Mobile App for OpenDay Tour Guides

Final report

Submitted for the BSc/MEng in

Computer Science

July 2020

by

Ionut Albu

Word Count:

Abstract

Since the market of mobile applications is ever-growing, the users have, nowadays, the option to choose and pick from a vast number of alternative applications while the user's expectations regarding the apps are higher than ever.

With context awareness and personalization playing a vital role in any mobile application development, when it comes talking about the functionality of a supporting guide application, this project seeks to demonstrate that map information retrieval and communication functions is what matters the most.

Table of Contents

1 Introduction

While studying at any university for several years, many students, even after 3 years of studies, don't know their way around and rely on staff and fellow students for information and directions inside university's premises.

The goal of the project is to create a tour application that assists the guide with useful information throughout the entire tour and to provide the means of communication between the supervisors and the guide and tools for the administrators to create and update the information available.

The application successfully follows identified main and secondary objectives. The administrators are able to track each groups location at any given time, communicate with the guides, make changes to the information available through the database while the guides are able to start the tour, easily access the objectives information stored in the database and respond to messages sent by the administrators.

1.1 Project background

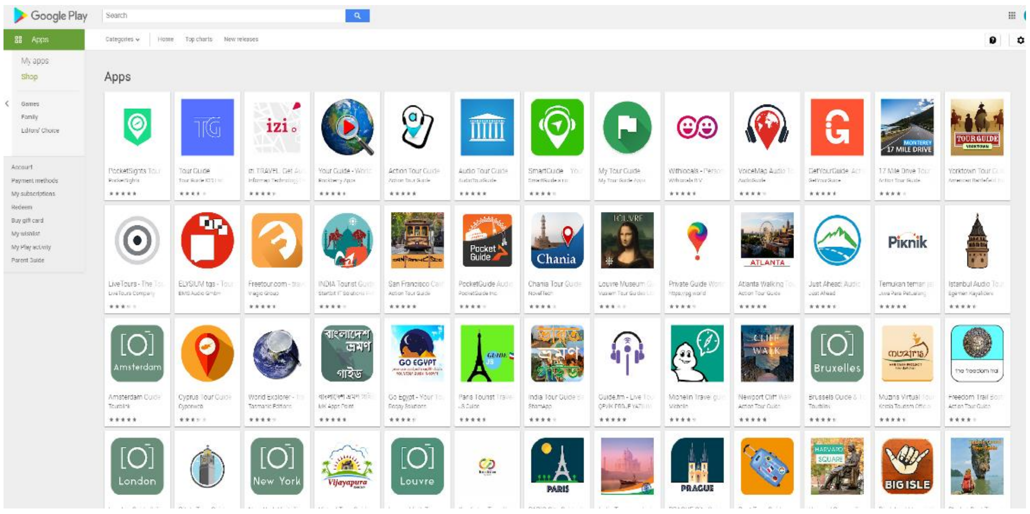
1.1.1 Smart Phone background

Worldwide, the number of smartphone users, today, is higher than three billion with few hundred million growth forecasted in the next years. Smartphones have become a normal part of day-to-day life with more than 94% of people aged between 18 and 29 years old owning a smartphone. 79% of the adults in the United Kingdom, 81% of Americans and a whopping 95% of South Korean adults own a smartphone. Statistically, the average user taps, clicks and swipes their smartphone 2617 times per day while the average time spent doing that is 2 hours and 51 minutes per day (Review42,2020). By becoming more powerful with each passing year (smartphones have long surpassed the capabilities of PCs from 10 years ago) and by being so small compared to other electronic devices, smartphones are starting to replace laptop and desktop usage for many people.

The multitude of tour guide applications on the google play store, while many of them are robust, suffer from similar lack of features and problems:

- Assuming data connection will be always on
- Assuming high speed access to the network
- User Interface design being too small or too packed
- Poor data sync
- Too many permissions requests

- Lack of human element



(Google Play,2020) *Fig 1*

1.2 Aims and Objectives

The application will consist of a system that aims to helps the guide with specific information about timeframes, information about the objectives, means for the administrators to alter or update objectives information and means of communication between administrators and guides.

1.2.1 Primary Objectives

1.2.1.1 Full development

The main objective of the project is to develop an android application that helps the guide with useful information and a communication channel.

1.2.1.2 The Graphical User Interface

The GUI (Graphical User Interface) should be very clear and very simple to use. It would be designed in such a way that the component elements will be easy to access and understand. All sections of the GUI will follow the same path and style so that the user will not be confused and frustrated when transitioning from one part of the application to another.

1.2.1.3 Registration, Authentication and Authorisation

The application will allow users to register an account which will be used to log into the application. There will be two kinds of users, User and Admin and the roles will be set by an administrator, manually, inside the database. The authentication system will be an email/password one and will be implemented by using an already existing, open source solution, firebase. On authentication successful the system will check if the user is an Admin or a regular User and will redirect towards the appropriate section of the application.

1.2.1.4 The GPS tracking service

The GPS tracking service will be implemented by using Google Maps APIs which is a tool that is used to check-in, search and create map functions, live display of data in sync with user's actual location and plan routes. Since the API requires real time location verification, the user, if the phone's location service is not enabled, will be asked to turn on the service and give permission for location tracking.

1.2.1.5 The Database

The database will be a real time database which will hold information about users, objectives and will be used in order to create a real time communication channel between the administrators/supervisors and the guides.

1.2.1.6 The communication channels

The communication channel will provide a hands-on solution for users to be able to exchange information in real time. It will mainly be providing administrators/ supervisors with the means of making announcements and send urgent messages when needed. It will use the database to store the messages.

1.2.2 Secondary Objectives

The application will incorporate tools for the administrators to be able to Update/ Create new objectives content and user management.

1.3 Research Question

Being a student is a wonderful thing nowadays but what happens when, after a couple of years studying at a university, a student finds himself in the middle of the campus looking for a place or location that he has never been to or heard of. Frustration can take place. By making sure that the university provides user friendly, precise and information-filled open days tours the frustration can be avoided. The problem is human error. Our brain is a wonderful "machine" but can sometimes fail us. People forget. All universities have open day tours but not all of them have a supporting app that is filled with useful information that is ready to be used when needed while maintaining a reliable communication means with their guides.

