	83
Figure 55 - MapsActivity Example	85
Figure 56 - Calendar Library	86
Figure 57 - Time Manipulation	87
Figure 58 - Mock Object	90
Figure 59 - Realm Data	91
Figure 60 - RawCluster	91

1. Introduction

1.1 Project Background

Over the duration of a college degree and work experience, individuals may find themselves fitting into a schedule. This schedule may vary day to day, particularly in college, as individuals have lectures at different times each day. To maintain this schedule is second nature to most however their process could be improved if they knew the exact time of each step in their journey. This author has a large interest in maintaining their own schedule and was partially inspired by the current implementation of a “smart scheduler” by Google in their Google Maps application. This author believes that through various methods of computing, a schedule can be refined and improved, and improving upon the current Google implementation as well. This interest is what inspired the foundation of this project.

1.2 Project Objectives

The primary objective of this project is to analyse a user’s physical movements and use this data to intelligently create a tailor-made schedule for the user. By using the schedule that was created, the user should receive relevant information such as the start and end time of the journey as well as the duration of the journey based on the speed they may walk, cycle or run, or based on the various transport times provided by Google Maps Directions. To learn and develop the user’s schedule, the following objectives were required:

- Develop a method of retrieving user’s movement data as well times, locations and speeds.
- Develop a schedule based on this data in an efficient, battery-friendly and secure way.
- Build a useful interface to display the most relevant item in the schedule and notify the user of their new journey.
- Generate visual charts of the data so users may see changes in each week’s schedule and potentially learn from their physical movement data.

1.3 Project Challenges

There are many challenges involved in developing an application that will involve data analysis and machine learning, while also incorporating design and third-party APIs. Below is a short discussion of the major challenges faced in the project.

1.3.1 Using appropriate APIs

Selecting the appropriate APIs required for the project is a challenge. Most notably in map selection as there is a wide variety of map-based APIs available, each with their own advantages and disadvantages. In selecting the movement recording API there was limited choice and ultimately Google Fit was the best selection.

1.3.2 Machine Learning

An inexperience in machine learning coupled with a lack of learning resources online for the specific method and data used proved to be a challenge in the early stages of the project which slowed development significantly in the early stages. Similarly developing a method of machine learning on the device added extra difficulties but would improve the security of the location data.

1.3.3 Designing an Efficient Process of Creating the Schedule

As the machine learning process occurred on the device it was important to be efficient with this process. As a result, a method of scheduling the machine learning process had to be implemented in such a way that suits the user, would not run on a metered Internet connection or drain the battery when it was valuable.

1.3.4 Time Analysis

Comparison between times was an unforeseen challenge that posed to be quite a complex challenge to overcome. A method of comparison relies on normalising the dates to milliseconds based on the Unix epoch and converting the time back to the local time format. However, as the Google Maps Directions API expects time in seconds, extra complexities were added.

1.3.5 UI Design

A challenge was faced in developing a simple design that will provide the data created in a clear and concise manner. Current existing solutions struggle at this hurdle due to the large quantity of data to provide the user as it is a challenge to display it in an efficient and clean way.

1.3.6 Time Management

Due to the magnitude of the final year of an undergraduate degree, an increased effort was made for each module and as the weeks passed, goals in the project may have been obstructed by the continuous assessment assigned by each module. This challenge was a constant battle throughout the development of the project however as the project matured I gained more experience in this area.

1.4 Structure of The Document

1.4.1 Section 2: Research

The research section will contain the valuable research that lead to the final decisions made in the development of this project. The background research will be discussed, including the increased interest in applications such as this project, as well as existing solutions and various technologies among other relevant research.

1.4.2 Section 3: Design

The design section details the design methodology selected as well as the project components design and features within the project.

1.4.3 Section 4: Architecture & Development

This section will detail the system architecture, including a technical discussion of the components and their development process, as well as relevant diagrams to further explain the communication between each component. The section will also discuss the challenges in development and the use of third-party APIs.

1.4.4 Section 5: System Validation

The system validation section described testing performed during the project and will demonstrate the project's features.

1.4.5 Section 6: Project Plan

The project plan includes a discussion and analysis of the project's life cycle from the initial concept to the final stage, detailing what changed and why.

1.4.6 Section 7: Conclusion

The final section will include a summary of the project and what was involved in its process. The section will also discuss potential future development on the application.

2. Research

2.1 Background Research

Before beginning the development of the project, it was important to understand whether the project idea is viable in modern day culture. In a report done by Yahoo, it was determined that the millennial generation are creating a demand for fitness products, representing 88 percent of consumers in the growth of fitness and health related apps (1). The millennial generation is the greatest generation in size when compared to previous generations and this fact combined with a growing digital culture results in a market that attempts to cater to every aspect of people's lives through some digital medium. In this specific case, there are many applications and gadgets available to promote fitness activities.

In the same report by Yahoo, it was concluded that 1 in 3 millennials share fitness content each week. Similarly, in research by Pew Internet (2) it was determined that 72% of users searched online for health information within the past year. It is safe to assume with the rise of digital fitness products, the availability of smartphones and the increase of interest into fitness, this number has increased.

When considering that fitness and health are of interest to the typical user it is important then to consider how to best deliver a product to these users that they can use in everyday life. This, combined with a personal interest in time management and fitness, resulted in the idea of this project. The project would target fitness minded users and allow them to record fitness activities while also using their own personal fitness information to improve estimates in travel and build a unique daily schedule and thus improve their everyday commutes.

To develop a project such as this it is first important to consider existing applications and products, as well as available technologies and the risks involved in fitness data and location data privacy.

2.2 Alternative Existing Solutions

When investigating existing solutions, it was difficult to find applications that would provide all the features that will be included in this project. It was then decided that the best way to investigate existing solutions would be to research individual applications that each perform a relevant function of the proposed project. The following applications were selected:

1. Google Fit for recording fitness
2. Google Maps for mapping and directions
3. MapMyWalk for an example of an application that would connect fitness and mapping.

2.2.1 Nielsen's Heuristics

Investigating these applications includes applying Nielsen's Heuristics to them and determining the positive and negative features of the three applications. This allows for the project to improve and be well refined based upon existing solutions. The most notable of Nielsen's heuristics include:

2.2.1.1 *User Control and Freedom*

Android applications would rely heavily on the native Android back button, whether software or hardware. The back button would allow for the user to switch intents in the event of an unwanted state. This allows for a native Android support for "undoing". It would also be common to include a method of escaping through the interface, such as an X or an arrow indicating going backwards. For example, in Google Fit, when viewing fitness data an X would appear to indicate the ability to close this screen as seen in Figure 1:

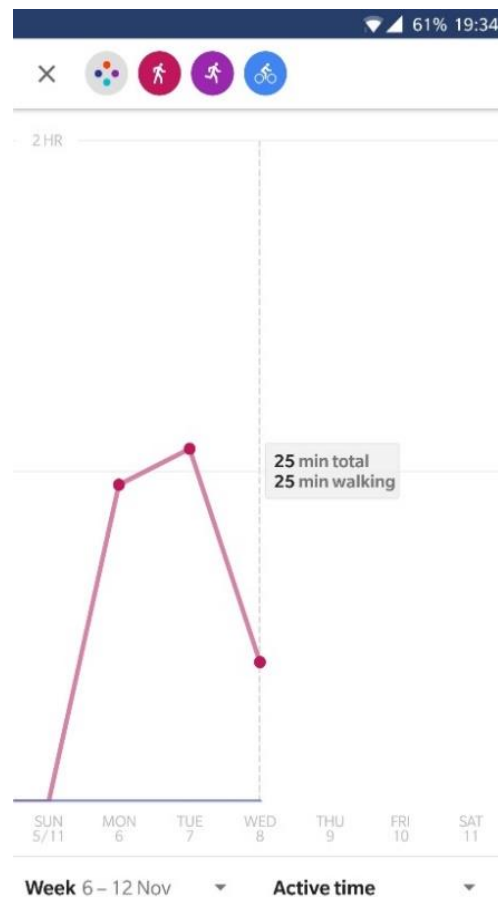


Figure 1 – User Control

This type of indicator would be evident in all three applications and would be a consistent feature in an Android application.

2.2.1.2 Consistency and Standards

Much like the User Control heuristic, Android applications would include consistent features and most applications would mirror one another in functionality and navigation. For example, the home screen of the Google Fit, Google Maps and MapMyWalk applications side by side as seen below in Figure 2:

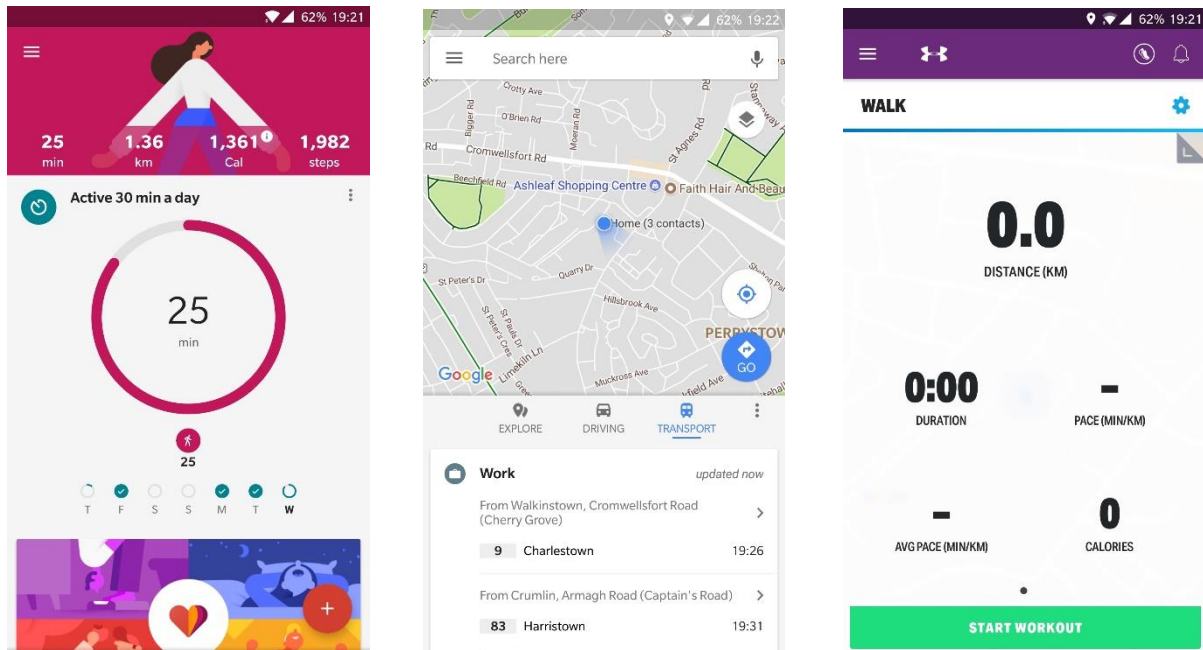


Figure 2 - Navigation Drawer

The main activity of an application would present quick, frequently used functionality. In this case this includes your fitness data for the day, the map with quick access to search and common locations, and the ability to start a new workout.

In the top left is a consistent feature to open the navigation drawer. The three lines would be a familiar icon to Android users and it is inherently understood that this would open the navigation drawer. While not in the scope of Nielsen's Heuristics on the three applications in question, it is an important design consideration to look at other applications and how they implement their navigation. In modern applications it is common to opt for a bottom navigation menu, for example Spotify, YouTube and Netflix, as seen below in Figure 4:

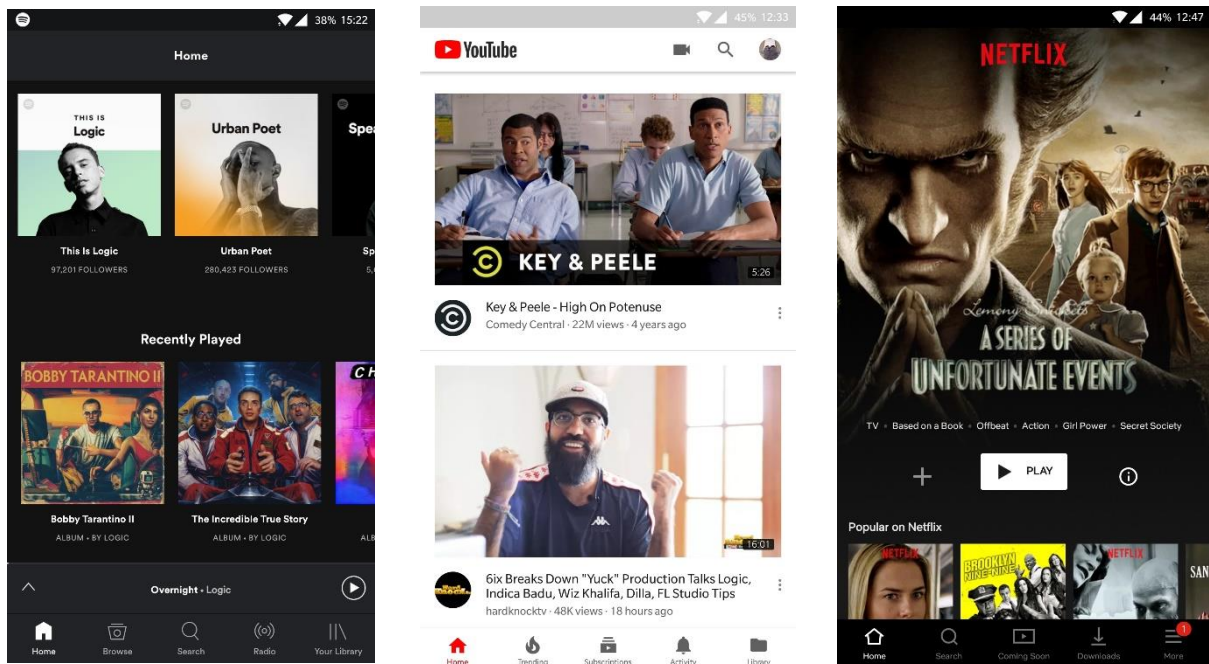


Figure 3 - Bottom Navigation Menu

While other applications such as Google Play, Twitter and IMDb opt to use a TabLayout on the top for navigation as seen below in Figure 4:

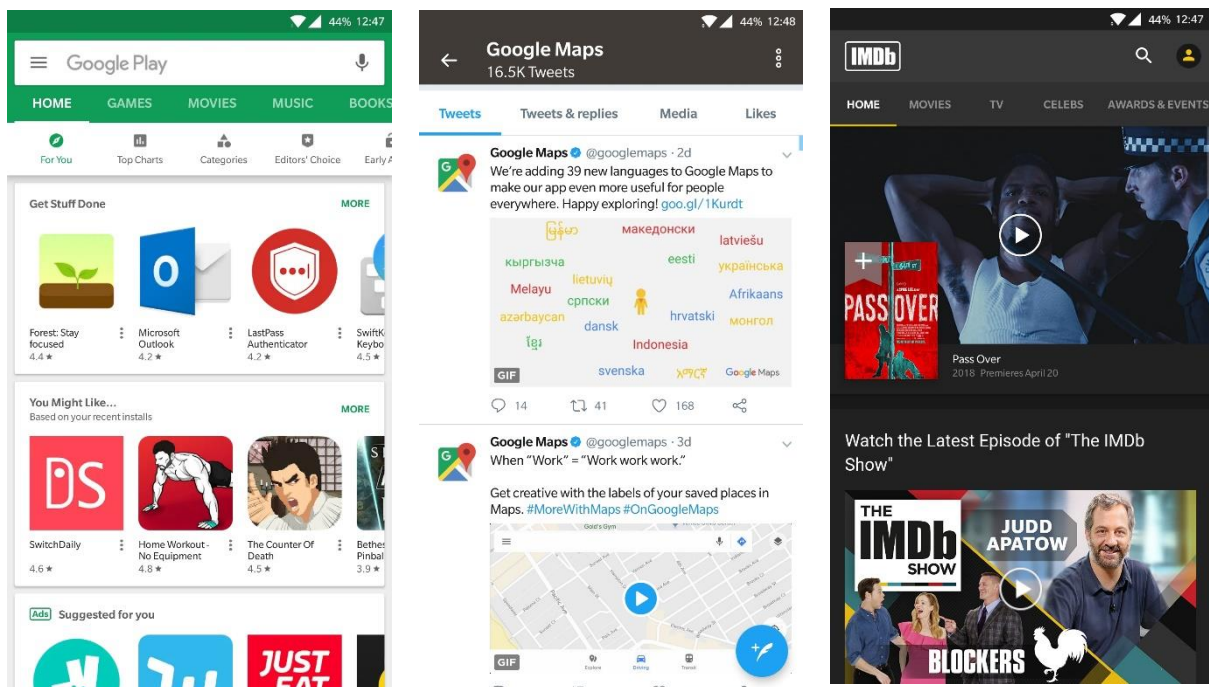


Figure 4 - TabLayout

2.2.1.3 Recognition Rather Than Recall

Fit and MapMyWalk would perform well in this heuristic. These applications would display all relevant information on each page and the user would not need to retain much information if at all. The system would also be familiar to use so it isn't complicated to navigate.

Google Maps however would not perform as well as the other applications in this category. On the surface, Maps is straightforward showing a map and a search bar. However, over the years it feels as features build the design begins to feel cluttered. Some features are hard to remember where they are as they are nested deep within other options.

Despite this however, relevant information is still on display for each screen. The difficulty only arises in recalling where that screen is when returning to the application later.

From experience, many users are unaware that Maps stores a calendar of locations that you have travelled to. The feature allows users to see journeys for every day. This feature even allows the user to view photographs taken at a specific location. Despite how well received this feature can be, and how unknown this feature is, it is never advertised within the application.

2.2.1.4 Flexibility and Efficiency of Use

This heuristic indicates that the system should allow the user to tailor their experience based on their own capabilities with the system. This is the only heuristic determined to be lacking in all three applications, although it is perhaps more of a disadvantage with Android. Android applications do allow for customisation of style such as colour however applications would not allow for the user to change much of, if any, of the navigation or layout.

The only application reviewed that would include some flexibility or efficiency would be Google Maps in that it stores two locations on the main application page, your home and your work. This feature only comes available after these locations were determined, which in general would be determined by an experienced user.

2.2.1.5 Aesthetic and Minimalist Design

The Google Fit application would score the best in this category as it has the lowest quantity of features, therefore its simple and minimal design does not restrict or obstruct the function of the application. For example, the main activity of Google Fit is quite bare showing only relevant information as seen below in Figure 5:

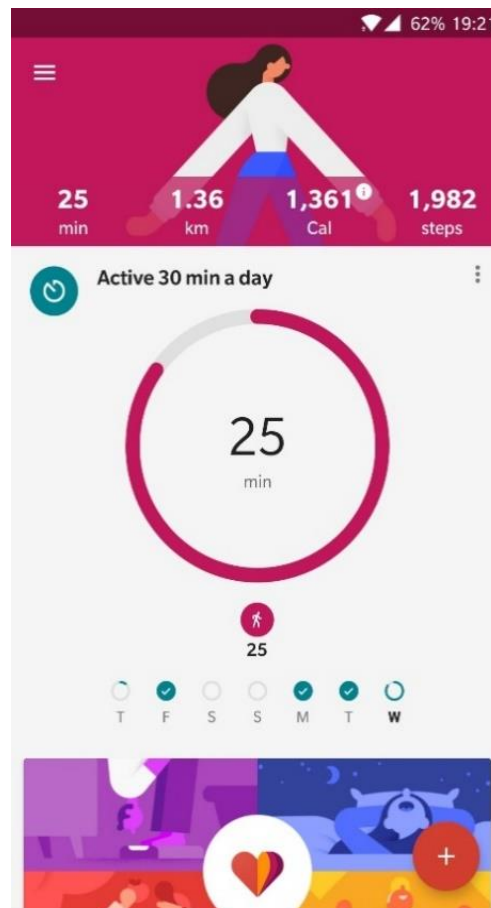


Figure 5 - Google Fit

Google Maps however is rich in features and while it has an aesthetic colour scheme, there are many elements on screen at once causing clutter and the function of the application can be intimidating to inexperienced users. However, all information is still relevant to that activity. Below is an example of a simple route between two locations in Google Maps, showing the clutter and demonstrates the difficult in reading the map quickly:

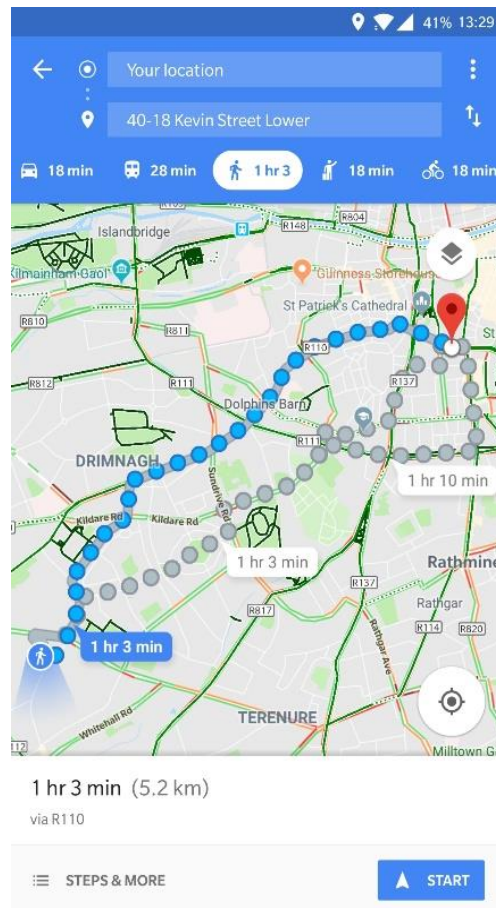


Figure 6 - Google Maps

Overall there are many Android consistencies and standards that a user expects. Following these standards, it is easy to remain consistent with similar applications on the market and reduce the chance of errors and improve design and user interaction with the system. It may be determined, after reviewing the three applications above, what is expected from fitness applications and mapping applications, as well as how to improve on some features. Google Maps could improve several features to reduce clutter and make the experience more intuitive. When implementing the mapping features, I attempted to improve on the areas that Google Maps has scored poorly on.