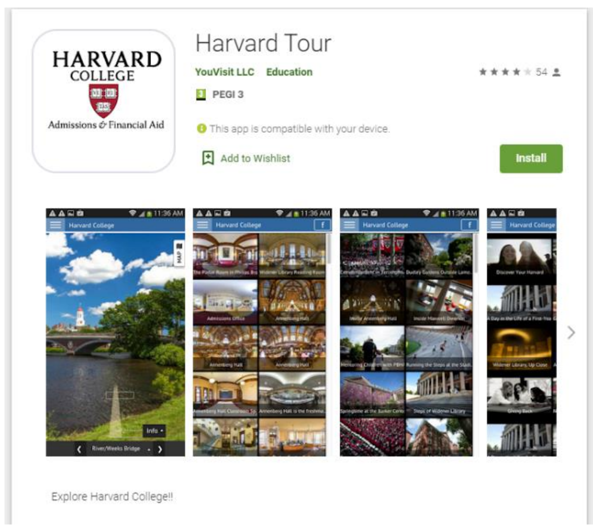
2 Literature Review

While many universities have tour guide mobile applications, most of them are self-tour, meaning that you install the application that follows a set script without the support of a real person who can answer any of the questions on the spot. Research is needed to be done in order to get a better understanding of other applications that exist on the market at present time.

2.1 Applications Examples

2.1.1 Harvard Tour and Yale Virtual Campus Tour

Both are applications that enables prospective students to navigate and learn about The Harvard College. The key features of the application are a self-guided tour of the campus map that marks the important objectives, descriptions of the landmarks and buildings, available campus photos and videos. The applications are similar, being developed by the same company and while they do have interesting features, they lack one common key element, the human element.



HARVARD COLLEGE
Admissions & Financial Aid
Harvard Tour
YouVisit LLC
Education
PEGI 3
This app is compatible with your device.
Add to Wishlist
Install
Explore Harvard College!!

Fig 3: Google Play Store

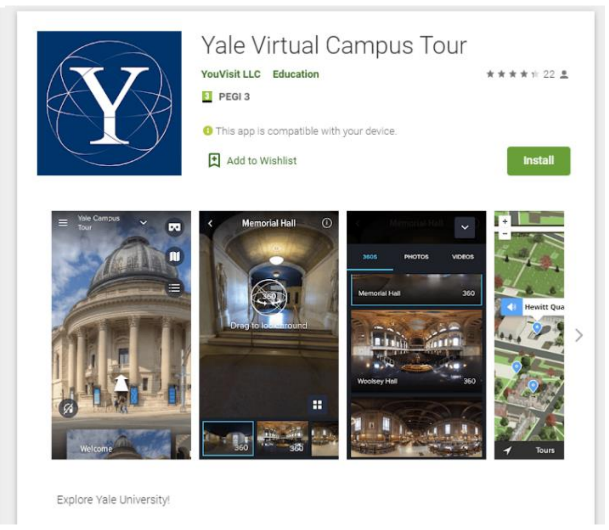
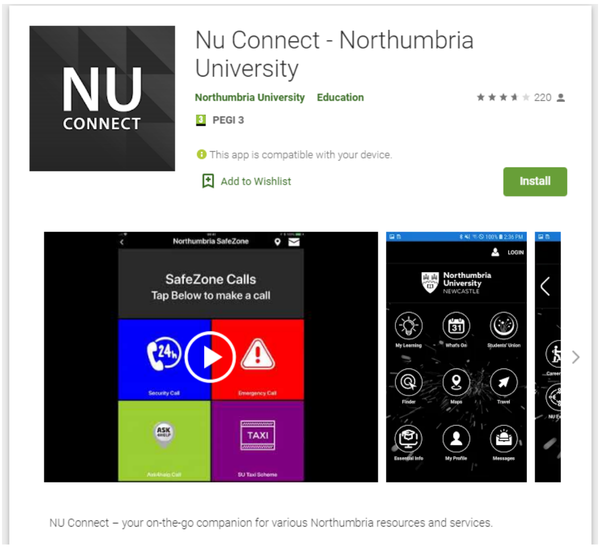


Fig 4: Google Play Store

2.1.2 NU Connect – Northumbria University

Nu Connect is an application that tries to combine a mobile self-guided tour of the university with other functional features such as building a personal tour schedule, ability for students to be able to access class and exam timetables, daily planner, a search function for points of interest within the university's campus and available campus parking. Like the previous two apps, this one is very well built, but, behind the multitude of available features there is the lack of human element.



NU Connect – your on-the-go companion for various Northumbria resources and services.

Fig 2: Google Play Store

2.2 Platforms

A very broad and simplified definition of a development platform is a big piece of software such as an environment, database or an OS (operating system - a type of software that acts as a communication middle man between a computer's hardware and software and manages all aspects of a computer), which enables smaller applications to run.

2.2.1 Android

Android is an operating system developed by Google and has the biggest market share, at over 74% (Statista, 2019), of the mobile development platforms. Having such a big market share implies that the applications developed on this platform has a much bigger chance of being discovered, have a considerably bigger scope and higher profitability.

Android is an OS which is based on Linux Kernel, an operating system itself, found most commonly on servers. Because Android is an OS designed with mobile in mind and while it has suffered many changes, it is not just another version of Linux but much more than that.

While Android might look or feel different on different devices, Samsung versus Motorola for example, the core is always the same. Most Android device manufacturers usually have created custom overlay designs for their devices in order to provide an experience unique to the chosen device manufacturer.

Application development for Android is done through Google's IDE named Android Studio and while the creating process of the applications is free, the publishing process involves the payment of a $25 one time per application fee.

2.2.2 Apple OS (iOS)

IOS is an operating system developed by Apple that powers iPhones, the iPod touch and until 2019 it was used on the iPad. Before 2010 it was called iPhone OS but, with the introduction of the iPad in 2010, since iPhones were not the only devices that used the OS, it was renamed as iOS.

iOS's market share figure sits at around 24% and has been growing over the last year. Unlike its counterpart, Android, iOS is closed source and Apple tries to heavily control it making development for iOS devices more difficult. The application must be approved by Apple and this, sometimes, can be a very lengthy process, taking up to several months. In order to have access to the development tools, the developer must pay a $99 per year fee and they can only operate on an Apple platform with zero options for Windows or Linux based versions.