2.2.2 Google Maps Schedule Update

Since the project's inception there has been an introduction of Google's own attempt of creating a schedule based on common active times, however with the lack of fitness aspects involved in the updates. Existing Google Maps features include the real-time transport information as well as estimate times in journeys to locations. An update to Google Maps saw

an implementation of both features together to track a user's schedule and provide real time information for the journey at that specific time. However, it was proving to be inaccurate. For example, following a lab on Wednesday in November of 2017, between 9 -11, a notification of a bus home was received, demonstrated below in Figure 7:

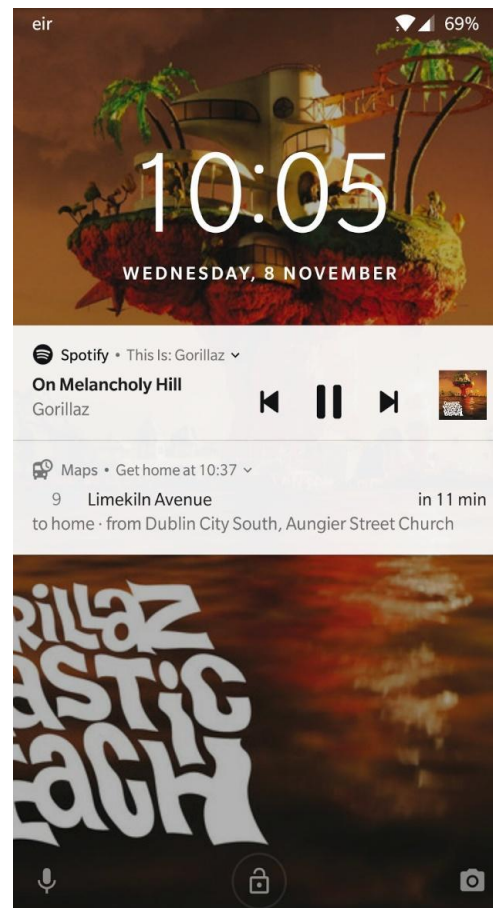


Figure 7 - Google Maps Schedule Notification

This notification presents itself knowing that the user would typically leave at that time and displays the time for the bus and the estimated time of arrival. However, in this example, the user did not leave at 10:05 or 10:15.

By March of 2018 the implementation of this feature had still not improved. The typical time to leave on a Friday was at 7:45 a.m. and the typical arrival time was 8:40 a.m., while Google Maps offers information at 7:05 a.m. and suggests arriving in college by 7:36 a.m. The following is an example notification in Figure 8:

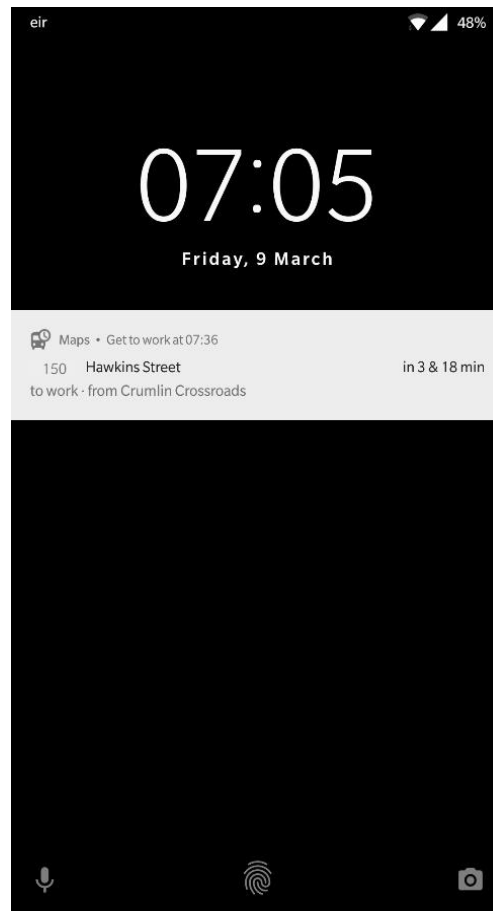


Figure 8 - Google Maps Schedule Notification

2.3 Technologies Researched

In the development of any application it is important to research all available technologies. These technologies include the development type of the application (native, web, hybrid) as well as available APIs and development environments. Before deciding on the software technologies, it is first important to decide on the development type.

2.3.1 Application Development Type

The application development type selected for this project is a native application. The alternatives to this development type include a web application and a hybrid application. Each development type consists of their own strengths and weaknesses, however ultimately in the end the key factor that led to the decision of a native application development is the access to the direct access to platform functionality such as gyroscope sensors and accelerometers used

in measuring fitness activity. This however does not mean a hybrid application, or a web application were instantly excluded.

When considering the best development type, the advantages and the disadvantages of each development type were compared. (3)(4)(5) This process is not as cut and dry as previously hoped however as it is important to then decide if the advantages are beneficial to the project.

The advantages of the three development types are:

Native:

1. Access to native Android APIs.
2. Direct access to platform functionality.
3. Ability to work offline.

Web:

1. The ability to run the application on multiple platforms.
2. The ability to update the application instantly on all platforms through a server, with no reliance on an app store.

Hybrid:

1. Access to native APIs.
2. The ability to run the application on multiple platforms.

After comparing the advantages of each development type, the most beneficial would be between the native and hybrid application due to their access to native APIs. However, the disadvantages are then important to consider:

Native:

1. Restriction to a single platform.
2. Updates must be processed through the Android app store.
3. Typically cost more in time and money.

Web:

1. No access to native APIs.
2. Low access to device APIs (for example, can access the microphone or camera, but cannot access the accelerometer or gyrometer).
3. Performance is network connection dependent.

Hybrid:

1. Abstraction layers can negatively impact performance.
2. Some restriction to device APIs, however more access than web applications.

After considering the disadvantages there are several points to take away from the three development types:

1. Web application development would restrict the project severely.
2. The points on app store access can be considered both an advantage and disadvantage to different sources and in the context of this project is unnecessary.
3. Native APIs and device APIs are most important to this project and native application development can provide the best support for these APIs.
4. The point made on the expense of native application development can also be disregarded as it is an educational task and is not being deployed in the industry.

In summary, after considering all possible advantages and disadvantages and their influence on this project, a native application development would be best suited for this project. This comes down to the advantages listed above as well as personal strengths in native Android application development in comparison to the alternatives that would require new technologies and languages such as HTML, Javascript and CSS. Following the decision to develop a native Android application, using Android Studio as the development environment was an obvious choice, due to the lack of alternatives after Google ended support for Android development in Eclipse. (6)

2.3.2 APIs

Following the decision to develop a native Android application, there is a wide variety of native APIs to select from. The two necessary APIs required for the fundamental features of this project are in fitness recording and mapping.

2.3.2.1 *Fitness Recording*

Fitness recording APIs appear limited on Android with popular applications all choosing to use the Google Fit API. The wide range of partners developing applications using the Google Fit API are visible on the Google Fit website. The Google Fit API was therefore the choice for this project and there was little to no competition in this category. The powerful platform provided by Google ultimately allows the recording of user's fitness data and accomplish the goal of this project. It consists of four essential components:

1. The fitness store: This is the cloud-based repository of data that the device has collected.
2. The sensor framework: A framework of functionalities that enable the developer to use the device to record and collect fitness data.
3. Permissions and user controls: Permissions provided by the API to allow the developer to request users for necessary permissions to record specific data.
4. Google Fit APIs: The API to allow access to the fitness store. This allows for the developer to record data and commit it to the repository as well as access it later for analysis or representation.

Figure 9 below is a diagram provided by Google Fit ([7](#)) on the following components:

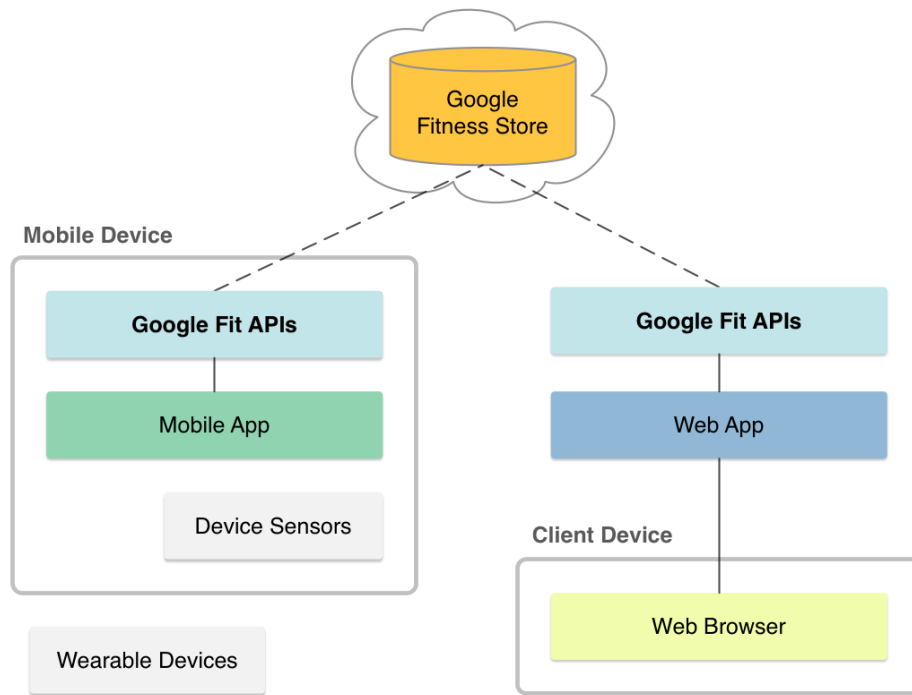


Figure 9 - Google Fit Components

This diagram shows how a developer may also use a RESTful API to communicate with the Fitness Store, however as previously mentioned, sensor access is limited in web applications and therefore the native Android API is better suited for usage with a native application.

2.3.2.2 Mapping

Unlike the selection of fitness recording APIs, there are many options to choose from in mapping APIs. (8) For this project, the focus was on the Google Maps API and the open source alternative OpenStreetMap. (9)(10)

Each application has its own set of advantages and disadvantages. The obvious advantage of OpenStreetMap is its open source nature. In updating a map OpenStreetMap relies on its community to volunteer and provide updates. This is a huge advantage but also comes with a cost. The advantage of a community-driven, frequently updated, accurate mapping service is that it would ensure a quality and accuracy a developer would be unable to deliver and thus relieves the risk of inaccuracy on the map for the developer. However, this also creates a risk for developers as some people would accidentally or maliciously alter a map incorrectly. This risk is somewhat limited however due to the responsiveness of the OpenStreetMap community.

Another advantage of OpenStreetMap is its ability to customise the map. Developers can change and control the map based on their requirements more than if they were to select Google Maps.

This is not to say however that OpenStreetMap is the immediate choice for a mapping API. The Google Maps API has advantages over OpenStreetMap as well. The API includes native directions and a “Place” API to allow provide nearby locations, while OpenStreetMap would require a third-party service to provide these features.

Another advantage of using Google Maps is its native Android API and, in this project, specifically, using two Google APIs to improve consistency across APIs. Unlike the Google Maps API, OpenStreetMap would require third-party JARs to be added to the build to communicate with OpenStreetMap, for example osmdroid. [\(11\)](#)

Finally, the Google Maps API was selected despite its disadvantages in customisation. The advantages of its native API and the native features such as directions and places, this API met the needs of this project more than OpenStreetMap could without adding unnecessary complexity with third party software.

2.3.3 Target Android SDK Version

Following the selection of APIs and application development type, the Android API level can be decided. Google Fit and Google Maps are both distributed through the Google Play Services [\(12\)](#)[\(13\)](#), a dependency in Android from Android 2.2. As a result, the minimum Android API version can be 2.2. However as of August 2017, only 9% of Android users are using an Android version lower than 4.4. [\(14\)](#) As a result, to maximise libraries and utilities while still targeting a large market, the minimum API level was selected to be 6.0 (or SDK level 23).

2.3.4 Machine Learning

With the selected development environment in mind, it is now necessary to investigate machine learning. Machine learning is growing in popularity and there are a range of libraries available

for use with Java. (15) However, Machine Learning with Android is typically for image classification. When researching machine learning for Android it is recommended to perform the operation on a remote, powerful server and return the data. (16) This process would be for large datasets and complex models. However, introduces security risks as now personal data such as time and location data would be stored remotely. This data, despite various measures of security, may still be too valuable to some and would be a reason to not use the application. This point is further discussed later.

With small datasets for each week combined with the growing strength of smartphones, the intention is to perform classification on the device itself, reducing the need for remote connections and increasing security. From a breakdown of smartphones this year, it is shown that the average amount of RAM in an Android device is 4GB and the CPU clock speed is 2.35GHz. (17) For processing small datasets this power is satisfactory. Data classification could also be scheduled to avoid using the device's resources while the device may be in use. For example, deploying a technique like the Google Play Store, by using the Android library `BatteryManager` (18) to detect the device is charging, it would be possible to classify the datasets without draining the device's battery or consuming the device's resources while it is in use. Ultimately, the Android Job Scheduler would be best suited for this task to determine battery levels and ensure the device is connected to an unmetered network connection when connecting to the Google Fit Store.

With the intent of applying machine learning on a dataset on the device, the range of available libraries is restricted. Upon research of available libraries, the Java library Weka (19), an open source machine learning software, has been converted to an AAR (Android Archive) by community members. (20) This AAR is a version of Weka that removes the GUI and incompatible Android features but still allows for access to the Weka Java API.

The advantages of this approach to machine learning is:

1. The open source software used.
2. The ability to classify on the device and avoid remote connections which subsequently decreases data privacy concerns.
3. Weka allows for many types of supervised learning such as classification and unsupervised learning such as clustering as well as time series analysis and forecasting.

2.3.5 Data Storage

The final necessary technology to research are the possible data storage solutions. The objective was to create a fast and lightweight local storage to reduce the need for remote data access and maintain speed in the application. This requirement was for two reasons. The first reason to maintain local storage was to reduce the time necessary for data access. The second and the most important reason is data security. Remote data storage is less secure and although the data stored would not contain personally identifiable information, it is important to reduce potential security threats in the event a link was made. A local storage solution successfully reduces this threat.

With the objective to create a local database and the dependency on the Android OS and Java, there were several choices. From a previous project, there was experience in using SQLite. SQLite is a native library for Android and is quite portable. However, there are several restrictions to SQLite, for example the inability to store objects. Upon research of alternatives, Realm DB was frequently suggested.

Realm DB is an open-source local database that is designed for performance and efficiency. The large overhead related to SQLite in reading objects into buffers, writing SQL to store the buffered data and then performing the opposite to read the data, is eliminated in Realm. Realm allows for direct object storage. As a result, Realm is significantly faster than SQLite as seen in Figure 10 ([21](#)) below:

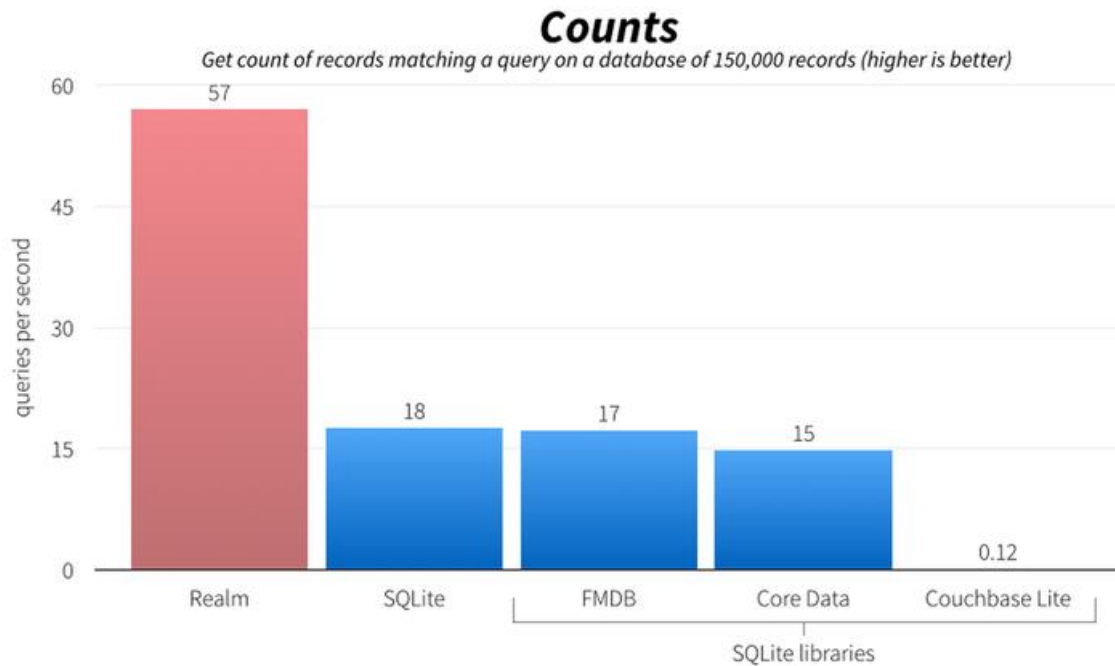


Figure 10 - Realm Speeds Compared to SQLite

2.3.6 Data Visualisation

The final area of technical research for this project was in data visualisation. Data visualisation is not a necessary component for the project's success however is a feature many users would enjoy, and the application could benefit a lot from rich visualisation of fitness data and scheduling data. Data visualisation on Android has several libraries to choose from, for example:

1. DecoView: A library for developing animated circular charts. ([22](#))
2. MPAndroidChart: A feature rich library for creating charts. This library also allows for storage and read access to a Realm database. ([23](#))
3. WilliamChart: Another chart library however not as feature rich as MPAndroidChart. ([24](#))
4. Google Charts: An HTML library with many data visualisation techniques. This however is not native to Android and requires an Internet connection and all graphs would be loaded in an Android WebViewer. ([25](#))

2.4 Other Relevant Research

2.4.1 Terms and Conditions

While other research such as market interest or technical requirements are important, it is also important to understand privacy concerns. The fundamental features of this application include fitness as well as location data. It also important to consider terms and conditions of APIs.

Google Fit includes an explicit statement on their developer's page that indicates how a developer would responsibly collect and manage user data. (26) This states that the users should always know what data you wish to collect and why as well as providing the option to delete fitness data upon user requests. Among these guidelines are rules for using the Google Fit API, for example Google Fit may not be used for "non-fitness" purposes such as selling user fitness data.

2.4.2 Data Privacy

In a discussion with Deborah Lupton (27), author of "The Quantified Self", she indicates how users are aware of data violations such as the Ashley Madison hack yet still do not understand that they have personal data at risk as well. Most people appear to be unaware of the dangers their smartphone is susceptible to or the various platforms their data is stored on. The main privacy concern with this project is that fitness data is stored in the Google Fitness Store – a requirement in the terms and conditions of Google Fit. As a developer, not having control of where the data is stored makes it challenging to provide data security, however choosing a reliable service such as Google does provide some comfort.

Despite the privacy concerns associated with fitness data, the greatest concern lies with location privacy. Under section 9.3 A, Geolocation privacy, of the Google Maps API terms & conditions, it is stated again that you must clearly state what type of data you intend to collect from the user. (28) Android 6+ requires an explicit permission check for "dangerous" permissions such as location that the user must first accept so that they understand location is tracked and the API terms and conditions are met. The terms & conditions continue to state

that the application must never identify an individual user, a condition most would agree upon to ensure location privacy.

In July 2017 however, Snapchat released an update to include Snapchat Map. This update allowed for Snapchat friends to see the exact location of a user on a map. The backlash received indicates many examples of why location privacy is important. Although the update did not break any terms and conditions or laws, guidelines still suggest location privacy is of major concern.

The main criticism of this update was the ability to identify the location of children, a large demographic for this application. (29) This is a major security concern and despite the feature requesting permission from the user and being off by default, it is not safe to assume that this demographic would understand the risks involved.

For the reasons above such as personal data hacks like Ashley Madison or Snapchat Map's breaking of location privacy guidelines, among other violations of data protection, the Data Protection Acts of 1988 and 2003 will be replaced. The General Data Protection Regulation (GDPR) will be enforced in the DPA's position (30), with a wider scope for data privacy and stricter punishments for offenders of the rules stated by the GDPR.

The GDPR states that there must be a reasonable level of protection of personal data, however leaves the definition of reasonable ambiguous. This will allow for developers to adjust the level of protection for the level of privacy associated with data and the volume of this data collected. It will also allow for the GDPR to assess each infringement of the rules differently again depending on the type of data and the amount.

2.4.3 User Feelings Consideration

During the duration of the interim report, Google launched an update to their Maps application on iOS on the 16th of October as well as a feature on the Android application by the end of October. These features further validate an interest in fitness as well as scheduling, as software giants such as Google begin to include features hoped to be implemented in the final project.

The iOS update on the 16th of October added the feature that would track calories based on walking routes entered on Google Maps. This update included a feature that would translate calories into the portion of a mini cupcake. While a harmless feature, it proved to cause some controversy. There was a considerable amount of negative user feedback that said it would encourage unhealthy eating habits and a need for constant calorie counting promotes a bad mentality with food. Figure 11 demonstrates this feature. ([31](#))

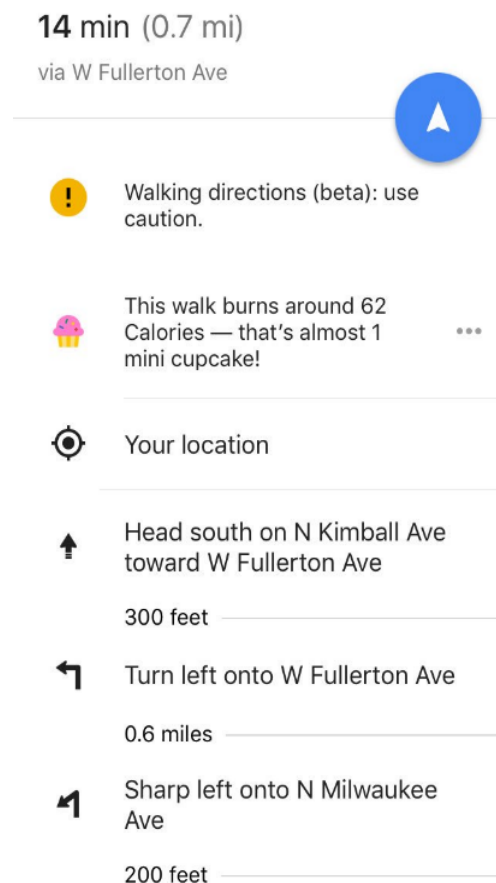


Figure 11 - Google Maps Calorie Update

The feature was then removed within 24 hours. This is an important lesson to learn from this update from Google. Not all users would be interested in seeing this feature, and those who could acknowledge that some people would enjoy the feature, they hoped it would be optional however Google had no option to disable the feature. ([32](#)) This teaches developers that it is important to label applications as fitness-oriented, so users know what to expect and would not be turned off from fitness information. Google however has always marketed Google Maps as a navigation service, not a fitness service. This update also teaches developers to always promote choice within applications, so users can see as many or as few features as they wish.

2.5 Findings

Following in-depth research and analysis, it is apparent what is required for the project to succeed. The idea of the project not being viable in the market was settled by the overwhelming increased interest into fitness and health applications. This was also reinforced by Google's additions to Google Maps, bridging fitness with mapping and increasing scheduling functionality. Despite the interest into these fields however, there is no competition for an application exactly like this project, although there are many applications that will implement features implemented within it.

As discussed, the three applications Google Fit, Google Maps and MapMyWalk all implement certain features that could be combined into a more powerful application. There is a lot to learn from these applications and there are insights on how to provide certain data and how to avoid pitfalls that these applications fell into or had to overcome.

The main obstacle for all three applications to overcome was data privacy. It is important to follow all terms & conditions of the APIs utilised in this project and display intentions clearly to the user, while avoiding data security threats such as identifying an individual or data leaks. As a result, only necessary data would be stored for a user to avoid leaking sensitive data and remote data storage will be not possible to reduce the threat of security attacks.

Another key aspect of the project research included the technical requirements. This includes the decision to develop a native Android application and utilise native Android APIs despite the advantages of other development types such as web applications. The native development can provide the best functionality for this project and plays to personal strengths in Java and Android development, as well. Similarly, the decision to utilise a Java based machine learning library will also benefit from personal strengths in Java however more experience is required in machine learning to develop a more efficient and powerful data classifier.

3. Design

3.1 Methodology

Software methodology is defined as “a framework that is used to structure, plan, and control the process of developing an information system”. (33) There are many software methodologies to choose from (34), for example:

3.1.1 Waterfall

Waterfall – A linear method where each phase is completed in sequence and typically not revisited after being completed. The method is easy to comprehend however its fatal flaw is that it very rigid. If the project direction is misguided in an early phase, the impact is typically huge on the stages that follow. (35)

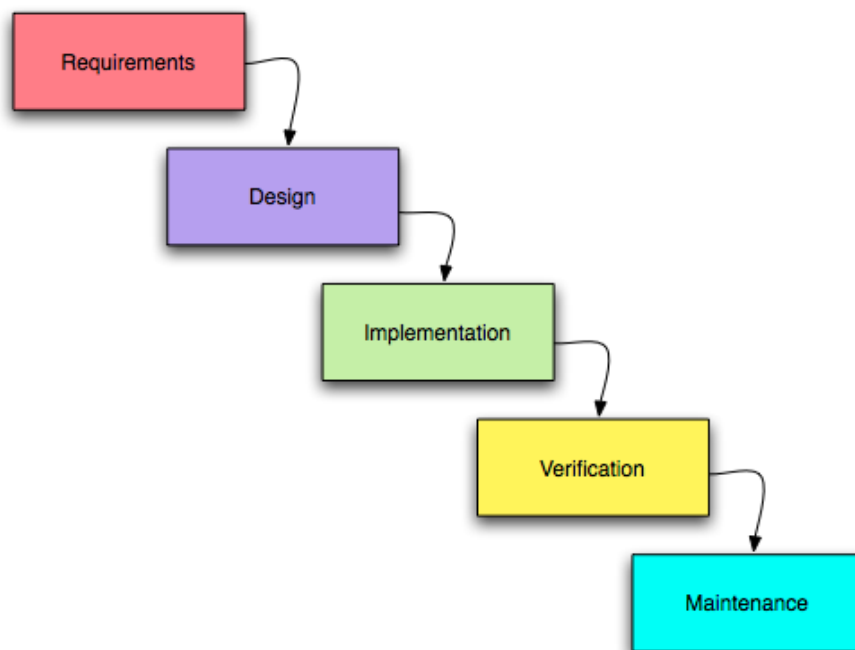


Figure 12 - Waterfall Diagram

3.1.2 Rapid Application Development

Rapid Application Development (RAD) – A method used for developers to adjust their project to requirement changes that consists of four stages and is considered completed after all requirements are catered to. RAD is typically used for projects that are fast paced, time sensitive and consist of developers that have a strong understanding of the project. (34)

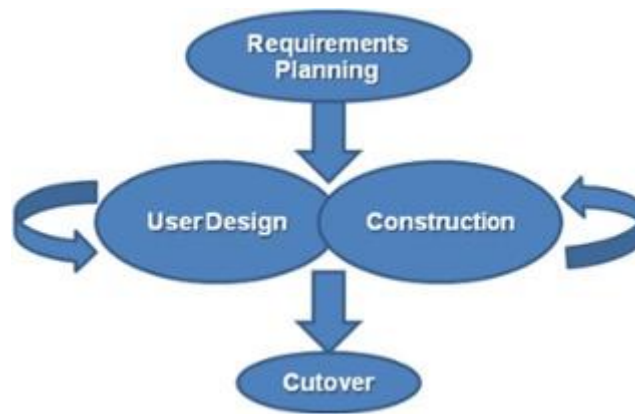


Figure 13 - RAD Diagram

3.1.3 Agile

3.1.3.1 The Agile Methodology

Agile – Agile methodology is a method which is based upon developing software in increments to reduce risk and maintain direction on the project. Unlike waterfall, agile allows for correcting earlier mistakes to reduce the impact a mistake would have. (36) There are many variations of agile, for example Extreme Programming and Scrum.

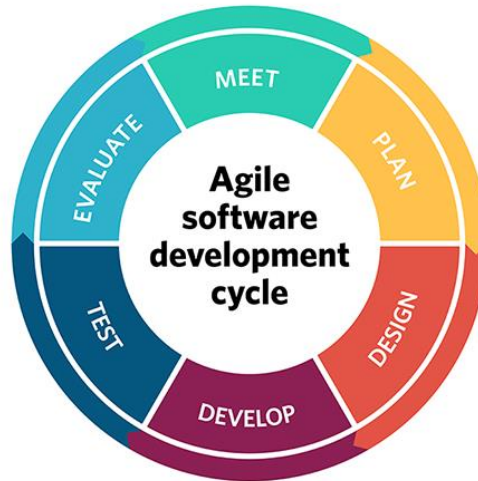


Figure 14 - Agile Diagram

For this project, an agile methodology has been selected. Agile would allow for incremental products and the ability to respond to change while balancing the project and modules in college. The main challenge with selecting an agile methodology however is the dependency of a team however may be manipulated in some way. The agile methodology was conceived from the agile manifesto in 2001. This manifesto states:

"we value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan". (37)

These values are the foundation of agile methodologies and particularly the final value is most important to this project. The unpredictable workload of college combined with the potential risks involved in the project such as a restriction to a necessary API, allows for flexibility in the project. Similarly, the customer collaboration value is important as it allows for design changes over time as features are included. This collaboration would be in the form of the project supervisor, lecturers and friends and family. There are many variations of the agile methodology however the selection was limited to Scrum and Extreme Programming (XP).

3.1.3.2 Scrum

Scrum is a development strategy using iterations and increments to manage a project until such a time the team has reached its final objective. It is typically a team-based methodology. Scrum inherits the agile value of customer collaboration and acknowledges that not all challenges are predictable. Thus, scrum's iterations allow for the development team and customer to react according to challenges that arise. (38)

Scrum has several key roles and components.

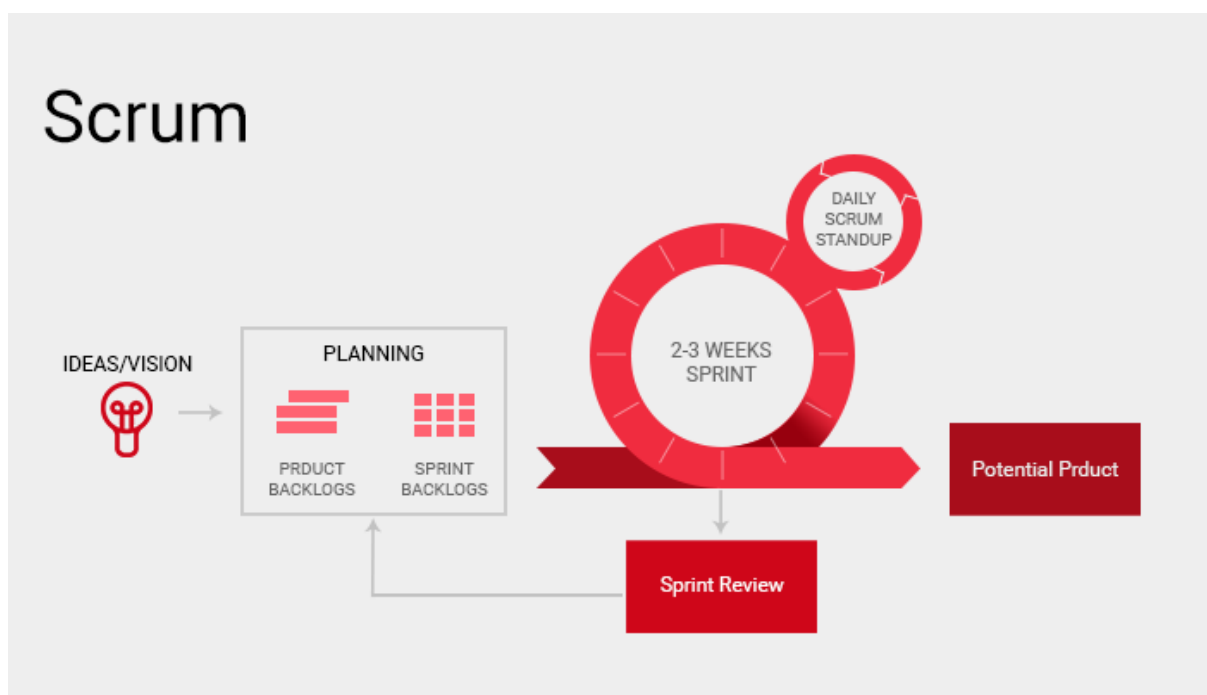


Figure 15 - Scrum Diagram

The key roles are:

1. Product Owner – The role responsible for providing all updates and requirements to the Product Owner.
2. Scrum Master – The Scrum Master is the role responsible for communication between Product Owner and the Development Team. The Scrum Master is also responsible for managing the development team and ensuring the team can reach the defined sprint goals.

3. Development Team – The Development Team must deliver “potentially shippable increments” (PSI) by each sprint.

The key component to the Scrum methodology is the idea of sprints. There is a defined sprint period that gives the development team time to produce a PSI and daily stand up meetings to address daily issues. The main drawback associated with Scrum is that it relies heavily on team development, roles and meetings to operate efficiently. As this is an individual project, the concept of roles and team development is harder to distinguish and maintain efficiency.

3.1.3.3 Extreme Programming

Extreme Programming (XP) is a methodology that has the customer satisfaction at the heart of the development. According to extremeprogramming.org, XP relies on five values ([39](#)):

1. Communication – Communication between developers, managers and customers is important to a team to clarify requirements and transfer knowledge.
2. Simplicity – Maintain an approach that addresses only necessary and immediate requirements.
3. Feedback – Feedback through developers, managers and customers allows for revision of the design and the functionality of the project.
4. Courage – Courage relies on making difficult decisions.
5. Respect – Respecting team members and customers to accept feedback and work together in the best way.

The rules of XP consist of the following ([40](#)):

1. Planning – Plan a new iteration
2. Managing – Maintain a consistent pace and fix the project as necessary
3. Designing – Design the simplest system and interface and refactor as possible.
4. Coding – Consistently develop code to a high, predetermined set of standards with the customer in mind.
5. Testing – All code must have unit tests.

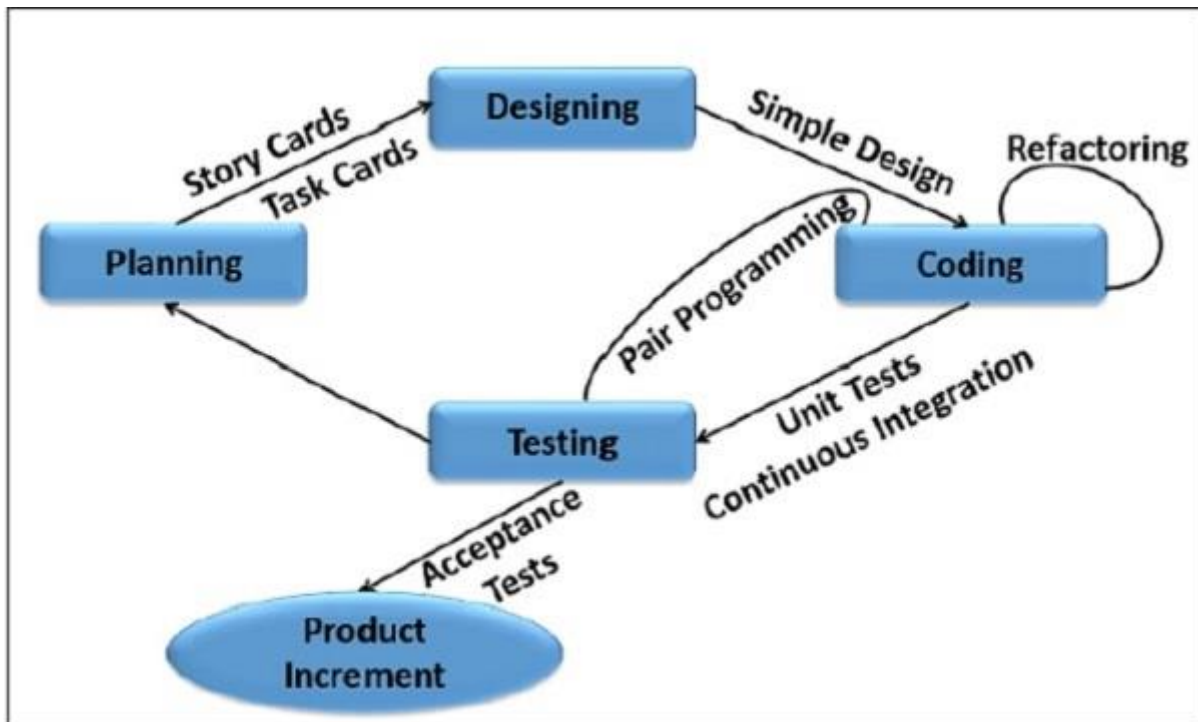


Figure 16 - XP Diagram

Extreme Programming is possible to implement for a solo project however requires adjustment. A key element of Extreme Programming is Pair Programming which is not possible due to the nature of the project. However, communication by the end of an iteration may be completed with the project coordinator.

When deciding between the two agile methodologies, it is important to compare their similarities and differences (41). For example, Scrum iterations would typically last 2 – 4 weeks, while XP would typically have shorter iterations. Shorter iterations expect frequent deliverables however on a smaller scale while longer iterations expect less frequent deliverables but larger deliverables. As the project would be during a college semester, shorter and smaller iterations appear to be more valuable and helps maintain a better sense of direction on the project.

During project iterations, Scrum does not cater to changes, while XP would allow changes provided the feature has not already been developed. If this project was to occur in an industry environment it would perhaps be better suited to stricter requirement changes during iterations, however, like the frequency of iterations, this flexibility within iterations would also be beneficial during the college semester and is important to consider. Another aspect to consider

is team size. Scrum relies heavily on many different roles and larger development teams while XP however, can function better with less members.

Finally, Scrum does not outline any specific testing practices. XP however considers testing as a key value. Following the comparison of methodologies and the subsequent breakdown of agile methodologies, Extreme Programming was selected for the project software methodology.

Due to selecting this approach, code is delivered in stages, roughly every two weeks. In each stage, design and code was delivered and refactoring occurred frequently after iterations to reduce code redundancy. I assigned myself as the development team and the customer as a collaboration of the project goals, the project supervisor and friends and family that helped implement testing and feedback.

3.2 Project Components

It is important to discuss the components of the project that together form the user experience. Most importantly this includes the user interface but will also include the role of each package in the source code and finally the use cases that connect the user interface to the functionality provided in the source code.

3.2.1 UI

The most important feature to the user of an Android application is the interface. The interface is naturally the first thing the user sees and must be simple and intuitive while also displaying all relevant information. The chosen design and colours adhere to Google's Material Design and are reminiscent of system alarm applications, so the design will be familiar to users. The chosen design makes use of a "TabLayout" that was demonstrated in other popular applications such as Twitter. The "TabLayout" combined with a "ViewPager" allows for intuitive swiping between pages. Figure 17 below is an example of the schedule layout:

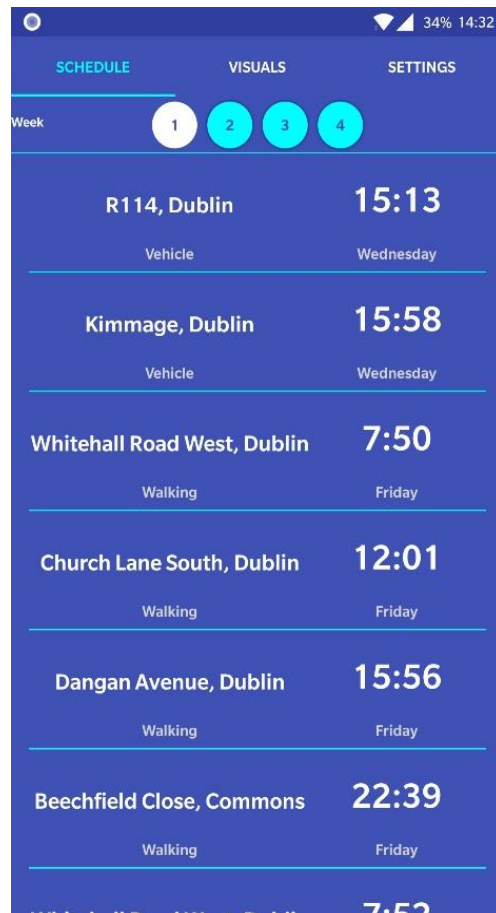


Figure 17 - Schedule Layout

The layout selected works as an intuitive list of times and locations, as well as days and the fitness activity. The list is ordered in order of relevancy, for example, on a Wednesday at 14:32 p.m., the schedule is ordered from all times after that. Similarly, on a Thursday where there are no schedule items, it displays Friday's items first, as demonstrated in Figure 18:

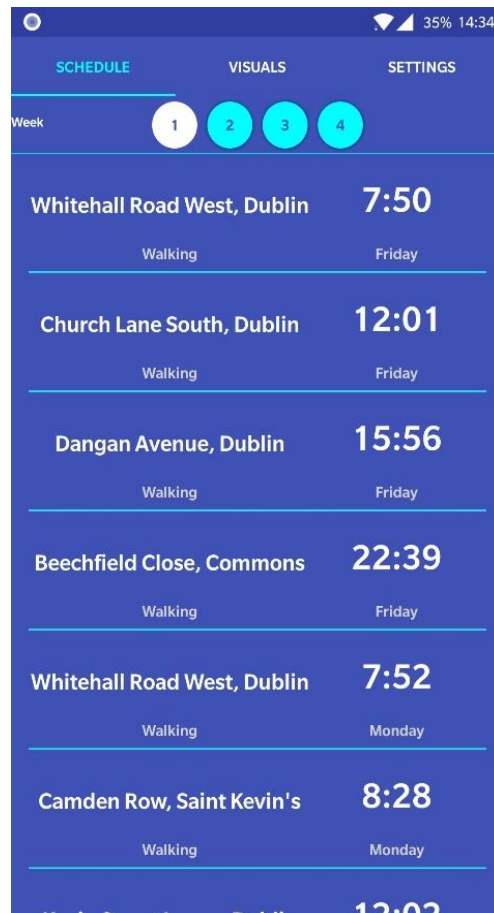


Figure 18 - Ordered List

Selecting the list item will open the map, showing a route between the current location and the destination. The details included are the arrival and destination times, distance and total duration, tailoring the route based on activity. If the activity is walking, cycling or running, it is denoted with a green route, the icon changes accordingly and user speeds are considered in the duration estimation. If it is a transit method, public transport details are displayed in an alternative window. Figure 19 below are screenshots demonstrating both walking and the bus:

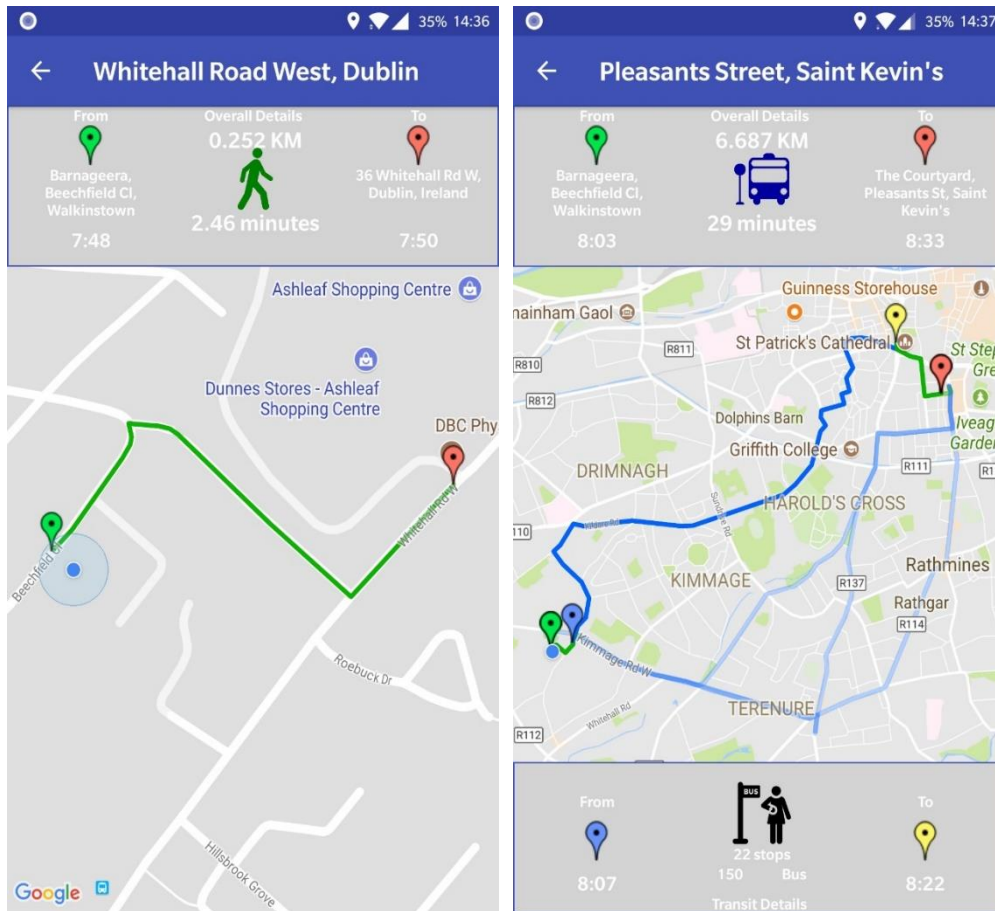


Figure 19 - Basic Map Design

The map design is designed to be basic and display all relevant information yet still manage to reduce all possible clutter in the design. This will not overwhelm the user and in user testing, positive feedback was returned, scoring quite well in terms of usability, especially in contrast to Google Maps.