Due to the restrictive, more expensive and time-consuming approach by Apple, the development of iOS applications is directly reflected in the market share margin. (Apple Developer, 2020)

2.2.3 Windows Mobile

Windows mobile is an operating system developed by Microsoft. It has been discontinued so the platform would make a very poor choice when developing applications. Originating back in 1996, it made its first appearance in the year 2000. In 2003 it was renamed from Pocket PC 2000 to Windows Mobile. The main aim of the operating system was the enterprise and business market. While the last major update dates back to June 2015, all support for the platform ended in December 2019.

With the death of Windows Mobile, developers have only two real options for mobile applications, Android or iOS.

2.2.4 Verdict for platform

With all points considered, since iOS is a closed source software with a lower market share and Windows Mobile being practically dead, Android was finally chosen as the platform of development.

2.3 Available IDEs

An IDE (Integrated Development Environment) is a tool that contains and consolidates a certain set of aspects when it comes to writing a software.

Important features like syntax highlighting, autocomplete, building executables, debugging and finally developing on your own machine.

Highlighting represents the ability of the IDEs to provide visual cues for words with special meaning like classes, types of variables, etc. thus making the code easier to read and understand the different elements.

Autocomplete represents the ability to anticipate what the developer wants to do next. For example, “Console.WriteLine()” is a command often used which displays chosen information to the console. When the developer types “Con”, the autocomplete feature displays “Console” and the options can be easily chosen and entered. This saves time by saving keystrokes. The time can be used for focusing on logic rather than typing.

Regarding executables building, before any program can run, the source code must be transformed into an executable by the compiler. The IDEs provide automated building therefore compiling and executing the source code is abstracted away.

Debugging refers to the process of deliberately inspecting the code. Since no developer can avoid errors and bugs this is a very important feature of an IDE. It provides hints that lead to prevention of errors before attempting compilation.

The biggest benefit of making use of an IDE is the ability to write code and run programs on your own machine.

2.3.1 Android Studio

Android Studio is an all-round IDE developed by Google which uses Java as the main programming language with Kotlin as a second language. It is 100% free and it is licensed under Apache 2 (Android Developers, 2020) as an open source. It supports source control (GitHub) and uses XML for User Interface design. It is the chosen IDE for the project and its top features include:

- Instant application run which consists of intelligently understanding the transmutations executed within the application and delivering without the need of rebuilding.
- Visual Layout Editor is a feature that facilitates the layout building process by adding certain attributes either by drag and drop or by code writing. Any adjustment is done dynamically while the preview can be easily seen on the editor screen.

- Fast Emulator replicates and emulates an android phone by providing the developer with real time experience of the application thus making the development cycle more efficient.
- Intelligent Code Editor that helps the developer by completing and analysing code in advance.
- Activities Code Templates provides effective solutions in building the application.
- Build the Application for all Devices feature consists of the IDE's ability to build for all screen sizes and emulate some of the hardware features.
- Firebase Connection helps with the building of scalable application infrastructure.

2.3.2 Xamarin

Xamarin is an open source IDE that uses C# as programming language and facilitates the creation of scalable, feature packed and secure applications. It is available for Windows (using Visual Studio) and Apple OS (using Xamarin Studio). Main features of Xamarin are:

- Just-in-time compilation that consists of compiling the code into and intermediate language and, at run time, into assembly code.
- .NET Framework
- Xamarin.Forms consists of 100% sharing of the User Interface code between all platforms.
- Xamarin.Mac enables developers to create code for MacOS

The biggest con of using Xamarin is the price of pushing an application to the market. Coding with Xamarin is free and the requirement is only a community edition of Visual Studio but the $300 subscription plan when trying to publish the applications is a major drawback and puts many developers off.

2.3.3 Verdict for IDE

Although Xamarin uses C# as native language, Android Studio's Java was the chosen development language. The fact that C# is very similar in syntax to Java will make implementation almost as easy. The main decision factor in choosing Android Studio was the high price of publishing of Xamarin.

2.4 Available Programming Languages

When developing android applications, the first step is picking a language. While the differences between the multitude of available programming languages can be quite complex, what matters the most is the features and goals of the platform the language is attached to. Choosing a new platform for the application, one that the developer isn`t familiar with, most of the times includes overcoming the difficulties that are encountered when learning a new programming language. A developer must choose wisely before the development process has started, must weigh in the benefits and obstacles of using a new programming language and decide if it is viable and worth it. The biggest concern, when choosing the right tools, is the resource availability. The less a platform or language is used the less the number of available resources. The less developers using a language, the less solutions for a potential problem are going to be available online and while security is a major concern on any platforms, including Android, the most obvious choice is to go with a programming language that, not only the developer is comfortable with but is used and exploited widely, on a as large as possible scale. The logical thing to do is to pick a platform that has been well documented and that has a plethora of ready-available resources while choosing the native language for that said platform could mean saving large amounts of time while trying to solve problems.

2.4.1 Java

When talking about Android applications development, Java, along with Kotlin, stands as one of the two official options, supported by Google.

Regarding IDEs that use Java, Android Studio is the top developer choice.

Java was introduced by Sun Microsystems in 1995. Since then, it has become so familiar that, even though Google has expressed its preference towards Kotlin, developers chose to stick with it.

It is an Object-Oriented programming language that isn’t the best option as a first language, especially when Android SDK is added to the mix. While it is nowhere close to a bad language choice, most of the inconveniences that come with using Java encourages developers towards writing a clean code. It is used mainly for developing applications for desktop, Android and back-end web frameworks.

2.4.2 Kotlin

Kotlin is a cross-platform language that has been one of Google’s Android development official languages since 2017. It has since become the preferred Google language when it comes to applications development but since many developers are already heavily invested into Java there are many who have chosen not to switch to it.

Like Java, Kotlin runs on JVM (Java Virtual Machine) and both can inter-operate while causing zero slow down or file size increase.

A notable difference between Kotlin and Java is that Kotlin removes the necessity to end the lines with a semi-colon and the "nasty" null pointer exception feature of Java.

All in all, Kotlin is a much simpler language to take on and it is more suited for beginner coders.

2.4.3 C/C++

While Android Studio offers support for C and C++ through the usage of Android NDK, it is not recommended for beginners. The code written does not run on the JVM but on the device directly. It is mainly used for graphic intensive applications such as 3D games and offers more control on things such as memory allocation, enabling the developer to squeeze the Android device of extra performance.

The language is less flexible, introduces a new range of bugs and is a lot harder to setup.

2.4.4 C#

Developed by Microsoft, it combines the ease of use of Visual Basic with the power of C++ and the syntax is very similar to Java. Knowing how to use either Java or C# makes the switch extra simple. It has a garbage collector, just like Java, which means the developer does not have to worry about free memory and leaks, it is done automatically.

2.4.5 Python

Being a general purpose and high-level language, Python is used for a wide range of applications, from desktop to websites. While it is not natively supported by Android, using specific tools enable the conversion of Python application into Android packages.

Being a high-level language with a basic, easy to understand syntax that looks like English language, makes Python the best choice for beginners and amateur programmers.

2.4.6 Verdict for programming language

Since Android Studio was previously chosen as the preferred IDE, Java or Kotlin would only come in as the only true choices. Java, being similar to C# will make the transition as smooth as possible with minimum amount of research needed, was the final decision.

2.5 The Database

"A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database." (Oracle, 2020)

Since 1960s, when they were originally introduced, databases have evolved from hierarchical and network based to relational databases in 1980s and object-oriented databases in 1990s. When it comes to how data is utilized, managed, collected and stored, nowadays, self-driving databases and cloud databases are making major breakthroughs.

There are several types of databases:

- Relational databases organise data in sets of tables with columns and rows and provide the most flexible and efficient way to access information which has been structured.
- Object-oriented databases, just like object-oriented programming, make use of objects to represent data.
- Distributed databases are databases that have their information scattered multiple machines inside the same network or even over several networks.
- Data warehouses are centralised databases optimised for fast analysis and query.
- NoSQL databases also known as nonrelational, allows for manipulation of semi structured or even unstructured information.
- Cloud Databases reside on a cloud computing platform and consist of both structured and unstructured data.
- Document/JSON databases are a modern way to store data in a JSON format.
- Self-driving databases are the newest type of databases that make use of machine learning to automate database many activities that would have been done, otherwise, by an administrator.

2.5.1 Firebase Realtime Database

Firebase Realtime is a Google Cloud backend-as-a-service NoSQL database that uses JSON format to transfer the data to and from the application (Firebase Realtime Database, 2020). It eliminates the need to manage the servers and to write APIs (functions that allow the access of features or data of other services or applications). It replaces the use of HTTP (Hypertext Transfer Protocol) with WebSockets (communications protocols that use a single TCP connection while providing full-duplex channels) which makes the data transfer as fast as the network can support it.

The benefits of using Firebase are simple to use APIs, ease of setup, cross platform support, free to start with and full authentication support. Several things that need to be considered are, in the long run, the user base growth which can lead to expenses, the limit on the total requests per minute, the JSON format which can be hard to understand and since it is a NoSQL database, traditional data models do not apply.

2.5.2 SQLite

SQLite is a very light, open source database system that is supported on most platforms available on today's market which makes it, in 2020, the most used database engine world-wide. It is built into most computers and into all modern mobile phones. Compared to the Firebase database, it needs more backend setup and has zero authentication support.

The biggest benefit of using SQLite is that it is an open source freeware making it a perfect fit for any entry level person that wants to learn relational databases. Other benefits include no configuration needed because of its server-less installation. Disk space usage is kept to the minimum, under 1 Mb for file size and if an android application needs an internal database it is most suitable. SQLite can perform all task SQL can but lacks some advanced features and security.

The drawbacks of using SQLite is that it is not scalable at a huge level therefore not suitable for Enterprise databases that require high protection and security and it only allows a single write at a time making multiple users connected and working at the same time impossible.

2.5.3 Parse

Parse is very similar in some respects to Firebase since it is an open source, online backend-as-a-service framework with its own authentication system. It was initially developed by Facebook and it is constantly developed by an active community of developers.

The main benefits of Parse are freedom of usage since it is open source, simplicity of use with simple APIs, great documentation with superb tutorials and multi-platform support while the drawbacks consist of limited customization, Facebook privacy concerns and low access to backend source code.

2.5.4 Custom Created Database

Being self-created and self-hosted it would bring the major benefit of having total control over the entire database, eliminating the limitations that other options have. It would allow for any programming language usage and it would be highly customizable. On the other side, the developer would need an excellent knowledge of backend development, the setup time could be very high, stability problems due to eventual mistakes and it would involve costs for either hosting or self-hosting over a 24/7 period.

2.6 Authentication System

Authentication is the process of establishing if someone's or something's identity by comparing the provided credentials to the existing stored information.

During registration, a user provides his/her/its information which is stored securely in a local or remote location in order to avoid unauthorised use and data leaks. After registration, the user can use the information previously provided in order to log into the application.

2.6.1 Firebase Authentication System

Firebase Authentication is Google's own authentication service that provides easy to use SDKs, backend services and ready-made libraries. It is compatible with a very large range of languages, OS and services such as Android, Web, Unity, iOS and C++. (Firebase Authentication, 2020)

While it supports OAuth2 for Facebook, Twitter, Google and GitHub, what is of interest is the built-in email/password authentication that integrates directly into the Firebase Database.

It is completely free, works excellent with all other Firebase services, it is easy to integrate and manage, has excellent documentation and has multi-platform support.

Since Android Studio and Firebase are both Google's services it is only logical that they will fit together extremely well and integration between them will be very simple.

2.6.2 Parse Authentication

Parse Authentication, initially developed by Facebook, is an authentication system that also supports multiple sign in methods and platforms. It uses OAuth over Apple, Twitter, GitHub and Google.

Similar to the Firebase Authentication system, Parse does not need a lot of backend setup, it is free and simple to use but since it does not have an excellent platform to start with it does need a little more backend knowledge and knowledge of Node.JS (lightweight and efficient event-driven platform built on Chrome's JavaScript runtime) while not being very flexible.

2.6.3 Verdict for Database and Authentication Services

Although Parse has visibly better Authentication support and supports more sign in methods, since email/password is the chosen authentication method, because Firebase Database and Firebase Authentication are both provided by the same developer, they interconnect and work so well together that choosing any other pair would not seem as logical.

Taking in the account the scope of the project, Firebase provides everything is needed and since it integrates so well with Android Studio, the chosen IDE, the time it will take in order to set them up will be significantly lower that using a custom-made database.

2.7 The APIs

API stands for Application Programming Interface and is, at base, a resource that allows communication between applications.

In order for the project to accomplish some of the objectives, like the GPS tracking service, an API service that has the needed features would have to be selected.