3.2.2 Source Code Layout

The source code included in the project is separated into individual Java packages based on a common attribute, for example “userdata” or “maps”. Figure 20 below is an example diagram of the source code packages layout from Android Studio:

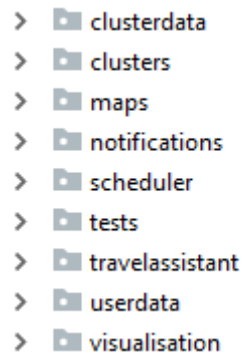


Figure 20 - Source Code Layout

Each package attempts to perform a certain task, with certain exceptions such as “userdata” which contains Objects that store information such as a user’s schedule. These objects are stored in the Realm database. Below is a short description of each package:

1. **“clusterdata”**: Contains all relevant Classes for obtaining and storing data used in the clustering process.
2. **“clusters”**: Contains the Classes responsible for clustering the data and storing the final clusters.
3. **“maps”**: A package containing several Classes for displaying the map and all subsequent functionality such as parsing Google Maps Directions JSON.
4. **“notifications”**: The package used to create, publish and receive notifications.
5. **“scheduler”**: This package is responsible for timing various activities such as the clustering process and notifications.
6. **“tests”**: The package containing all tests in the application such as the Realm database export Class or notification test.
7. **“travelassistant”**: The main package containing the main fragments responsible for displaying the user’s schedule, enabling swiping between views and connecting to Google Play Services for Google Fit data.
8. **“userdata”**: This package contains previous clusters and schedules.
9. **“visualisation”**: The final package responsible for storing all graph related Classes, including the main fragment.

3.2.3 Features and Use Cases

With a schedule in place, the user has the following functionality available to them:

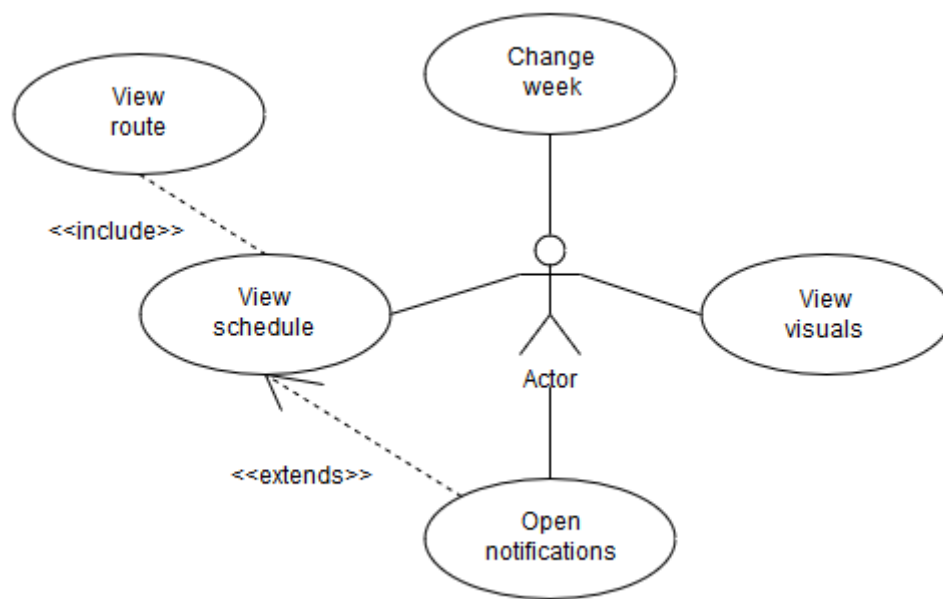


Figure 21 - Use Cases

The user may view their schedule, open notifications and view visuals. When a notification is opened, the schedule is displayed and from there, the user may view the route.

3.2.3.1 View user's schedule

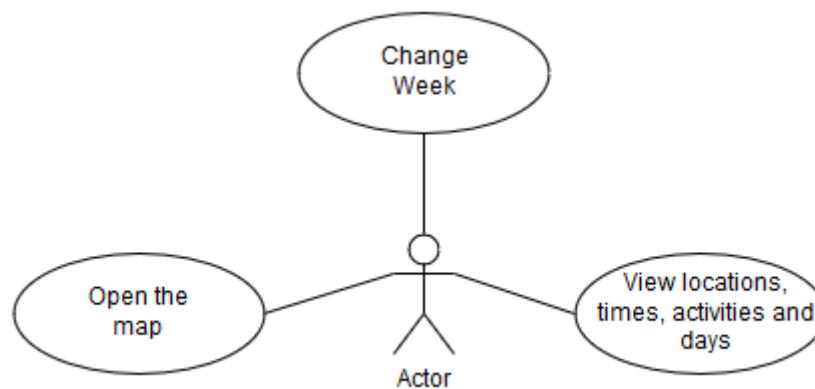


Figure 22 - User's Schedule

When viewing the schedule, the user may select between weeks to display alternative schedules in the past. Each schedule entry will include:

1. The location the user is at that point in time.

2. The time the user is active at that location.
3. The activity the user is typically does at that time, e.g. walking.
4. The day that the previous information occurs on.

3.2.3.2 View route

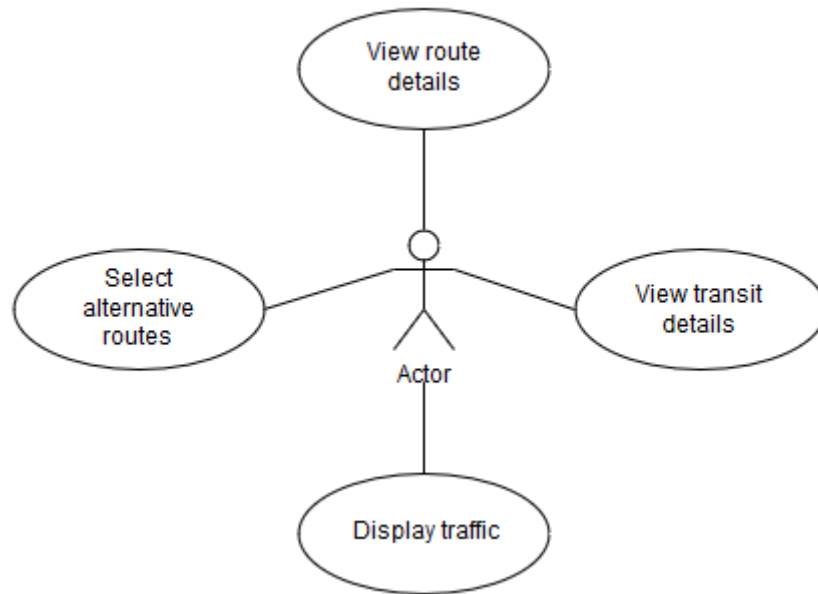


Figure 23 - Route

When a user selects a list item and is currently viewing the route, the following options are available to them:

1. **View route details:** This contains the overall details of the route such as the total duration, distance, departure and destination as well as the overall mode of transport.
2. **View transit details:** If the route contains a transit step such as a bus then an alternative view is displayed showing the departure and destination stop and time, the distance, duration, number or name of vehicle and number of stops.
3. **Display traffic:** The user may choose to hide the route and transit details which will display a floating action button to display a traffic layer over the map. This is an added feature for the user to consider traffic based on each route they may take.
4. **Select alternative route:** Select an alternative route and update the route and transit details for each route accordingly.

3.2.3.3 Display visuals

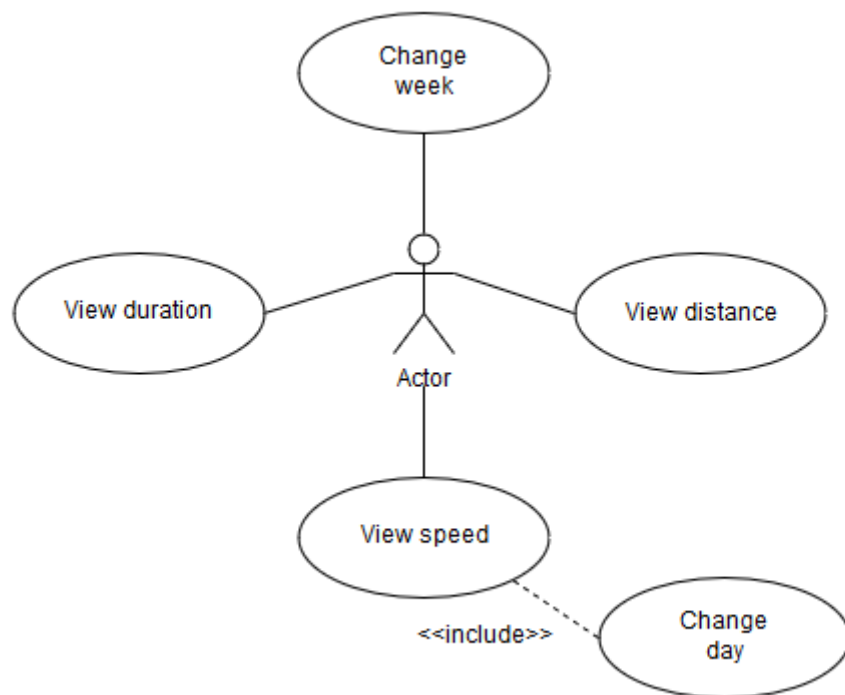


Figure 24 - Visuals

The user may also choose to visualise the data that has been collected. This allows them to view the total duration per day, distance per day and speed per hour. Users may change week to view the changes in their schedule visually, so a fitness-oriented user could, for example, detect slow walking or more vehicle time than walking and adjust accordingly.

1. **View duration:** This will display the total active duration per day for the previous week in a bar chart.
2. **View speed:** This is a line chart to display the speed of the user based on the activity they are doing for each hour of the day. The user is given an option to change days as well as weeks here.
3. **View distance:** This will display the total distance travelled per day for the previous week in a bar chart.

3.3 User Scenarios

Although use cases are a helpful way of explaining the user interaction with the application, it is important to build a narrative behind the use case. This would include describing the type of user that would use the application and how they would interact with it. Therefore, the following short, elaborated user scenarios help answer the following questions:

1. What type of user is the application for?
2. How will the user accomplish their goals with this application?
3. Why would a user select this application over others?

3.3.1 User A Scenario

User A is a young individual in a college degree. This user typically never misses a lecture and is quite strict with their college routine, following a similar schedule each day, however finds it challenging to constantly maintain it. User A however finds that current methods in “realtime” applications are too manual and requires too much user involvement to be able to accurately measure when to leave for a bus to or from college and has never measured the exact time it takes them to travel to their bus stop. User A has received notifications about bus times and times to begin their commute from other applications such as Google Maps however finds them quite inaccurate. User A wants to find some method of accurately recording their schedule to find a solution to the challenge they face each day in accurately maintaining their schedule.

3.3.2 User B Scenario

User B is in the same degree as user A however is less worried about maintaining a schedule and typically arrives late or not at all to a lecture. User B doesn’t aim for any specific bus in the morning, instead user B will get the bus that best suits them at the time they decide to leave. User B has no interest in maintaining a digital schedule and existing applications that provide this functionality are unknown to user B.

3.3.3 Scenario conclusion

From a comparison between the two brief user scenarios for user [A](#) and user [B](#), answers to the previously asked questions may be resolved. Clearly this application would be of no interest to user B however would prove to be exactly what user A is looking for. By using this application, user A could receive notifications tailored to their daily life, including their own personal

speeds and capabilities, unlike current implementations that use averages for every user. This make this application a clear candidate over existing applications due to the accuracy and tailor-made user experience for each user.

4. Architecture & Development

4.1 System Architecture

4.1.1 Diagram

The system architecture of a project outlines an overview of the structure of the system. The system in this project is based upon an Android device as the execution environment with several dependencies to achieve complete functionality. These dependencies include a machine learning library, a local database and third-party APIs. Figure 25 below is an example diagram of the system architecture:

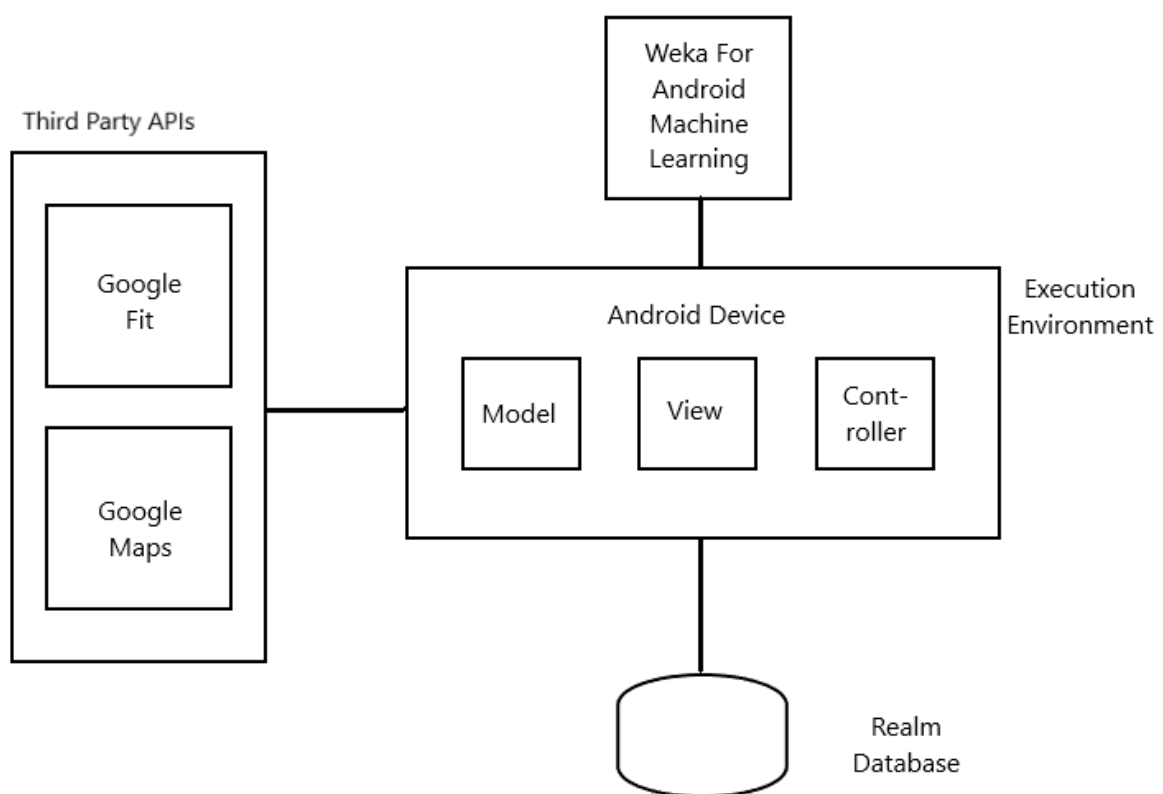


Figure 25 - System Architecture

4.1.2 Execution Environment

From the above diagram, the device is the core of the system. The design of this device is broken into three parts and follows the design pattern of Model View Controller (MVC). MVC is quite intuitive on Android as the model represents the classes and data, the view is the layouts and UI and the controller are the activity that controls the data and the interface.

The execution environment is the Android device the application runs on. This environment consists of the model, view and controller that each communicate with other relevant components where necessary and will also work together to form the complete, working application. The three individual components as previously mentioned are elements of the MVC design pattern. The Model will communicate with the Realm Database for reading and writing. The Controller has a range of functionality to consider, including but not limited to the creation of notifications and the clustering process. Finally, the view would communicate with the various controller classes to display data collected in the model. Views are developed in Android using XML and values are updated and displayed in Controller Classes.

4.1.3 Realm Database

Realm DB is an open-source local database that is designed for performance and efficiency. Realm reduces the overhead associated with alternatives and thus greatly improves the speed. Realm is a NoSQL database that acts as an object store. Objects are created in the application and are stored in Realm and may be retrieved later. This is beneficial for the main functionality of the project in storing schedules that are created, however also has other benefits. These benefits include sharing Objects between Classes with loose coupling, instead of bundling Objects into intents or initialising new Classes and passing the Object as a parameter in a constructor. This approach also allows for the Object to be changed and the changes to persist across the application.

4.1.4 Third Party APIs

4.1.4.1 Weka

Weka is an open source machine learning software that has been converted to an AAR for use on Android devices. [\(19\)](#)[\(20\)](#) Using the Weka Java API, it is possible to cluster on the device.

This library is called to perform a clustering process on the initial launch of the application if Google Fit data is found. From there on, the process is rescheduled to occur after every Saturday night when the device is charging and connected to an unmetered Internet connection. The resulting schedule created is then stored in the Realm database.

4.1.4.2 Google Fit

The Google Fit API is a part of the Google Play Services and is supported in Android 2.3 and above ([42](#)). The API is made of a collection of other APIs, including the History API that is used to view previously collected fitness data. The API also contains the Recording API for recording fitness data such as walking, cycling or in a vehicle.

4.1.4.3 Google Maps

The Google Maps API is used to visualize an interactive map and display relevant information such as directions and transit details, provided by the Google Maps Directions API. ([43](#))

4.2 Project Components

The following section details the Classes within the project that generates the scheduling functionality and provides a usable experience to users of the application. For most, a short description of the functionality of the Class is all that is required however there are Classes that deserve a more detailed discussion where the complexity, challenges and solutions will be outlined. The following list is ordered in a logical order of functionality. Before beginning the discussion of each group however, an overview Class diagram may demonstrate each the communication between each Class.

4.2.1 Clustering

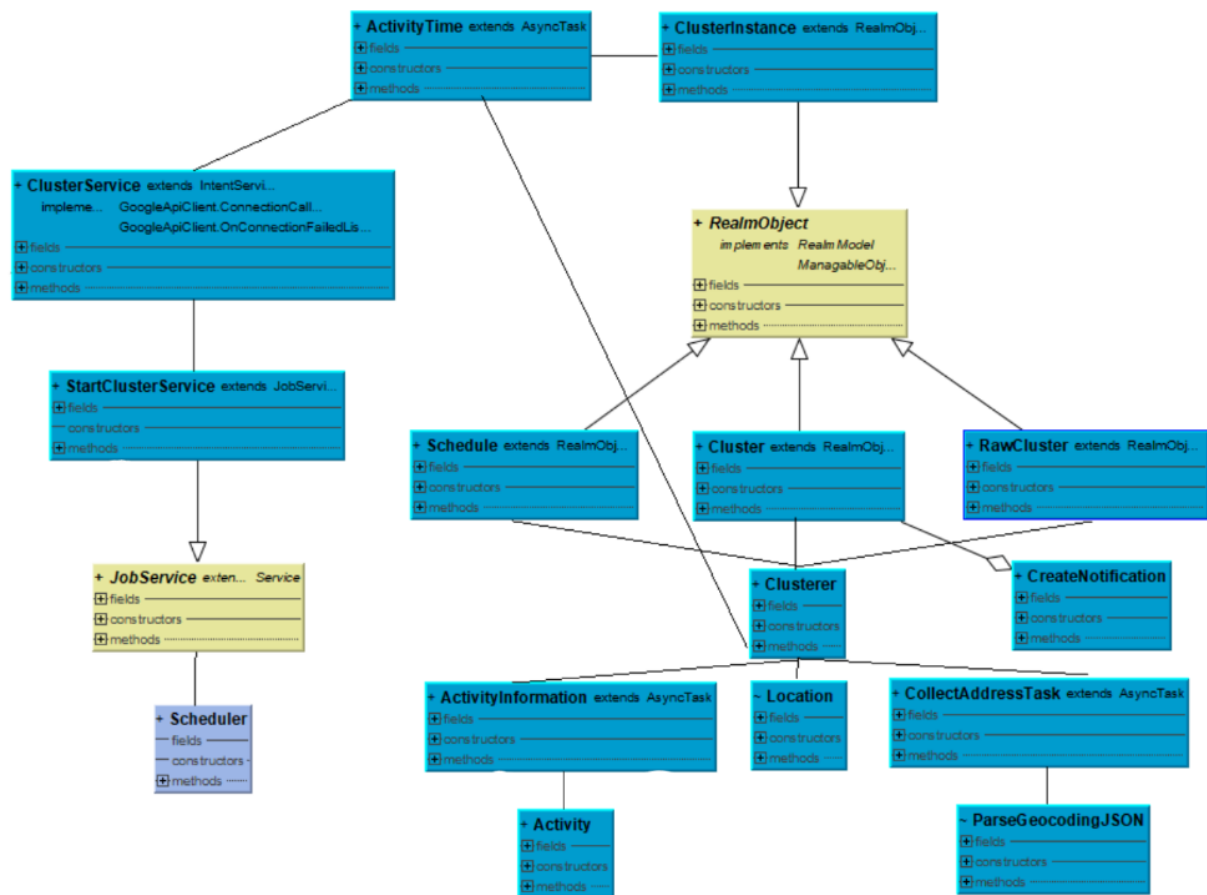


Figure 26 - Clustering Class Diagram

4.2.1.1 Scheduler.java

The Scheduler Class is a utility Class to create a “JobScheduler”, a native Android library, to execute a task when certain conditions are met. In this case, the conditions are that there is an unmetered network connection and the device is charging. These conditions are in place so data retrieval from Google Fit does not consume a user’s mobile data and so the clustering process will not drain the battery or run while the device is in use. This Class is called on the initial launch of the application and is subsequently called at the end of every week at 12 a.m. on Sunday.