2.7.1 Google Maps API

As previously stated, Google Maps API is a very robust tool that combines excellent features while offering a free $200 per month credit when using Maps, Places and Routes while the usage of SDKs is completely free with no usage limits.

Since the creation of a digital map is already provided by Google, Google Maps API is an excellent choice in the process of creation of a geographic information system saving both effort and time and allowing the developer to focus on other aspects such as the data and information. Only basic knowledge of JavaScript and HTML is needed in order to understand Google Maps API.

Several benefits of using Google Maps include a simplified storage system maps, the creation of new maps is unnecessary, fast data change of maps, it can be accessed anywhere if the user has an internet connection and the using system will be light.

2.7.2 OpenLayers

OpenLayers is a free API used for displaying dynamic maps while drawing tiles from multiple sources. One of the sources is OpenStreetMap which is an API that is used for raw geodata.

Although it is a free to use tool, it does allow some nice flexibility in the form of allowing the developer to drop markers and render vector layers on the map at will.

2.7.3 TomTom

TomTom has a very good reputation, especially when it comes to satellite navigation. While the service is payed, it does allow up to 2500 free daily transactions which is more that what Google offers. It offers more services than OpenLayers like tiled and vector maps, location search, markers, traffic analysis and route finding.

2.7.4 MapBox

MapBox is an open source mapping choice of both Snapchat and Facebook. It is used to create solutions for problems with maps, data and spatial analysis, drawing data from both open and proprietary sources. Like TomTom and Google, it offers maps, navigation, searches and, most important, custom map features which makes MapBox popular.

Price wise, MapBox sits somewhere between Google and TomTom with 50000 free monthly requests.

2.7.5 Verdict for geolocation APIs

Overall, Google Maps API was selected. Considering the scope of the project and the extensive user familiarity with this service and while offering reliable data from over 200 countries and territories, Google offers everything that is needed in order to achieve proposed objectives.

2.7.6 Firechat

Firechat is an open-source, real-time chat widget built on Firebase. It offers fully secure multi-user, multi-room chat with flexible authentication, moderator features, user presence and search, private messaging, chat invitations, and more. (Firebase, 2020)

Since it uses the Firebase Realtime Database as a backend, it requires not developer coding to be done on the server side. Other benefits include zero cost as long as the userbase doesn`t exceed a certain amount, making it great for starting developers, excellent support and documentation, very good security level due to using Firebase Authentication system and extreme speed due to using the Realtime database as backend.

2.7.7 Firebase

While Firebase is a tool and not a chat solution, it is a great choice in developing the communication channel objective. Codelabs incorporates such an application and this should give clear hints to the developers.

Although it easy to build and it is an excellent showcase of capabilities when it comes to real time messaging and notifications, it doesn`t scale well for large number of users due to its dependability on Firebase backend.

2.7.8 Mesibo

Mesibo is a high-end chat applications technology which enables real time messaging, voice and video calling. Its compatibility spreads towards iOS, Android and Web Applications, it supports multiple languages, huge scalability potential, end to end encryption and provides analytics in real time.

2.7.9 Applozic

Applozic prides itself with its frontend and backend infrastructure. It provides solutions for broadcasting, e-commerce and healthcare with integrated chat based and Chat Bot support while taking customer service to another level.

2.7.10 Verdict of communication

While Mesibo and Appolozic are considered the more complete solutions for Enterprise level applications, taking into account the project’s scope and the fact that Firebase is used both in the development of the database and as an authentication method, going that far is not necessary and the more appropriate choice is Firebase.

2.8 Potential Problems

2.8.1 Not enough knowledge of Java programming language

While C# has been studied and used throughout the entire duration of the studies, Java, as practice, has never been used. This would imply spending extra time trying to understand and learn how Java works, what are Java’s strengths and what are its weaknesses. Looking from distance Java and C# are largely identical but when looking closely subtle and significant differences can be found. They are both class-based, object-oriented programming languages and the main difference stands with the libraries they each use.

3 Technical Development and Implementation

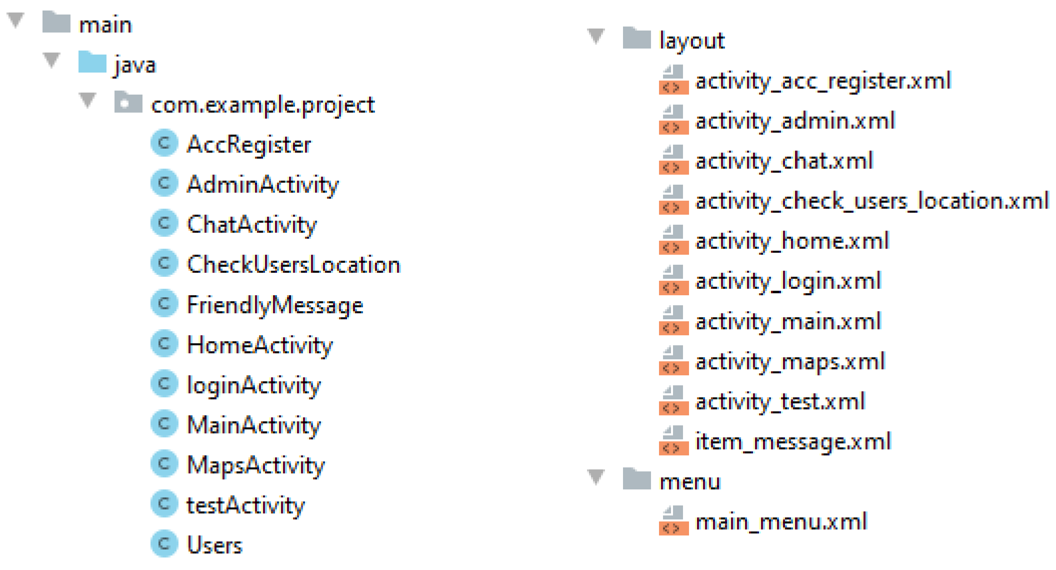
3.1 Coding Convention

Adhering to correct coding techniques has major benefits when it comes to the readability and the understanding of the project. It allows other developers, that are also familiar with the conventions, to easily read, understand and follow the code. There is practically no downside especially when it involves sizable projects.

Most Java conventions were used doing the development process of this application thus ensuring the project is future proof regarding this aspect. Adhering to conventions not only helps the current developers but the future ones also if the application is to ever be expanded.

Examples of conventions:

- Catching exceptions
- Variable names
- Comments attached to every major method
- Class and Layout separate packaging (**Fig 24)**



```
main
  java
    com.example.project
      AccRegister
      AdminActivity
      ChatActivity
      CheckUsersLocation
      FriendlyMessage
      HomeActivity
      loginActivity
      MainActivity
      MapsActivity
      testActivity
      Users
```

```
layout
  activity_acc_register.xml
  activity_admin.xml
  activity_chat.xml
  activity_check_users_location.xml
  activity_home.xml
  activity_login.xml
  activity_main.xml
  activity_maps.xml
  activity_test.xml
  item_message.xml
menu
  main_menu.xml
```

Fig 24: Class/Layout

3.2 UI Design

The UI design (**Fig 24**) was created in such a way that is adheres to the standards of Android. It is focused on simplicity and minimalism and the average user should have no difficulty following the menus and using the application. It is a pure digital design meaning that a paper based one is non-existent. The reason behind this is the time limitation.

The application simply starts with a splash screen and into the login page. The login page has two text fields, one for email and one for password and contains two buttons, one for log in and one for account creation. From there, depending on the type of user logging in, the application splits in two parts: The Admin UI and the Guide UI.

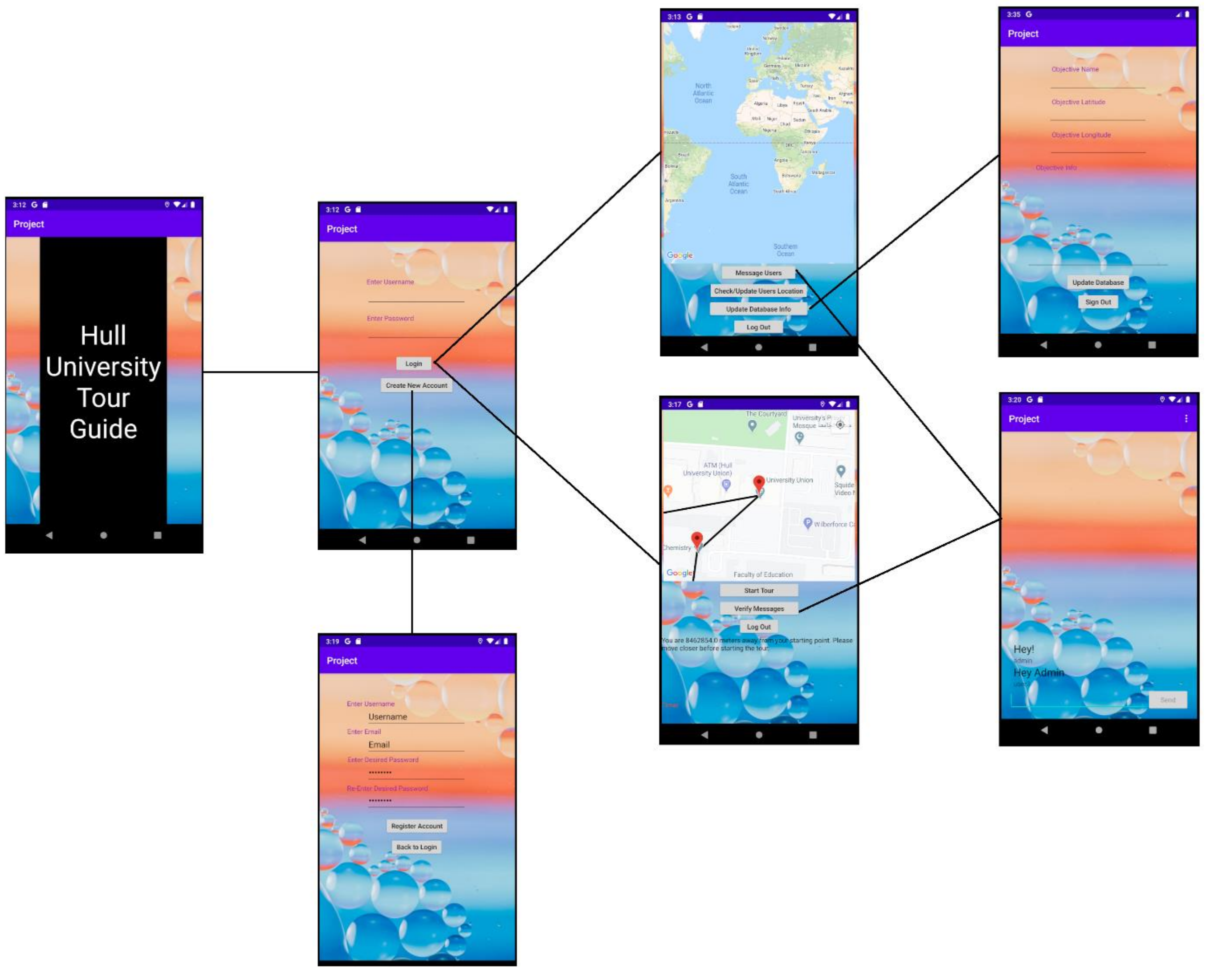


Fig 24: UI

The Admin UI is a simple one with a general google maps layout without any kind of markers. This will fill and focus on the markers only if a guide is logged in and the admin wants to check the location by pressing the right button. From here, the admin has the option to either go to a chat window to send messages or to a window that enables him to update the objectives in the database.

The Guide UI is also very simplistic but with more elements that the Admin UI. It has a google map layout filled with objective markers which are ready to use and filled with information from the database. All he must do is click on a marker and the text window, which is another element of the UI, will fill with information about the selected objective. The UI also contains a timer display that informs the guide how much time he needs to spend at each objective and how much total time it is left. If certain conditions are met, once the tour and timer have started, additional information will be displayed alerting the guide when he needs to move onto the next objective (explained in the next sections). The last element on this UI is the check messages option which takes the guide to a chat window in order to communicate with the admins or other users.

The Chat UI is basically a Text Layout with a text input section and a “Send” button. When a message is sent or received, the layout populates with the message signed by the sender. It might look as simplistic as possible to the naked eye, but beautiful things happen behind the scenes with the help of Firebase and a few listener methods (explained further down).

The last available option is to create a new account. This is accessed from the login page by pressing the create account button. On this page there are three text input elements for the username, the email and the password and a button that takes the user back to the login page. The new user has the option to skip setting up a username, but the email and password fields are mandatory. If the user skipped the username field, not to worry, the application, at the right time, will pick up the email that has been set up and will set the first part of the email as replacement for the username.