4.2.1.2 StartClusterService.java

When the Scheduler's Class's conditions are met the StartClusterService Class is called. This Class extends JobService and thus overrides the following methods:

1. **onStopJob:** A method used to logic after a job completes. This method is unused in this case due to multithreading and post-clustering logic is handled in a BroadcastReceiver.
2. **onStartJob:** The method responsible for handling the beginning of the job itself. This method creates a notification to inform the user of the running process and calls ClusterService. The method ends with a return true. This indicates that the job will be handled manually as when the job executes on a different thread and the current thread ends, the job is considered completed and this would interfere with user communication and post-clustering processes. As a result, the previously mentioned BroadcastReceiver is used. The figure below demonstrates the notification the user receives during the clustering process:

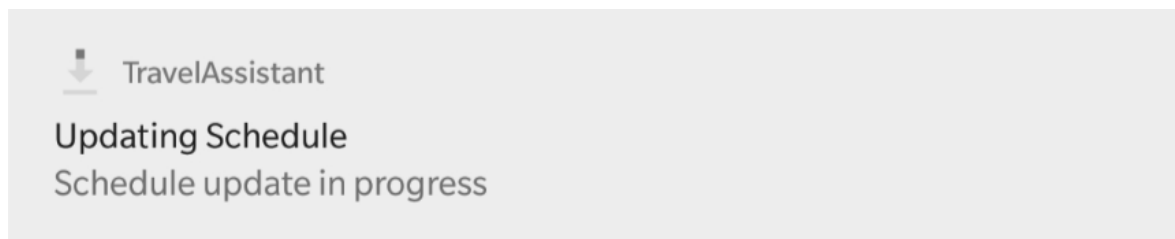


Figure 27 - Schedule Update Notification

The BroadcastReceiver is called on completion of the clustering process and removes the current notification and sends a new one that the process is complete as seen below in Figure 28:

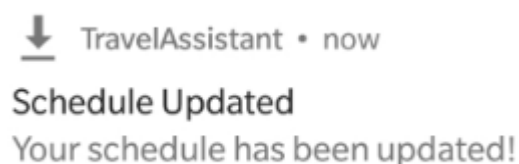


Figure 28 - Schedule Updated Notification

Upon clicking this notification, the user's schedule will open. The receiver will then reschedule the process to occur again at the end of the next week. The rescheduling is done using the AlarmManager Android library that will execute a process at a given time.

4.2.1.3 ClusterService.java

The ClusterService is a Class that will begin the entire clustering process. The service connects to the Google Play Services first to retrieve Google Fit data. It is assumed that the user will login without error to the Google Play Services as they previously must sign in when they first open the application and the job only executes if connected to the Internet, however, connection callbacks are in place to handle errors in connecting. Following the connection, an asynchronous task is executed to retrieve all active times from Google Fit in the past two weeks. It is at this point the previous onStopJob method in StartClusterService would be called as this thread has completed its task.

4.2.1.4 ActivityTime.java

As mentioned, the ActivityTime Class is an asynchronous task. The process of making a request in Android should be performed as an asynchronous task. Asynchronous tasks in Android execute their main logic in the doInBackground method. This method initialises Realm and deletes all ClusterInstance Objects from Realm before proceeding. Following this, the process creates two loops, one for each day and another for each week. For that given day in that week, a Calendar Object is created. First the time is set to 12am, recorded as the starting time in milliseconds, and then set to 12pm and is again recorded as the end time in milliseconds. A DataReadRequest is created to request all activity data for that given day that is longer than two and a half minutes. This time frame was selected after several tests to eliminate small, short movements that are not noteworthy in the grand scheme of the schedule.

The DataReadRequest is then executed as a PendingResult and stored in a DataReadResult. If the result is successful, the data is iterated. If the activity type is in a vehicle, cycling, walking or running, then a new asynchronous task is executed. This task will collect location data. Following all data processing, the task completes and calls Clusterer.java to execute the clustering process.

4.2.1.5 Location.java

Much like ActivityTime, Location is an asynchronous task to collect the location in latitude and longitude of the activity found by ActivityTime. The process is like ActivityTime however the times are passed from ActivityTime as the start and end time of the activity itself. A ClusterInstance Object is created for each location found, which contains the day, the time in milliseconds based on the Unix Epoch, the latitude and longitude.

4.2.1.6 ClusterInstance.java

A simple data object stored in Realm that is created in Location.java and holds the day, time, latitude and longitude.

4.2.1.7 Clusterer.java

The most complex Class within the entire process. This is the heart of the scheduling process as it collects previously created ClusterInstances and attempts to establish a relationship between them to form a user's schedule. The Class is responsible for all machine learning and calls multiple other asynchronous Classes to collect more data on the clusters that are created in the clustering process.

When the Class is initialised, a loop begins for each day. For each day, it will collect all ClusterInstances from Realm. If there is a sufficient number of instances, currently set to 25 after several tests to determine a reasonable number, then the clustering process occurs for this day.

The cluster process first declares ArrayLists of Weka Attributes and Weka Instances. The current number of attributes is set to three and the number of instances is the number of ClusterInstances for the current day in the process. The three attributes are time, latitude and longitude. For every instance, a new SparseInstance is defined and time, latitude and longitude are stored in the instance. Now a dataset is created using the attributes and instances.

The first challenge is faced here although was only solved much later in development. When clustering the data using EM or Simple K-Means, Euclidean Distance may be used to create relationships between items based on their minimum distance. When comparing one item with another in this application, the scale between each attribute is too different. For example, time is in milliseconds and may be up to eight digits in length and the difference between two times may be in the millions. However, latitude and longitude are in double digits with several decimal points and a change in either could simply be less than .1. Normalising the data allows for all comparisons to be performed on the same scale and thus all data is converted to single digits to improve comparison between attributes.

With the dataset normalised, there is an attempt to remove outliers. Outliers in this dataset can be something simple such as an impromptu walk at a time where the user typically is inactive. To remove outliers, the interquartile range filter is applied. The interquartile range is a measure of variability based on dividing a data set into quartiles. (44) The data set is divided into four parts of equal size. The first quartile is the middle value in the first half of the data set and the third quartile is the middle value in the second half of the data set. The interquartile range is the third quartile minus the first quartile. This range plus or minus the median value is all acceptable points in the dataset. Anything outside of this range is deemed an outlier and is thus removed.

The model is then created after normalising the data and removing outliers. The model selected is EM, an iterative algorithm that does not require a set number of clusters to create. Instead, the algorithm iterates over the dataset multiple times and attempts to determine the number of clusters through cross validation. The results are then created when the model is built. By logging the results to the debugger within Android Studio it is possible to view the data in a raw form. The following is an example of a cluster set before the addition of normalization and the removal of outliers. The time is in milliseconds and the day is Monday.

```

Day: 2
EM
==
Number of clusters selected by cross validation: 4
Number of iterations performed: 7

```

Attribute	Cluster 0 (0.44)	1 (0.14)	2 (0.19)	3 (0.24)
time				
mean	56705161.3463	28677230.0443	30635977.0252	61800323.6626
std. dev.	6775634.0367	473466.2335	635729.1055	4300296.7417
latitude				
mean	53.3364	53.3152	53.331	53.3167
std. dev.	0.0031	0.002	0.0046	0.0028
longitude				
mean	-6.2673	-6.3096	-6.2725	-6.3063
std. dev.	0.0036	0.0103	0.0054	0.0144

```

Clustered Instances
0      45 ( 44%)
1      14 ( 14%)
2      19 ( 19%)
3      24 ( 24%)

```

Figure 29 - Clustering Results

The above clusters in Figure 29 are not sorted by time. Investigating cluster 1 and converting the time to hours and minutes, the 28677230.0443 translates to 7:57 a.m. However, after normalising data and removing outliers, the numbers of clusters and the accuracy of them improve. For example, in Figure 30:

```

Day: 2
EM
==
Number of clusters selected by cross validation: 11
Number of iterations performed: 11

```

Attribute	Cluster 0 (0.12)	1 (0.17)	2 (0.06)	3 (0.02)	4 (0.07)	5 (0.21)	6 (0.09)	7 (0.03)	8 (0.04)	9 (0.13)	10 (0.06)
time											
mean	64622200	60850846.9521	62264107.0893	29037166.6667	28206588.2353	52441312.9603	54157857.1429	39482371.5067	63202000	31213257.9187	28462200
std. dev.	632134.7641	3025203.5296	215173.6538	115996.2883	63027.5932	3477208.3701	284298.0136	435134.5395	2696534.2403	497416.2704	107800.5566
latitude											
mean	53.3151	53.3362	53.3428	53.313	53.3149	53.3372	53.3157	53.3376	53.3164	53.3354	53.3162
std. dev.	0.0002	0.0025	0.0009	0.0004	0.0001	0.0003	0.0005	0.0001	0.0021	0.0019	0.0002
longitude											
mean	-6.3178	-6.2664	-6.2601	-6.3036	-6.3187	-6.2678	-6.318	-6.2675	-6.2941	-6.2674	-6.3164
std. dev.	0.0023	0.003	0.0011	0.0044	0.0006	0.0004	0.0016	0.0003	0.0027	0.002	0.0003

```

Clustered Instances
0      30 ( 12%)
1      42 ( 17%)
2      14 (  6%)
3       6 (  2%)
4      17 (  7%)
5      52 ( 21%)
6      21 (  9%)
7       7 (  3%)
8      11 (  4%)
9      31 ( 13%)
10     15 (  6%)

```

Figure 30 - Improved Clustering Results

Most notably in this result for Monday is cluster 4 and 10. The time for each translates to 7:50 a.m. and 7:54 a.m. which are much more accurate times in the testing experience. When testing, the morning walk occurred at 7:50 a.m. and the bus arrived at 7:54 a.m. These results are much more accurate in comparison to the previous results.

These results are clearly not suitable in that form and thus must be translated and stored as an Object. When iterating over the dataset again, it is another opportunity to catch abnormal data and thus any cluster that happens to have another cluster too close in time is then removed from the list before being converted back to their original value to remove the normalisation changes and stored as a Cluster Object. This entire process then loops for each individual day until all days are complete.

When each day has been evaluated and potentially clustered, the clusters are then compared to previous schedules. This process attempts to improve the accuracy of the clusters in the schedule by first looking for previous schedules and then comparing against them. As an example, assume one week, a user has a schedule where they are active at 9 a.m., 11 a.m., 1 p.m. and 3 p.m. The following week the user is only active at 1 p.m. and 3 p.m. The schedule will see that the previous week those times, plus or minus the standard deviation, were there and store them in the current week's schedule. If another week went by and the user was only active at 9 a.m. and 11 a.m., the schedule would look at the previous week and discover the user was not active at that time the week before, however, will also look at the week before that and see that the user was active then. This improves the accuracy of the scheduling process over time by providing more data to the application to process and compare to. For each comparison, the time is compared and then the location, each within the range of the standard deviation. The location is compared so the schedule does not record activity at the same time in different areas. If there are no matching clusters, then all clusters from the current week's schedule are taken so the user still has a schedule.

At this point in the process the list of clusters has been refined and reduced. It is now important to determine the activity as well as the typical duration, distance and speed of that activity at the time of each cluster. This process is another asynchronous task in ActivityInformation. Any cluster that does not have an activity for that time is then disregarded, creating the final list of clusters. If successful however, ActivityInformation returns an Activity Object that populates the cluster.

For every cluster in the final list of clusters, the address must be determined. This process is completed using the Google Maps Geocoding API. The API accepts a URL with certain parameters and `CollectAddressTask` is then called which subsequently calls `ParseGeocodingJSON` to return the address to the cluster.

Finally, a schedule may be created out of the final list of clusters that now contain the day, time, activity, distance, duration, speed and address. A new `Schedule` object is created and stored in `Realm`, using the week of the year as an identifier. For every entry in the schedule, a notification is created in `CreateNotification` that will trigger at that time on a certain day for the user with all relevant details and allow the user to open it and view a map containing directions and times to maintain their current predicted schedule.

The final step of the clustering process is to then return a message to the `BroadcastReceiver` in `StartClusterService` that the process has been completed.

4.2.1.8 Cluster.java

A `RealmObject` Class that stores important information collected in the clustering process such as the day, time, location, durations, distances and speeds. Perhaps the most important `RealmObject` Class in the application. While discussing `RealmObject` Classes it is important to mention the process of creating a `RealmObject`. The Class must have an empty constructor as seen below in Figure 31:

```
// Default Constructor necessary for Realm, instantiate the Object variables with setters
public Cluster()
{
}
}
```

Figure 31 - Empty Constructor

And when creating a new `RealmObject`, for example a `Cluster`, the method `createObject()` is used and must be a part of a `Realm` transaction, demonstrated in Figure 32:

```

realm.beginTransaction();
Cluster cluster = realm.createObject(Cluster.class);
cluster.setDay(day);
cluster.setMinTime(time - timeDev);
cluster.setTime(time);
cluster.setMaxTime(time + timeDev);
cluster.setMinLatitude(latitude - latitudeDev);
cluster.setLatitude(latitude);
cluster.setMaxLatitude(latitude + latitudeDev);
cluster.setMinLongitude(longitude - longitudeDev);
cluster.setLongitude(longitude);
cluster.setMaxLongitude(longitude + longitudeDev);
realm.commitTransaction();

```

Figure 32 - Setting Values

4.2.1.9 RawCluster.java

Like Cluster, a RawCluster is a RealmObject that stores a list of Clusters and uses the week of the year as an identifier. This Class is used in comparisons between old and new schedules.

4.2.1.10 ActivityInformation.java

ActivityInformation is yet another asynchronous task that queries Google Fit to ascertain the total duration and distance of each activity at the given cluster's time, plus or minus ten minutes to catch more data. The activity with the highest duration is considered the primary activity in the cluster and is assigned accordingly, while the rest are still recorded for visualising purposes later.

4.2.1.11 Activity.java

A simple Object that is returned from ActivityInformation containing all relevant information on each activity type.

4.2.1.12 CollectAddressTask.java

CollectAddressTask is the final asynchronous task in the entire clustering process. This task's purpose is to determine the address based on the latitude and longitude of the given cluster. The process involves using the Google Maps Geocoding API which accepts a URL and returns

JSON. Before discussing the format of the URL and the results, the operation of the Class includes creating a `HttpURLConnection` and connecting to the URL and reading the results into a buffer, before finally reading the buffer into a `String` and creating a `JSONObject` out of the string.

The Geocoding URL may have the following format:

[https://maps.googleapis.com/maps/api/geocode/json?latlng=53.337620,-
6.267877&key=APIKEY](https://maps.googleapis.com/maps/api/geocode/json?latlng=53.337620,-6.267877&key=APIKEY)

The URL accepts a latitude, longitude and API key. The latitude and longitude in the previous URL is for DIT Kevin Street (53.337620, -6.267877), however would typically contain the latitude and longitude of the cluster which is defined in the `Clusterer` Class before calling `CollectAddressTask`. The following is an example of the results in Figure 33:

```

▼ results:
  ▼ 0:
    ▼ address_components:
      ▼ 0:
        long_name:      "18"
        short_name:     "18"
        ▼ types:
          0:            "street_number"
      ▼ 1:
        long_name:      "Kevin Street Lower"
        short_name:     "R110"
        ▼ types:
          0:            "route"
      ▼ 2:
        long_name:      "Dublin"
        short_name:     "Dublin"
        ▼ types:
          0:            "locality"
          1:            "political"
      ▼ 3:
        long_name:      "Dublin City"
        short_name:     "Dublin City"
        ▼ types:
          0:            "administrative_area_level_2"
          1:            "political"
      ▼ 4:
        long_name:      "County Dublin"
        short_name:     "County Dublin"
        ▼ types:
          0:            "administrative_area_level_1"
          1:            "political"
      ▼ 5:
        long_name:      "Ireland"
        short_name:     "IE"
        ▼ types:
          0:            "country"
          1:            "political"
    formatted_address:  "18 Kevin Street Lower, Dublin, Ireland"

```

Figure 33 - Geocoding JSON Results

These results in Figure 33 are obviously in JSON format and must be parsed, therefore, ParseGeocodingJSON is called, a utility Class for parsing JSON and returning the necessary results.

4.2.1.13 ParseGeocodingJSON.java

ParseGeocodingJSON, as previously mentioned, is called from CollectAddressTask and accepts a JSONObject as a parameter in its constructor. The JSONObject is parsed into a JSONArray and the “long_name” field of the address components 1 and 2 are stored to give the resulting address. In this example, the address would be “Kevin Street Lower, Dublin”.

4.2.1.14 Schedule.java

The Schedule is a RealmObject quite like RawCluster in that it stores the list of Clusters and the week of the year as an identifier. The distinguishable feature between the two Classes is that RawCluster is an unrefined list of Clusters, meaning that no Clusters have been removed yet, while Schedule contains a list of Clusters that have been processed and compared with previous schedules.

4.2.1.15 CreateNotification.java

Following the result of the clustering process, notifications must be made to inform the user of their schedule when the appropriate time comes. The notification in Figure 34 includes the location and the time the user is to arrive by:



Figure 34 - Sample Notification

Creating a notification involves creating a NotificationCompat Builder where the notification channel, content text, title, icon and intent action when clicked is set. The intent in this notification is to open the map to view the relevant route with details such as time or transit details. An AlarmManager is created for the given Cluster’s time to trigger the NotificationPublisher Class. This Class is stored as an intent. A challenge emerged at this point in development as only a single notification was displayed. Upon research ([45](#)) it was

discovered that intents are cached if they are not unique and a method is required to create unique intents. One method of creating a unique intent is to set the data of the intent to the current time in milliseconds:

```
intent.setData(Uri.parse("custom://" + System.currentTimeMillis()));
```

Following this change all notifications are displayed.

4.2.2 Scheduling and Notifications

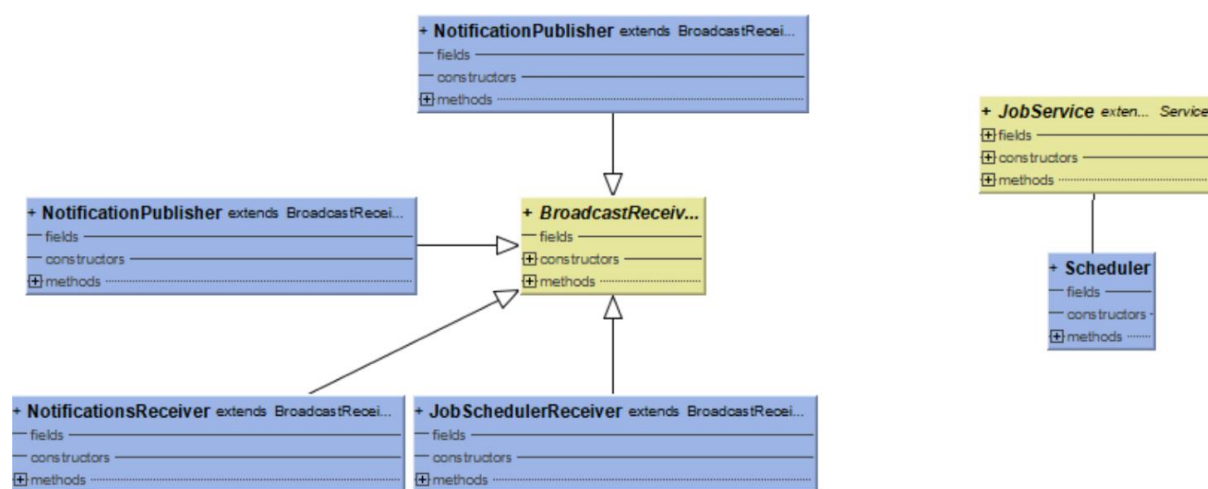


Figure 35 - Scheduling and Notifications Class Diagram

4.2.2.1 Scheduler.java

The Scheduler Class as previously discussed in [4.2.1.1](#) is a utility Class to create a “JobScheduler”, a native Android library, to execute a task when certain conditions are met.