This was the final design and all features are included and shown in Fig 24.

3.3 System Design

3.2.1 Class Diagram

The following class diagram (**Fig 5**) loosely describes the base design of the project. It shows how each class relates to other classes before any implementation took place.

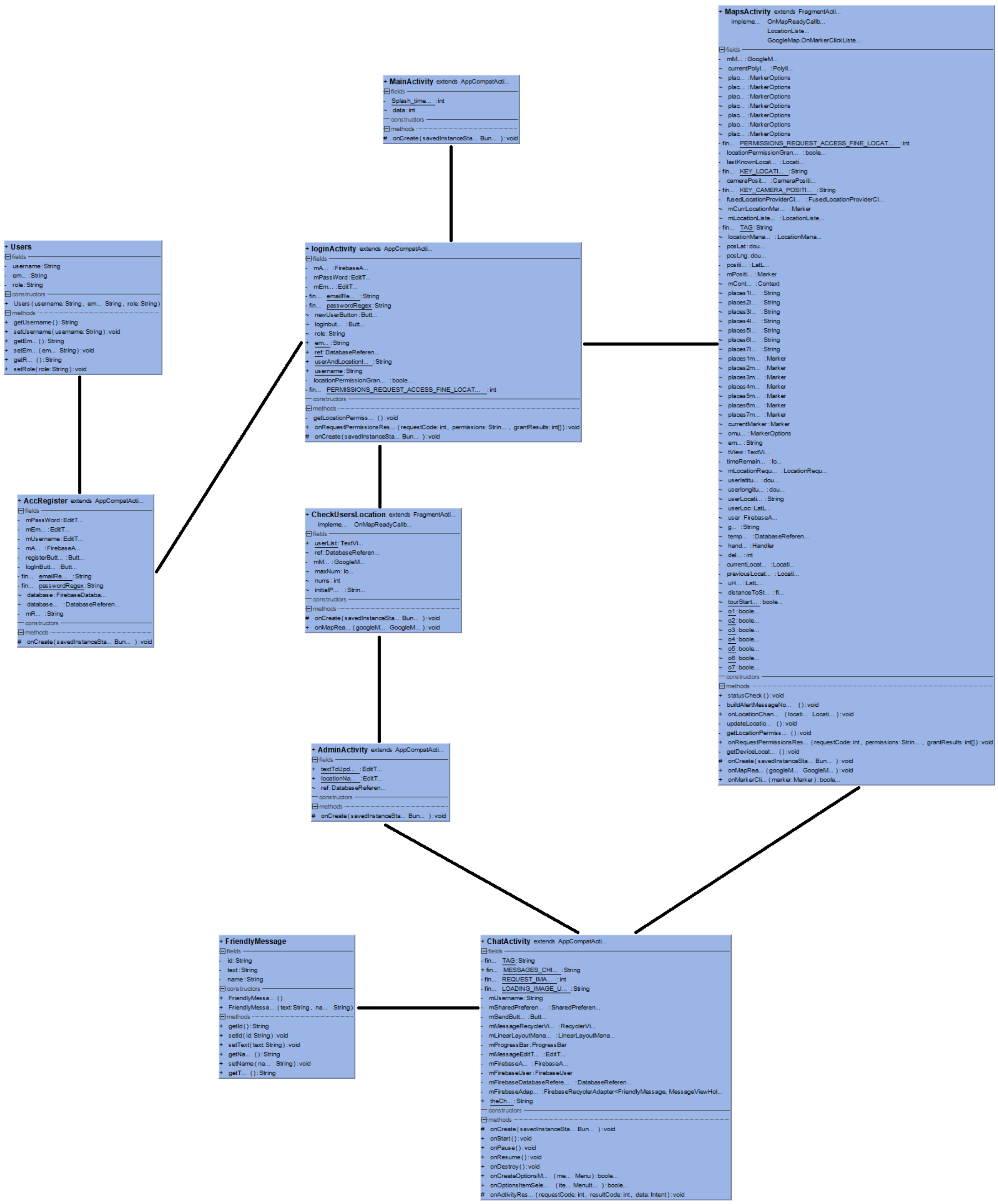

MainActivity extends AppCompatActi...
Users
loginActivity extends AppCompatActi...
MapsActivity extends FragmentActi...
AccRegister extends AppCompatActi...
CheckUsersLocation extends FragmentActi...
AdminActivity extends AppCompatActi...
FriendlyMessage
ChatActivity extends AppCompatActi...

Fig 5: Class diagram – Zoom for clarity

The MainActivity class is basically a splash screen that lasts for 3 seconds and starts the LoginActivity class. From LoginActivity class the other activities are accessed dependin on the user's authorisation. If a user has Admin authorisation, it will start the CheckUsersLocation class while if a user is just a guide user it will start the MapsActivity class.

3.3.1 Use Case Diagram

A Use Case Diagram is a diagram that lists the steps needed to be taken in order to reach a specific goal. The following diagram (**Fig 6**) contains two actors. The first actor is the User while the second actor is the Firebase Realtime Database.

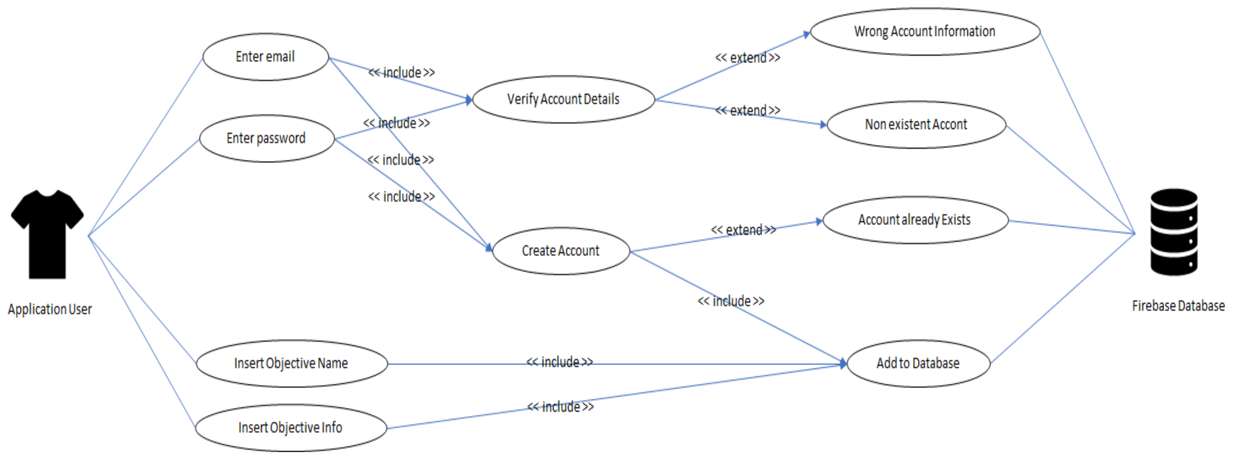


Fig 6: Use Case diagram – Zoom for clarity

The triggers in this use case diagram are the "Log In" button, the "Create New Account" button and "Update Database" button.