4.2.2.2 JobSchedulerReceiver.java

An important consideration when dealing with the AlarmManager Android library is that when a user’s device is turned off, the AlarmManager is wiped. As a result, a BroadcastReceiver must be set up to restart the AlarmManager to call the Scheduler Class on Sunday morning. This is accomplished by creating a BroadcastReceiver that will be triggered by the device

successfully booting. This is controlled in the Manifest of the application demonstrated in Figure 36:

```
<receiver android:name="com.assistant.mccab.scheduler.JobSchedulerReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Figure 36 - Boot Completed Receiver

Upon a successfully completed boot, JobSchedulerReceiver is called and will collect the most recent Schedule based on the week of the year and proceed to recreate the necessary AlarmManager for Sunday morning.

4.2.2.3 NotificationPublisher.java

Two NotificationPublisher Classes exist and ultimately serve the same goal however due to requirements in the notification design the beginning process is different. One NotificationPublisher is for publishing a notification to the user to alert them that there is a schedule update required and that they must charge the device, seen below in Figure 37:

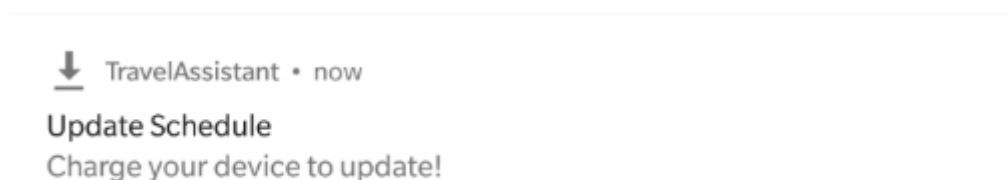


Figure 37 - Available Update Notification

The second NotificationPublisher Class is used to publish each notification for a schedule item. The notification is received through the intent stored in the CreateNotification Class and the user is notified.

Another challenge materialised at later point in the development as the latest version of the Google Play Services required an increase in the TargetSDK version to Android Oreo. With this change then the method of creating a notification relies on a NotificationChannel that is defined by the developer. Although easily overcome when the solution was known, the main

challenge was determining what was causing this issue as it was unclear how an update in TargetSDK could affect notifications. This change however allowed for an easy addition of enabling notification lights and vibration:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{
    NotificationChannel notificationChannel = new NotificationChannel( id: "UpdateNotification01", name: "Channel for schedule updates",
    notificationChannel.enableLights(true);
    notificationChannel.enableVibration(true);
    notificationManager.createNotificationChannel(notificationChannel);
}
```

Figure 38 - Android O+ Notification Channels

4.2.2.4 NotificationsReceiver.java

Much like the JobSchedulerReceiver, when a device is turned off, all notifications are lost. As a result, the BroadcastReceiver NotificationsReceiver is created to recreate a notification for each schedule entry in the latest schedule created for the user. Similarly, the trigger of this receiver is added to the Android Manifest:

```
<receiver android:name="com.assistant.mccab.notifications.NotificationsReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Figure 39 - Boot Completed Receiver

4.2.3 Maps

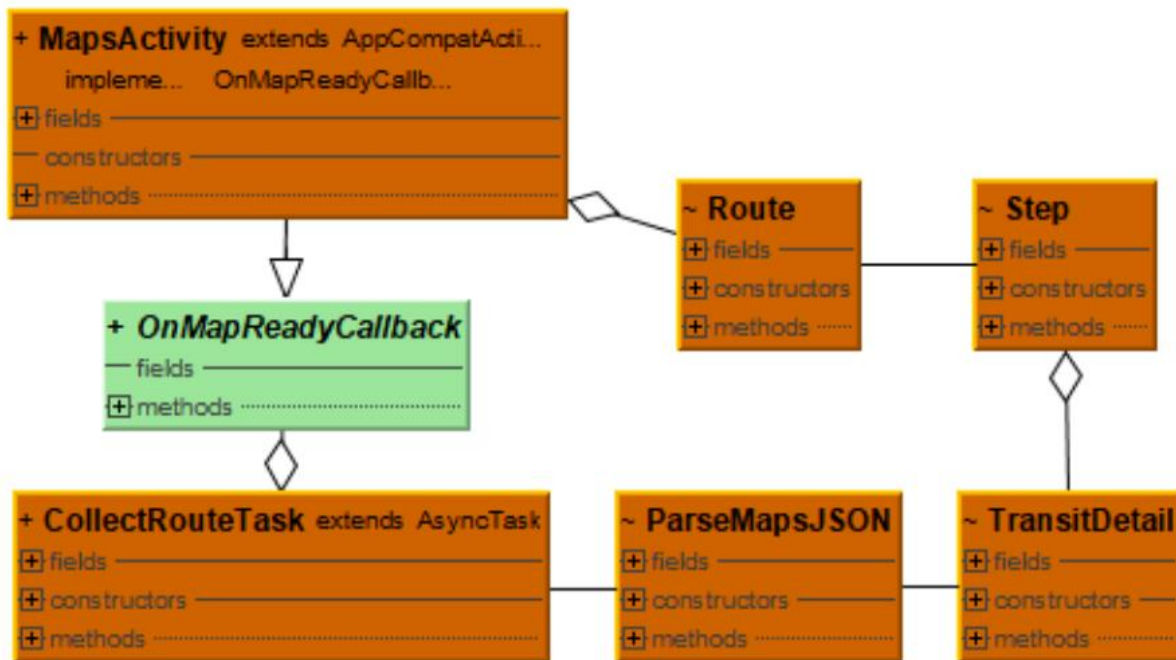


Figure 40 - Maps Class Diagram

4.2.3.1 MapsActivity.java

The MapsActivity is opened when a user views an individual schedule item. The activity can be bootstrapped using the default Google Maps Activity in Android, handling various set up processes for the developer to display a basic map and get the user's current location. Following that, it became the responsibility of the author to develop the added functionality required such as displaying routes between two locations, adding markers at the start and end of relevant steps in the route and adding two views that will contain relevant data. The discussion of the views belongs in the development of the user interface however this section will contain the retrieval of data and creation of routes, as well as offering alternatives and updating the data based on the route selected.

When the activity is started, and the user location is determined, a call to **CollectRouteTask** is made to using the Google Maps Directions API to determine the route details between the user's location and the destination provided by the current schedule item being viewed. When the results are obtained by **CollectRouteTask**, a custom method **drawRoutes()** is called which accepts an **ArrayList** of **Route** Objects. For every **Step** Object in each **Route**, a new polyline is added, with markers being added at the start of the first step and end of the last step. The first **Route** is drawn using a solid colour while the alternatives are drawn using a transparent colour.

Each polyline has an `onClickListener()` method to update the colour and update the details of each route.

The most important function of the `MapsActivity` is at the beginning when the Google Maps Directions API URL is formed in Figure 41:

```
// Build the URL and return it
return "https://maps.googleapis.com/maps/api/directions/" + output + "?" + parameters + APIKey;
```

Figure 41 - Google Maps Directions API URL

The URL expects the following parameters:

1. Output: The format of the output, in this case, JSON.
2. Parameters:
 - i. Origin: The origin latitude and longitude which is the user's location
 - ii. Destination: The destination latitude and longitude
 - iii. Mode: The method of travel such as walking or cycling
 - iv. Alternatives: Provide more than one route

The destination and mode are determined by the relevant schedule item selected. The arrival time and departure time are excluded so the user may always get directions and transit times based on the current time in the event they stray from their schedule. All possible parameters are available in the Directions API documentation. (46) An example URL is:

<https://maps.googleapis.com/maps/api/directions/json?origin=53.31562084621854,-6.316883458031548&destination=53.337620,-6.267877&mode=walking&alternatives=true&key=APIKEY>

Which returns a list of routes and for every route there is a list of legs and for every leg there is a list of steps. The leg will contain an overall distance and duration with a start and end location, while steps include more specific and shorter journeys that make up the entire leg. The steps allow for customisation to adjust the duration based on user speed's if the user is active as compared to adjusting the entire leg by their speed but also affecting transport duration. An example screenshot of the results is below in Figure 42:

```

▼ routes:
  ▼ 0:
    ▼ bounds:
      ▼ northeast:
        lat: 53.3393982
        lng: -6.2678631
      ▼ southwest:
        lat: 53.3156624
        lng: -6.317120399999999
    copyrights: "Map data ©2018 Google"
    ▼ legs:
      ▼ 0:
        ▼ distance:
          text: "4.8 km"
          value: 4830
        ▼ duration:
          text: "59 mins"
          value: 3559
        end_address: "18 Kevin Street Lower, Dublin, Ireland"
        ▼ end_location:
          lat: 53.3376566
          lng: -6.2678631
        ▼ start_address: "Unit 9, Ashleaf Shopping Centre, Cromwellsfort Rd, Walkinstown, Dublin 12, Ireland"
        ▼ start_location:
          lat: 53.3156624
          lng: -6.317120399999999
        ▼ steps:
          ▼ 0:
            ▼ distance:
              text: "70 m"
              value: 70
            ▼ duration:
              text: "1 min"
              value: 48
            ▼ end_location:
              lat: 53.3162835
              lng: -6.3169587
            ▼ html_instructions: "Head <b>north</b> on <b>Whitehall Rd W</b> toward <b>Cromwellsfort Rd</b>"
            ▼ polyline:
              points: "{elIdI~xpe@EAEAMCUGKCMCKCOAUA}"
            ▼ start_location:
              lat: 53.3156624
              lng: -6.317120399999999
            travel_mode: "WALKING"
          ▼ 1:

```

Figure 42 - Directions JSON Results

4.2.3.2 CollectRouteTask.java

The CollectRouteTask is an asynchronous task that will form an HTTP connection and read the data returned like the CollectAddressTask referenced in [4.2.1.12](#). This Class initialises a new ParseMapsJSON Class to parse the JSON Data.

4.2.3.3 ParseMapsJSON.java

ParseMapsJSON takes a JSONObject as a parameter in its constructor and will proceed to parse every route, leg and step and store the relevant details collected. These details include the step's distance, duration, end and start location, encoded polyline points and travel mode as well as collecting transit details if the step has them, which also includes the arrival stop, departure stop, number of stops, duration, transit name and transit type. Each time a step is iterated, a new Step object is created, potentially containing another TransitDetail object. The collection of steps is then added to a new Route object to form a complete route, containing the bounding box coordinates and time. The bounding box coordinates are used to zoom the map in or out based on the coordinates.

4.2.3.4 Route Objects

The three Classes required to store various details for the route functionality in the map include the Route Object that stores a list of Step Objects which stores step information and a TransitDetail Object to store Transit step information.

4.2.4 Visuals

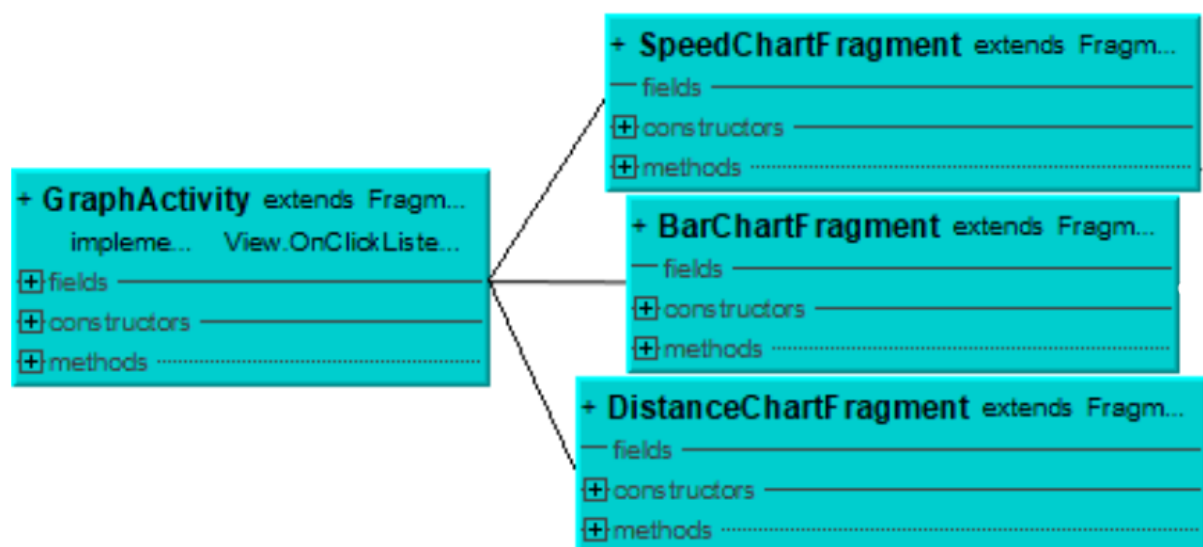


Figure 43 - Visuals Class Diagram

4.2.4.1 GraphActivity.java

A Fragment that contains a bottom navigation bar to switch between three graphs available to the user and a row of buttons on the top of the device's screen to allow for alternating between different weekly schedules and in the event of the speed chart, an extra row of buttons to choose between days. Complexity within this application lies in the selection of the graph and the appropriate preparation of the data based on the week or the day as well as when to display the day buttons. For each graph, a new child fragment is created and displayed to allow for a smooth transition between the graphs. Each graph is created using the MPAndroidChart library ([23](#)), combining tutorials and knowledge obtained from the documentation. ([47](#))

4.2.4.2 BarChartFragment.java

A Fragment to display a bar chart with the total activity duration by day for a given week. The user may wish to visualise this data to adjust the duration spent in vehicles they may use and be more physically active instead. Below is an example of the bar chart in Figure 44:

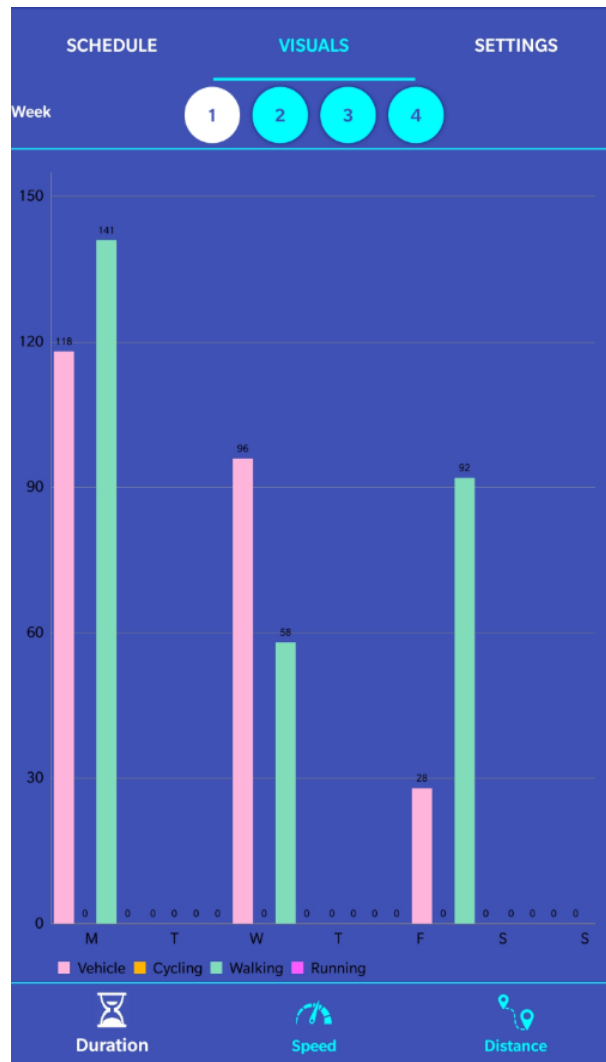


Figure 44 - BarChartFragment

4.2.4.3 SpeedChartFragment.java

For a given day, display a line chart showing user speeds based on all possible activities such as walking, running, cycling or in a vehicle. The user can change the day and the week. A user may wish to visualise this data to adjust their schedule according to fitness desires such as walking faster or to opt for less vehicles in the evening as they may walk faster than the bus in traffic. Below in Figure 45 is an example graph for a Wednesday showing that at 4 o'clock, the user is on average, able to walk faster than the bus in traffic:

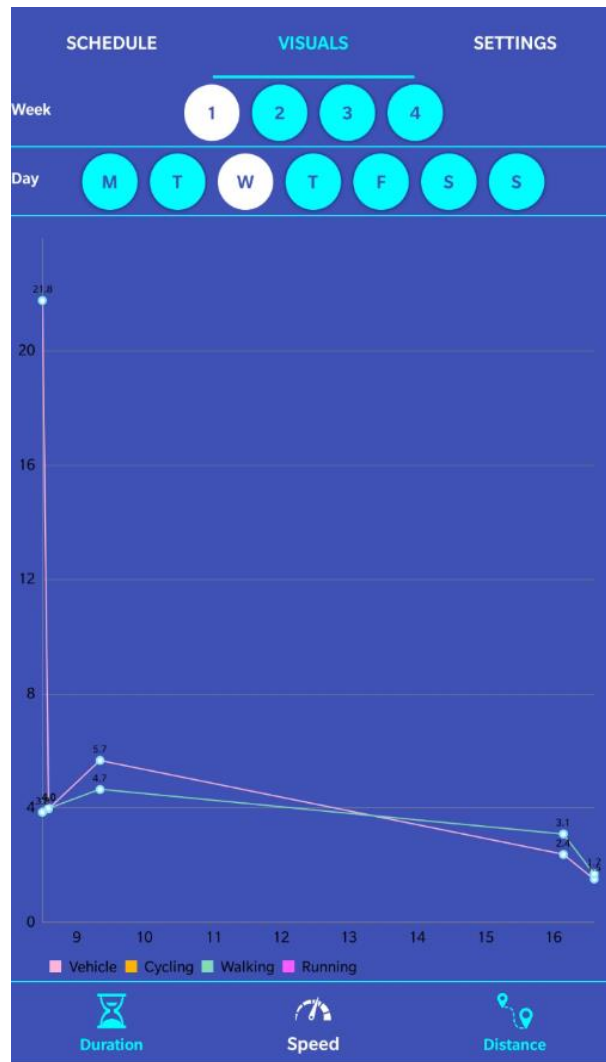


Figure 45 - SpeedChartFragment

4.2.4.4 DistanceChartFragment.java

A Fragment to display a bar chart with the total distance per activity by day for a given week. The reason this graph is selected is to provide users with a way of visualising their life style, for example, if the user wishes to increase the amount of walking they do, they can view the distance they are travelling. Figure 46 below is an example screenshot demonstrating data on Mondays, Wednesdays and Fridays where the activities included walking and vehicles:

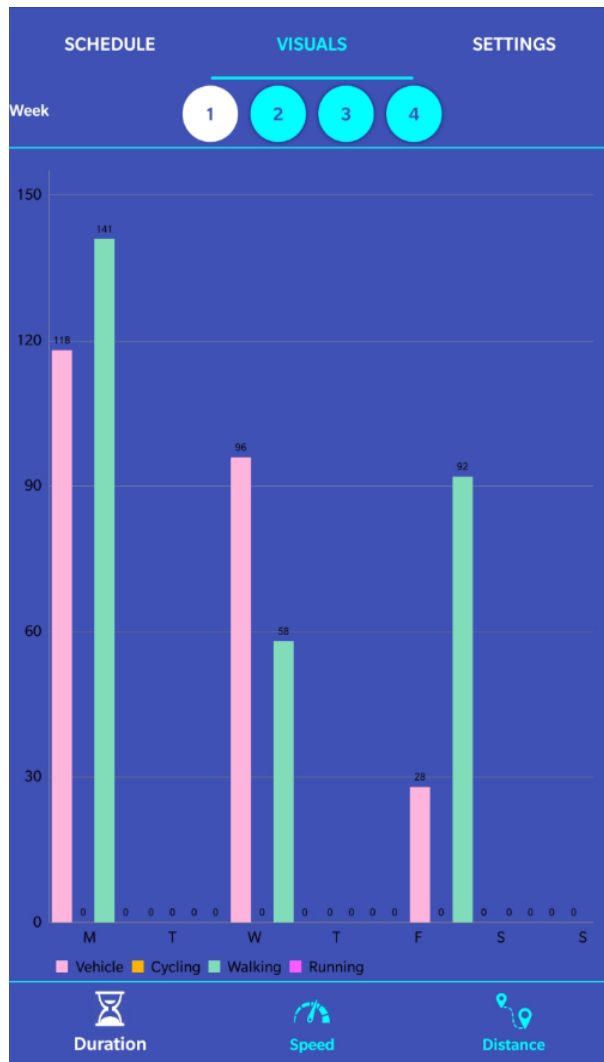


Figure 46 - DistanceChartFragment

4.2.5 UI

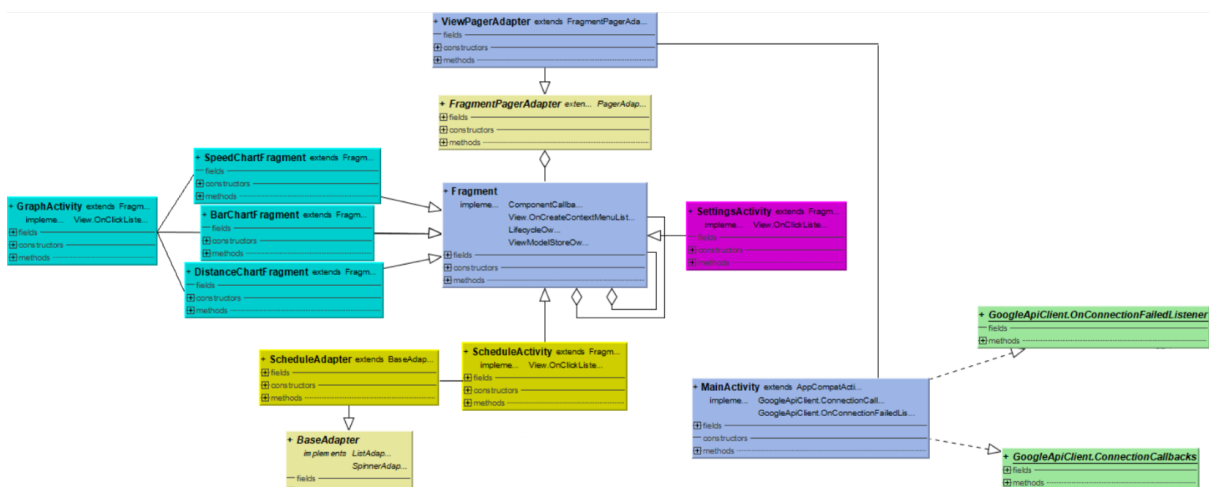


Figure 47 - UI Class Diagram

4.2.5.1 MainActivity.java

An important component to discuss in any Android application is the UI. The UI for this application was constructed in XML and adheres to the Google Material Design guidelines. (48) The MainActivity is the first activity loaded by the application when it is first opened. This activity has several tasks to perform first such as connecting to the Google Play Services and preparing the schedule and graph fragments, therefore a splash screen is displayed while this process is occurring. Figure 48 is an example of the splash screen is below:

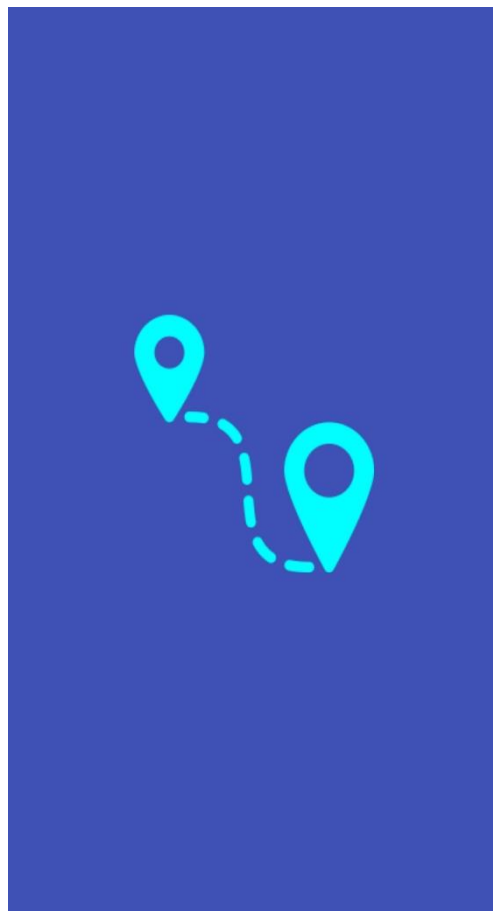


Figure 48 - Splashscreen

The MainActivity also includes a TabLayout to display the options to the user. These options include viewing the schedule and viewing the schedule. Currently settings contain developer options for demonstration purposes. The TabLayout here conforms to the material guidelines that may be found on the material.io website (49). A ViewPager was implemented to allow swiping between tabs. The concept of using a TabLayout was decided upon from the research

section when viewing existing applications such as IMDb and Twitter. The GraphActivity utilises a bottom navigation menu which was inspired from other applications such as YouTube and Spotify.

4.2.5.2 ViewPagerAdapter.java

The ViewPagerAdapter is a custom adapter to implement the given fragment when a user swipes between tabs. The position is obtained in the getItem() method and a new fragment is instantiated to display the appropriate fragment:

```
// Get each Fragment based on the current item
@Override
public Fragment getItem(int position)
{
    // Create a new Fragment
    Fragment fragment;

    switch(position)
    {
        case 0:
            fragment = new ScheduleActivity();
            return fragment;

        case 1:
            fragment = new GraphActivity();
            return fragment;

        case 2:
            return new SettingsActivity();
    }
}
```

Figure 49 - getItem()

With the Page Title and the number of items being declared in this adapter as well:

```

// Set the number of tabs
@Override
public int getCount() { return 3; }

// Set the title of each tab
@Override
public CharSequence getPageTitle(int position)
{
    switch(position)
    {
        case 0:
            return "Schedule";

        case 1:
            return "Visuals";

        case 2:
            return "Settings";
    }
}

```

Figure 50 - getCount() & getPageTitle()

4.2.5.3 ScheduleActivity.java

The ScheduleActivity was designed to display the user's schedule in an efficient way and remain intuitive and familiar. The challenge here was finding a design that would display the appropriate data in a user friendly and aesthetic way. The following Figure 51 demonstrates the transformation from the original display of information to the final product:

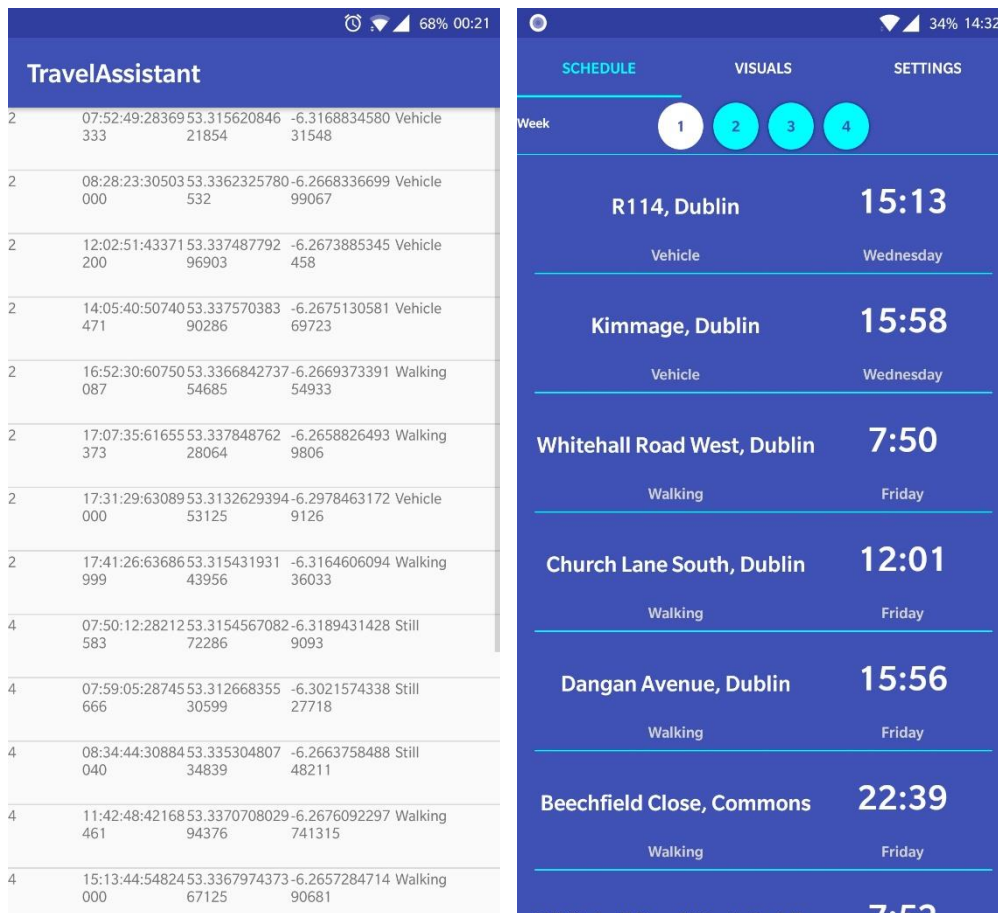


Figure 51 - Comparison Between Original and Current

The final design of the schedule displays all relevant detail and allows for switching between alternative schedules. Once again conforming to the material guidelines, the current option selected is highlighted in an alternative colour. In this example, white.

Another challenge faced in the design of this schedule was the order of the results. The schedule was to display all data in an order that made sense to the user. For example, if it was a Friday and the user was to view their schedule, instead of first scrolling through several days or complicating the view with an extra day selection, the list is always sorted based on the current day and time. This is achieved by first sorting the clusters in order of day and time:


```

// Sort the clusters
Collections.sort(clustersList, new Comparator<Cluster>()
{
    public int compare(Cluster cluster1, Cluster cluster2)
    {
        // Compare the days
        Integer day1 = (int) cluster1.getDay();
        Integer day2 = (int) cluster2.getDay();
        int dayComparison = day1.compareTo(day2);

        // If the days aren't the same
        if(dayComparison != 0)
        {
            return dayComparison;
        }

        // If the days are the same
        else
        {
            // Compare the time
            Integer time1 = (int) cluster1.getTime();
            Integer time2 = (int) cluster2.getTime();
            return time1.compareTo(time2);
        }
    }
});

```

Figure 52 - Cluster Day & Time Comparison

Then determining a list of clusters that are before the current day and time to remove them and append them to the end. This is achieved by using `Math.abs()` for the absolute value of the time as a Monday, Tuesday or Wednesday is considered before the Unix Epoch and thus a negative value:

```

// For each Cluster
for(Cluster cluster: clustersList)
{
    // Set the calendar time based off the cluster
    calendar.setTimeInMillis((long) cluster.getTime());
    calendar.set(Calendar.DAY_OF_WEEK, (int) cluster.getDay());

    // Only compare days less than or equal to the current day
    if(cluster.getDay() <= currentDay)
    {
        // If the current time is less than the cluster time, meaning it's
        if(Math.abs(milliseconds) < Math.abs(calendar.getTimeInMillis()))
        {
            removedClusters.add(cluster);
        }

        else if(milliseconds > calendar.getTimeInMillis())
        {
            removedClusters.add(cluster);
        }
    }
}

```

Figure 53 - Cluster Removal

4.2.5.4 ScheduleAdapter.java

The ScheduleAdapter is the Class that is defined to customise each entry in the schedule list and assign the correct values to each field. Although not overly complex for any Android developer, it is worth noting as without it the schedule lacks all content.

4.2.5.5 GraphActivity.java

The GraphActivity as previously discussed in section [4.2.4.1](#) is highly relevant to the UI as it is responsible for the creation of all visuals in the application.

4.2.5.6 MapsActivity.java

The final user activity that was a key element in the development of the application. As previously discussed, the map is responsible for displaying relevant information such as location, times, duration and distance of a journey based on a specific schedule item. This section however will discuss the design choices where this application attempts to improve

upon current applications that prove to be cumbersome and complex when displaying a simple route. An example comparison between Google Maps and this application is below:

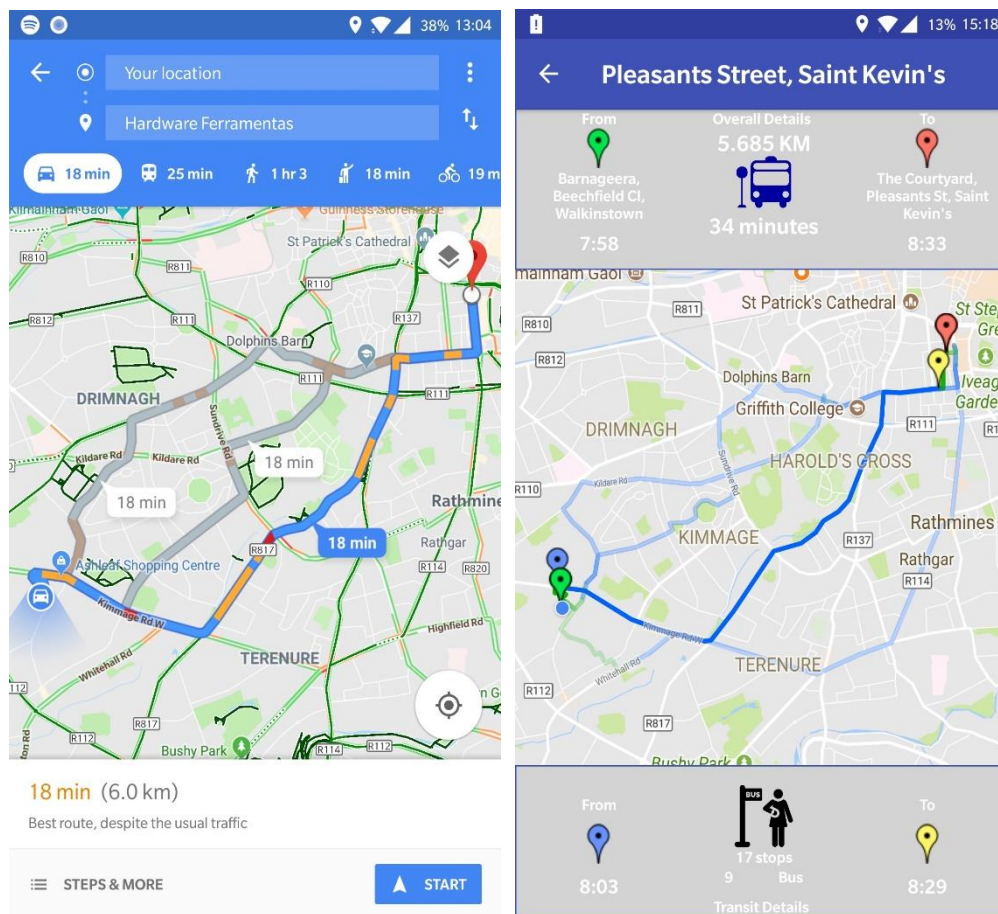


Figure 54 - Google Maps Comparison

The nature of maps includes displaying a lot of information, especially when alternative routes and transit details are to be displayed. Google Maps' inclusion of banners for each route and two floating action buttons adds clutter to the view and complicates the overall usage of the application.

Research into the spatial abilities necessary for map reading is being carried out by geographer Amy Lobben who has demonstrated that when reading a map, the parahippocampal place area within the hippocampus of the brain is active. (50) This area of the brain is a part of the navigation and spatial orientation for an individual. A conclusion can be drawn from this that the processing of a map ultimately comes down to the individual's own capabilities and although an application cannot improve this for that individual, the developer can attempt to make the map less cluttered and easier to read to allow for easier processing.

Therefore, it was a challenge to display all relevant information in a clutter-free way. The final design incorporates two views, one for overall journey information and one for transit information. In the event of walking, the transit information view is not available and nor is the traffic floating action button that appears when a user hides the information views. The design choice to only show the button when the views were hidden was to reduce the overall clutter. As stated in the material guidelines ([51](#)), it is recommended only one floating action button per view. Finally, as all navigation through the application is possible through menus but the map does not allow for this, a back button is in place in the top left of the menu bar. Accompanying this button is the name of the location the user will travel to.

Below are the final four screenshots of the map activity, demonstrating a map of only walking (a green route), walking and taking a bus (green and blue route), the information views hidden, and the traffic button pressed showing the traffic layer on the map:

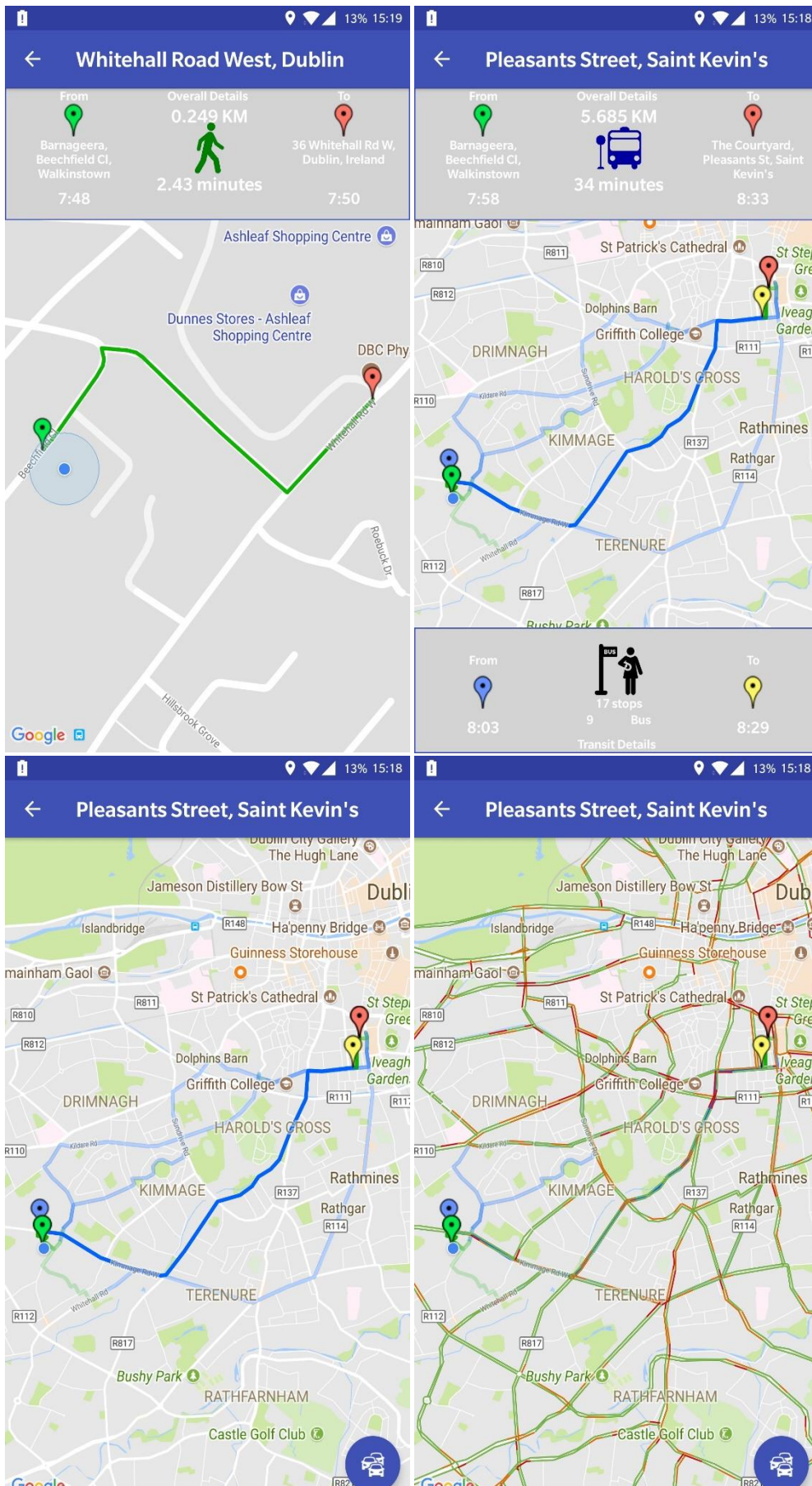


Figure 55 - MapsActivity Example

The traffic layer button was a necessity because of the constraints applied by the Google Maps API. It is currently impossible to overlay a custom route over the traffic layer; therefore, it must be provided as an option to users to reduce the overall clutter on the map.

4.2.6 Time

An important aspect throughout the entire application that proved to be a source of complexity was the translation of time. When comparing two times, for example 12 p.m. on Monday the 2nd of April and 12 p.m. on Monday the 9th of April, converts to a comparison between 1522666800000 and 1523271600000 milliseconds. Obviously in a clustering process, a relationship cannot be made between these two times and so they must be converted to the same scale, taking only the time and counting it from the Unix Epoch (01/01/1970). This now becomes a comparison between 43200000 and 43200000 milliseconds. The complexity materialises at this point as weeks and days must be remembered and returned accordingly however is not overly challenging.

The true challenge faced with time conversion was in using the Google Maps Directions API. The API expects time in seconds and the date must be included, therefore the Java Calendar was utilised in many Classes where time must be altered. An example of the Calendar library in this application is below in Figure 56:

```
// Create a calendar to get the time
Calendar clusterTime = Calendar.getInstance();
clusterTime.set(Calendar.HOUR_OF_DAY, 0);
clusterTime.set(Calendar.MINUTE, 0);
clusterTime.set(Calendar.SECOND, 0);
clusterTime.set(Calendar.MILLISECOND, 0);

// Add the time to the calendar
clusterTime.add(Calendar.MILLISECOND, (int) time);
```

Figure 56 - Calendar Library

The previous block of code gets the current date and time by using `Calendar.getInstance()` and proceeds to set the time to midnight. Following that, the time is added in milliseconds. In the usage with the Google Maps Directions API, the time added was in seconds. Similarly, when

querying the Google Fit Store for activity data at a certain time on a given day in a specific week, the following code was necessary:

```
// Create Calendar and Date objects
Calendar calendar = Calendar.getInstance();

// Set the time to 0
calendar.set(Calendar.HOUR_OF_DAY, 0);
calendar.set(Calendar.MINUTE, 0);
calendar.set(Calendar.SECOND, 0);
calendar.set(Calendar.MILLISECOND, 0);

// Set the day to the current day in the loop
calendar.set(Calendar.DAY_OF_WEEK, (int) day);

// Set the week to current week in the loop
calendar.add(Calendar.WEEK_OF_MONTH, (int) week * -1);

// Add the cluster's time
calendar.add(Calendar.MILLISECOND, (int) startTime);
```

Figure 57 - Time Manipulation

5. System Validation

5.1 Testing

As stated as a part of Extreme Programming, unit tests should be crafted first, then functionality is written that will pass the tests. This approach was followed at the beginning however as the difficulty of the project increased and the project due date approached, it was challenging to remember to write appropriate tests under pressure. However, following the functionality, tests were later written.

Due to time constraints the testing performed in this project lacks any testing frameworks. Instead, the testing performed in this project was a combination between white box testing in the form of creating unit tests and mock objects to test the system functionality as well as user validation.

5.1.1 White Box Testing

White box testing enables testing the application from the developer's point of view. An advanced knowledge of the entire application must be known. This option is more expensive in terms of time however it is the opinion of this author that it is a stronger approach to testing than black box testing which simply tests that the correct result is achieved, but not how it was achieved. By testing the inner workings of the application, code quality is improved, and the likelihood of unknown bugs is reduced. White box testing was achieved by creating sample mock Objects and unit tests. Examples of this testing include testing the clustering process on demand, testing notifications and exporting the Realm Database to view the data in Realm Studio.

5.1.1.1 Clustering Process

This process was achieved by a simple button click on the MainActivity before the design was in place. This button would trigger the schedule to occur immediately, creating the chain of events that is outlined in section [4.2.1](#). This process was more about runtime testing and ensuring the application would not crash based on different users and thus, different data sets. By logging into a user's account that has no Google Fit data, originally the application did

crash as it expected data when the clustering process would run. This has since been corrected and in the event the user deletes Google Fit data then the application will not crash. Following this process, by testing different data sets that do exist, different schedules can be created. This process tests the functionality of the clustering process and tests for various bugs that may exist when different data is included such as cycling or running, as well as testing the overall quality of the schedule created. The results of this test were quite positive with no bugs found in the clustering process when data is provided.

5.1.1.2 Testing Notifications

The process of testing notifications is important as it is a crucial component in the usability of this application. The notifications may be tested by a button click within the settings app which will manually trigger a notification, indicating a journey to College Green, Dublin. The result of this test is visible in the demonstration video in section [5.2](#). This test utilises a mock Object as it creates a mock Cluster. The relevant fields of this Cluster were edited to test the functionality of the notification and subsequently the functionality of the map based on the data provided by the Cluster, such as different activity types, locations and times. The following code in Figure 58 demonstrates creating a mock Cluster before creating a new notification:

```

case R.id.testNotification:
    // Create a Calendar for the time and day
    Calendar calendar = Calendar.getInstance();

    // Add 45 minutes to the current time
    calendar.add(Calendar.MINUTE, 45);

    // Create a Cluster
    Cluster cluster = new Cluster();

    // Set the time of the Cluster
    cluster.setTime(calendar.getTimeInMillis());

    // Set the Latitude and Longitude of the Cluster
    cluster.setLatitude(53.344425);
    cluster.setLongitude(-6.259294);

    // Set the activity type of the Cluster
    cluster.setActivity("Vehicle");

    // Set the address
    cluster.setAddress("College Green, Dublin");

    // Create a notification to display after ten seconds
    new ScheduleNotificationTest(cluster, getActivity(), ID: 666);

```

Figure 58 - Mock Object

5.1.1.3 Realm Database Testing

It is important to determine whether the database is correctly populated or not when developing an application with large amounts of data. To view a local database first the data must be collected from the device. Realm requires several extra steps to write the database to a file and with the help of a tutorial ([52](#)) the database may be exported to local storage. From there, through whatever suitable means possible, transfer the file to a PC. Using the application Realm Studio ([53](#)) it is possible to then view the content of the database in a clearer way than viewing it through a debugger or in an activity. For example, the following screenshot in Figure 59 demonstrates the content of the database, showing the 19 Clusters, 355 ClusterInstances (data collected from Google Fit that is filtered and clustered into the 19 Clusters), 1 RawCluster (containing a list of Clusters) and the various schedules created:

CLASSES		Enter a query to filter the list										?	Cr
Cluster	19	day int	minTime double	time double	maxTime double	minLatitude double	latitude double						
ClusterInstance	355	2	28236745.417007845	28369333.333333332	28501921.24965882	53.31522766272866	53.31562084621854						
RawCluster	1	2	30050119.67559276	30503000.000000004	30955880.324407246	53.334821349857584	53.3362325780532						
Schedule	10	2	43247587.056497075	43371200.000459366	43494812.94442166	53.33740837416917	53.3374879296903						
		2	49471620.1770564	50740471.844036795	52009323.51101719	53.33692572913705	53.3375703890286						
		2	59211264.1162924	60750087.499744326	62288910.88319625	53.33551465725176	53.336684273754685						
		2	61428397.53741099	61655373.44975034	61862349.36208969	53.33763374710749	53.33784876228064						
		2	48001489.92434328	63089000.00000001	77376510.07565674	53.303309756801944	53.313262939453125						
		2	63307486.06938693	63686899.99999994	64066513.93061295	53.31458129386226	53.31543193143956						
		4	28054519.813225467	28212583.333333332	28370646.853441197	53.314877288171594	53.315456708272286						
		4	28701529.5067081	28745666.666666666	28789803.82662522	53.31250263238271	53.3126683530399						
		4	30571674.326120067	30884040.72482061	31196407.123521153	53.33394312286037	53.335304807348376						
		4	38976311.984305695	42168461.44080309	45360610.89730048	53.3366988411801	53.337070820949376						
		4	54346348.70620061	54824000.795923166	55301652.885645725	53.335070818963715	53.336797437367125						
		4	45582516.756637886	57502000.00000001	69421483.24336213	53.30112599770888	53.3122940063476556						
		6	28073885.92498292	28238542.85714285	28403199.78929741	53.31464383091336	53.31510707310268						
		6	35616286.18460416	33288929.292929284	50961572.40125441	53.33563302798196	53.33702623001254						
		6	56048854.97771789	57407312.499999985	58765770.02228208	53.31396386479817	53.31508517265319						
		6	63317568.56263697	72881000	82444431.43736303	53.30584997874744	53.30619730268205						
		6	81108437.62705134	81552727.2727273	81997016.91840325	53.3146542177649	53.3155632019043						

Figure 59 - Realm Data

A closer look at the RawCluster Class demonstrates that it contains a list of Clusters as well as a week of the year identifier:

Cluster	19	clustersList	weekOfY...
ClusterInstance	355	Cluster[]	int
RawCluster	1	[list of Cluster]	14

Figure 60 - RawCluster

By viewing this data in a GUI or copying the data to Excel for further data analysis, conclusions can be drawn from the data and if any form of errors have occurred during the process. Similarly, data may be manipulated here and imported as the application's database to view the results of a data change.

5.1.2 User Validation

Several tests on friends and family were performed. When developing an application, it is easy to suffer from tunnel vision and not realise that the design is not understandable to those who have not developed the project or is too complex for others who are unfamiliar with technology. As a result, tests were performed to get opinions on UI design, visuals and to test the operation of the application on other devices.

5.1.2.1 UI Design

A personal weakness of the author of this report is in UI design. A personal value of functionality over design impedes on design choices and therefore, as design is so ultimately crucial to an Android application, advice was extremely valuable to this project. After

describing the problem to friends and family, a common opinion was that the schedule design should mimic familiar scheduling applications and alarm applications as their design is intuitive and descriptive. The outcome of all discussions of UI design resulted in the final design with no negative comments on the final design.

5.1.2.2 Visuals

When analysing data sets the developer is easily accustomed to the data in its raw form. For example, all times are in milliseconds in their raw form in this application and in an oversight on the author's behalf, the original visuals used milliseconds for time. This was quickly corrected after the initial testing of visuals with family as they were extremely puzzled by the axis on each graph.

5.1.2.3 Other Devices

The final test was to determine if the application could run effectively on another device. Two extra devices were selected in this process. From launching the application on these devices, the following was learned:

1. The chosen version of Realm database was malfunctioning on one device. This device was operating on the minimum supported SDK version of this application, Android M, and the conclusion was that this was the underlying cause of the malfunction. This is a valuable lesson in terms of version inconsistencies and would be undetected if it was not for this test. Due to time constraints however, it was not possible to correct this and test the application again on the selected device. Ultimately because of the database malfunctioning, the application crashes.
2. The other device proved successful however. The device used for production of the application was running on Android O by the final development stage while this test device was running Android N. The database succeeded in this case and because of the user already having Google Fit installed, the clustering process completed successfully as well. The results of the schedule were considered accurate by the test user and the outcome of the test proved to be a success.

5.2 Demonstration

As most major application features have already been demonstrated through screenshots in sections [3.2.1](#) and [4.2](#), most notably [4.2.4](#) and [4.2.5](#), repetition of the screenshots is not

necessary however below is a URL to a demonstration video of the application in use:

<https://youtu.be/bWWTEkgjlvY>

Most notable from the video that is not visible or fully expressed through screenshots are:

1. The receiving of notifications and the following process of opening the map and changing routes as well as displaying traffic.
2. The animations in showing and hiding the day picker as well as map details, along with the swiping transition between views on the home screen.

The rest of the functionality of this application relies on a slow process of collecting data and the subsequent machine learning that follows. This process is impossible to demonstrate however the result is visible.

6. Project Plan

In the interim stage of the project plan, the following were the objectives planned for the final project:

1. Improve the current clusters for user schedules.
2. Add a classification for user travel to better calculate estimates for travel time instead of using an average.
3. Develop data visualisation for users.
4. Implement an effective way of scheduling classification.
5. Implement custom routing in the map to allow users to create routes based on fitness requirements.

From the above list of requirements, all objectives were accomplished except for objective five. This objective was set as a potential feature if the complexity of the previous objectives was not sufficient. However, through the development of the application, each objective was sufficiently complex and challenging and as a result, the fitness aspect of the application suffered. The original mockups and objectives called for more fitness related features such as a calorie count and the previously mentioned planned route that would consider user speeds already obtained.

Perhaps the most significant major change to the development of this application was in the decision to change the clustering algorithm from Simple K-Means to EM. As previously discussed in section [4.2.1.7](#) the EM algorithm is an iterative algorithm that does not require a set number of clusters before running. This reduces the error in using Simple K-Means as a method of determining the number of clusters is now suppressed.

If the project was to be repeated, improved time management would be crucial. Although time management for this project was satisfactory, to achieve each objective time management must improve. Similarly, if the project was to be repeated with the knowledge and mistakes of this project, the entire project would be a greater success. Taking on board the knowledge obtained in machine learning and complications in APIs and time conversion, the process of obtaining and clustering the data could take less time and greatly improve the project.

7. Conclusion

7.1 Project Summary

It is the opinion of this author that the result of the project was a success. The scheduling process is extremely accurate for the most part with most entries in the schedule more accurate than the Google Maps alternative. An area of improvement however is within location. The determined location is not as accurate as Google Maps' estimate although it is still close. An alternative method of location evaluation could result in an improved estimate however was not feasible within the time constraints of this project.

That is not to say that there are not areas where the project fell short. There were several shortcomings in this project that exist due to time constraints and the pressure from the college year. An example of this is in the development of fitness related details incorporated into the project. Similarly, in testing, there was much more hope in applying a proper testing framework to the application to develop more automated and rigorous tests.

Despite the short list of shortcomings however, there was still many successes within the project. The areas within the project that were a success include:

1. Successfully collecting data from the user's device
2. Processing the data and applying a clustering algorithm to generate a tailor-made schedule
3. Scheduling the data collecting and clustering process in a secure and battery efficient way
4. Provide the user with relevant and accurate journey information
5. Provide all features in an aesthetically pleasing and intuitive interface

For the previously mentioned areas of the project, it is a reasonable to be extremely satisfied with their result. Similarly, the same satisfaction may be shared with the complexity of the project. The project incorporated a challenging task that has yet to be implemented correctly, while also working with somewhat complex APIs and integrating machine learning, an area of which this author had no previous experience in.

On the topic of the author's experience in the field of machine learning, skills were obtained that are only earned through experience. The same comment may be made about all aspects of the project, most notably the main programming language, Java, and other areas of the project such as Android design in XML and the Android architecture. Android is an area of development that is of great interest to the author of this report and all skills and experiences obtained are extremely valuable for the future. It is of the opinion of this author that in the future, the mistakes experienced, lessons learned, and skills obtained from this project may be applied to future endeavours.

7.2 Further Development

When looking to the future with this application there is certainly plenty of features that may be added and improved, given the time. A project that includes fitness data, visuals and maps can appear to have a wide scope. It is easy to lose sight of the goal but given the time, features may be added for users based on user feedback. For example, the application could benefit largely from usability features such as more customisable visuals as well as a community-based features to share statistics or anonymously compare with those in your area. Similarly, by developing the application on other platforms and allowing the data to be visible online in a web application would be a feature many users would enjoy.

Areas of improvement on existing features may consist of:

1. Improving the location estimation
2. Allow for design customisation based on user preference
3. Improve upon the current lack of fitness features

Ultimately, there is a long list of potential features and improvements to be made to the application. The decision of what to add depends on the direction the application will take in the future if development was to continue.

8. Bibliography

1. Morrison K. Millennials Drive Growing Demand for Mobile Fitness Apps [Internet]. – Adweek. Adweek; 2015 [cited 2018Apr5]. Available from: <http://www.adweek.com/digital/millennials-drive-growing-demand-for-mobile-fitness-apps/>
2. Pewinternetorg. Pewinternetorg. [Internet]. [cited 2018Apr5]. Available from: http://www.pewinternet.org/files/old-media/Files/Reports/PIP_HealthOnline.pdf
3. Morony J. What's the Difference between Native, Hybrid and Web Mobile App Development? [Internet]. joshmorony - Build Mobile Apps with HTML5. 2015 [cited 2017Nov24]. Available from: <https://www.joshmorony.com/whats-the-difference-between-native-hybrid-and-web-mobile-app-development/>
4. Pronschinske M. The State of Native vs. Web vs. Hybrid - DZone Mobile [Internet]. dzone.com. 2016 [cited 2017Nov24]. Available from: <https://dzone.com/articles/state-native-vs-web-vs-hybrid#>
5. Choosing the Right Technology for Your Mobile App Strategy [Internet]. Signal. 2015 [cited 2017Nov24]. Available from: <https://signalinc.com/choosing-the-right-technology-for-your-mobile-app-strategy>
6. Google Ends Support for Eclipse Android Developer Tools [Internet]. xda-developers. 2016 [cited 2017Nov24]. Available from: <https://www.xda-developers.com/google-ends-support-for-eclipse-android-developer-tools/>
7. Platform Overview | Google Fit | Google Developers [Internet]. Google. Google; [cited 2017Nov24]. Available from: <https://developers.google.com/fit/overview>
8. Stack Overflow [Internet]. Stack Overflow [cited 2017Nov24]. Available from: <https://stackoverflow.com/a/4233638>
9. admin A. Google Maps vs Open Street Maps Comparison. Who is Winner? [Internet]. Agile Store Locator. 2017 [cited 2017Nov24]. Available from: <https://agilestorelocator.com/blog/google-maps-vs-open-street-maps-comparison/>
10. Buczkowski A. Why would you use OpenStreetMap if there is Google Maps? [Internet]. Geoawesomeness. 2016 [cited 2017Nov24]. Available from: <http://geoawesomeness.com/why-would-you-use-openstreetmap-if-there-is-google-maps/>
11. osmdroid [Internet]. osmdroid - OpenStreetMap Wiki. [cited 2017Nov24]. Available from: <http://wiki.openstreetmap.org/wiki/Osmdroid>
12. Google. Google; [cited 2017Nov24]. Available from: https://developers.google.com/maps/documentation/android-api/config#step_2_install_the_google_play_services_sdk
13. Getting Started on Android | Google Fit | Google Developers [Internet]. Google. Google; [cited 2017Nov24]. Available from: https://developers.google.com/fit/android/get-started#step_5_connect_to_the_fitness_service
14. Ali Z. Android Version Distribution For August 2017: Nougat Drags to 13.5% [Internet]. Wccftech. 2017 [cited 2017Nov24]. Available from: <https://wccftech.com/android-distribution-august-nougat/>
15. Top 5 machine learning libraries for Java [Internet]. JAXenter. 2017 [cited 2017Nov24]. Available from: <https://jaxenter.com/top-5-machine-learning-libraries-java-132091.html>

16. Biewald L. Build a super fast deep learning machine for under \$1,000 [Internet]. O'Reilly Media. 2017 [cited 2017Nov24]. Available from: <https://corporate.oreilly.com/learning/build-a-super-fast-deep-learning-machine-for-under-1000>
17. Ferron E. 2017 Smartphone Comparison Guide [Internet]. New Atlas - New Technology & Science News. New Atlas; 2017 [cited 2017Nov24]. Available from: <https://newatlas.com/best-smartphones-specs-features-comparison-2017/49418/>
18. BatteryManager [Internet]. Android Developers. 2017 [cited 2017Nov24]. Available from: <https://developer.android.com/reference/android/os/BatteryManager.html>
19. Weka 3: Data Mining Software in Java [Internet]. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. [cited 2017Nov24]. Available from: <https://www.cs.waikato.ac.nz/ml/weka/>
20. santhosh-kumar. santhosh-kumar/weka-android [Internet]. GitHub. [cited 2017Nov24]. Available from: <https://github.com/santhosh-kumar/weka-android>
21. 5 Reasons Why You Should Choose Realm Over CoreData/SQLite [Internet]. Sebastian Dobrinu. [cited 2017Nov24]. Available from: <https://sebastiandobrinu.com/blog/5-reasons-why-you-should-choose-realm-over-coredata>
22. Bmarrdev. bmarrdev/android-DecoView-charting [Internet]. GitHub. 2016 [cited 2017Nov24]. Available from: <https://github.com/bmarrdev/android-DecoView-charting>
23. PhilJay. PhilJay/MPAndroidChart [Internet]. GitHub. 2017 [cited 2017Nov24]. Available from: <https://github.com/PhilJay/MPAndroidChart>
24. Diogobernardino. diogobernardino/WilliamChart [Internet]. GitHub. 2017 [cited 2017Nov24]. Available from: <https://github.com/diogobernardino/WilliamChart>
25. Charts | Google Developers [Internet]. Google. Google; [cited 2017Nov24]. Available from: <https://developers.google.com/chart/>
26. Platform Overview | Google Fit | Google Developers [Internet]. Google. Google; [cited 2017Nov24]. Available from: <https://developers.google.com/fit/overview>
27. Boran M. Fitness trackers run into resistance over data security concerns [Internet]. The Irish Times. The Irish Times; 2017 [cited 2017Nov24]. Available from: <https://www.irishtimes.com/business/technology/fitness-trackers-run-into-resistance-over-data-security-concerns-1.3119483>
28. Google Maps APIs Terms of Service | Google Maps APIs | Google Developers. Google; [cited 2017Nov24]. Available from: <https://developers.google.com/maps/terms#3-privacy-and-personal-information>
29. Who's on Snapchat: The Changing Age Demographics [Internet]. Likeable Media. [cited 2017Nov24]. Available from: <https://www.likeable.com/blog/2017/3/whos-on-snapchat-the-changing-age-demographics>
30. Nadeau M. General Data Protection Regulation (GDPR) requirements, deadlines and facts [Internet]. CSO Online. CSO; 2017 [cited 2017Nov24]. Available from: <https://www.csoononline.com/article/3202771/data-protection/general-data-protection-regulation-gdpr-requirements-deadlines-and-facts.html>
31. Mezzofiore G. Google removes Maps feature showing calories and mini-cupcakes after huge online backlash [Internet]. Mashable. Mashable; 2017 [cited 2017Nov24]. Available from: http://mashable.com/2017/10/17/google-maps-feature-calories-cupcakes/#C_qGoKAhuiqx

32. Lovejoy B, Ben Lovejoy @benlovejoy Google Maps gets – and then loses – cupcakes in short-lived iOS app update [Internet]. 9to5Mac. 2017 [cited 2017Nov24]. Available from: <https://9to5mac.com/2017/10/17/google-maps-cupcakes/>
33. Software Development Methodologies. [cited 2017Nov24]. Available from: <http://www.itinfo.am/eng/software-development-methodologies/>
34. Intern THC. Top 4 Software Development Methodologies [Internet]. Black Duck Software Blog. [cited 2017Nov24]. Available from: <https://blog.blackducksoftware.com/top-4-software-development-methodologies>
35. Comparing Traditional Systems Analysis and Design with Agile Methodologies [Internet]. The Traditional Waterfall Approach. [cited 2017Nov24]. Available from: <http://www.umsi.edu/~hugheyd/is6840/waterfall.html>
36. What is Agile Software Development? - Definition from WhatIs.com [Internet]. SearchSoftwareQuality. [cited 2017Nov24]. Available from: <http://searchsoftwarequality.techtarget.com/definition/agile-software-development>
37. Kelly Waters, 10 February 2007 | 10 Key Principles of Agile Development. What Is Agile? (10 Key Principles of Agile) [Internet]. 101 Ways. 2017 [cited 2017Nov24]. Available from: <https://www.101ways.com/what-is-agile-10-key-principles/>
38. Scrum Methodology | Plan for best & create better sprints [Internet]. Maxartkiller. [cited 2017Nov24]. Available from: <http://maxartkiller.in/scrum-methodology-plan-for-best-and-create-better-sprints/>
39. What is Extreme Programming (XP)? [Internet]. Agile Alliance. 2017 [cited 2017Nov24]. Available from: [https://www.agilealliance.org/glossary/xp/#q=~\(filters~\(postType~\('post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video\)~tags~\('xp\)\)~searchTerm~'~sort~false~sortDirection~'asc~page~1\)](https://www.agilealliance.org/glossary/xp/#q=~(filters~(postType~('post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~('xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1))
40. Wells D. [Internet]. Extreme Programming Rules. [cited 2017Nov24]. Available from: <http://www.extremeprogramming.org/rules.html>
41. Cohn M. Differences Between Scrum and Extreme Programming [Internet]. Mountain Goat Software. [cited 2017Nov24]. Available from: <https://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming>
42. Android APIs | Google Fit | Google Developers [Internet]. Google. Google; [cited 2018Apr5]. Available from: <https://developers.google.com/fit/android/>
43. Google Maps APIs for Android | Google Developers. Google; [cited 2018Apr5]. Available from: <https://developers.google.com/maps/android/>
44. Stat Trek [Internet]. Interquartile Range: Definition. [cited 2018Apr5]. Available from: <http://stattrek.com/statistics/dictionary.aspx?definition=interquartile%20range>
45. Android: Managing Multiple Notifications [Internet]. java - Android: Managing Multiple Notifications - Stack Overflow. [cited 2018Apr5]. Available from: <https://stackoverflow.com/a/3470456>
46. Google. Google; [cited 2017Nov24]. Available from: <https://developers.google.com/maps/documentation/directions/intro#Waypoints>
47. PhilJay/MPAndroidChart [Internet]. GitHub. [cited 2018Apr5]. Available from: <https://github.com/PhilJay/MPAndroidChart/wiki/Getting-Started>

48. Introduction [Internet]. Material Design. [cited 2018Apr5]. Available from: <https://material.io/guidelines/>
49. Tabs - Components [Internet]. Material Design. [cited 2018Apr5]. Available from: <https://material.io/guidelines/components/tabs.html#>
50. Southorn G. Why are some people better at map reading than others? [Internet]. HERE 360. [cited 2018Apr5]. Available from: <https://360.here.com/2014/10/06/people-better-map-reading-others/>
51. Buttons: Floating Action Button - Components [Internet]. Material Design. [cited 2018Apr5]. Available from: <https://material.io/guidelines/components/buttons-floating-action-button.html>
52. Rotolo P. Example class to Export/Import a Realm database on Android (1/3) [Internet]. Medium. Glucosio Project; 2016 [cited 2018Apr5]. Available from: <https://medium.com/glucosio-project/example-class-to-export-import-a-realm-database-on-android-c429ade2b4ed>
53. Realm Studio: open, edit, and manage your Realm data [Internet]. Edge Sync Mobile Platform & Database Realm. [cited 2018Apr5]. Available from: <https://realm.io/products/realm-studio/>