The goals in this diagram are to Log in with an existing account or Create a new account and Update an objective or Add a new objective to the database.

The alternatives for this use case diagram are:

- If the account information does not exist in the database it will display a message informing about failed log in attempt.
- If the account information is wrong, it will display a message informing about failed log in attempt.
- If trying to register an already existing account, it will display a message informing about an already existing account with that email.
  The preconditions for this use case diagram are:
- A connection to the internet that must be secure.
- Correct and in the right format information has be entered when trying to Log in or Create a new account.

The first purpose of the Use Case Diagram is to facilitate the understanding of how the actors, which can not be fully controlled, interact with each other. The second purpose of the diagram is to establish what would be needed to be thought about during the implementation process in order to avoid and deal, pre-emptively, with as many problems and situations as possible.

3.3.2 Application Layout

The application layout, which is represented in **Fig 7**, displays each fragment of the application and the links between the fragments. One example would be when if an Admin would like to check the location of existing tour groups, he would press the Check/Update User Location button located in CheckUserLocation class which will trigger the application to check the database for logged in User guides, will extract the location information from the database and will create static markers on the Admin’s map showing the exact location of each group. Of course, successfully completing the task implies that a User guide is logged into the application and the database successfully stored the coordinates and username of the guide. If there is not User guide logged into the application, it will simply return a message informing the Admin that no users are logged in at the moment.

The design of the application was thought so when a User or an Admin would press the back button on their phone, if the fragment that follows is not the LogIn fragment, it will not take the users to the main page of the application. The reason behind this decision is the fact that the application uses a different activity for the chat functionality. When an user is in the middle of an activity and wants to verify the chat channel for new messages or simply wants to send a new message, he has to open the chat activity first by clicking either “Verify Message” or “Message Users”, depending on the authorisation, and, very important, must not lose the work he was previously doing. Clicking the back button on the phone from the chat activity will simply return the user to what he was previously doing while saving the state of his previous activity or, for Guide users, even running in background.

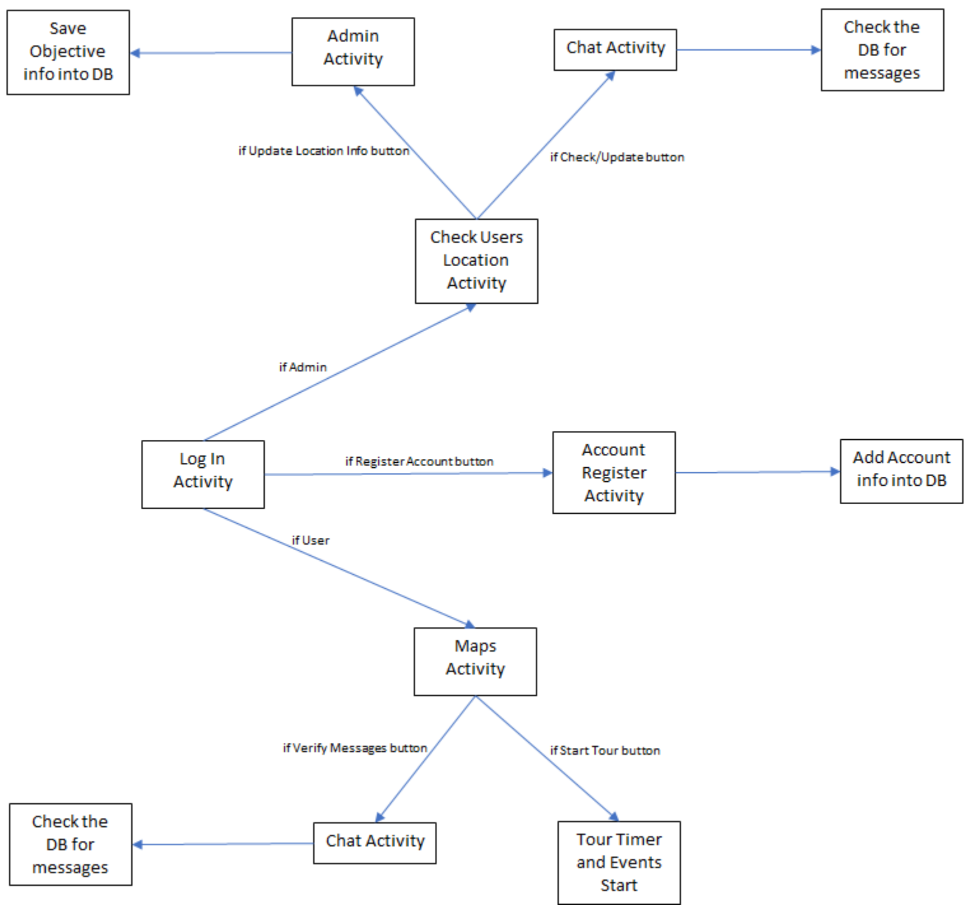

Save Objective info into DB
Admin Activity
Chat Activity
Check the DB for messages
if Update Location Info button
if Check/Update button
Check Users Location Activity
if Admin
Log In Activity
if Register Account button
Account Register Activity
Add Account info into DB
if User
Maps Activity
if Verify Messages button
if Start Tour button
Check the DB for messages
Chat Activity
Tour Timer and Events Start

Fig 7: Use Case diagram

3.3.3 Database JSON

Firstly, since the stored data format of the Firebase Realtime database is represented by a JSON and the default data format would not necessarily be of use, some changes needed to be made in order to maintain a good application functionality. Secondly, Firebase functions on rules. The only and most important rule that needed to be adjusted is the read and write permissions to the database. The default rule does not allow those permissions and needed to be changed from:

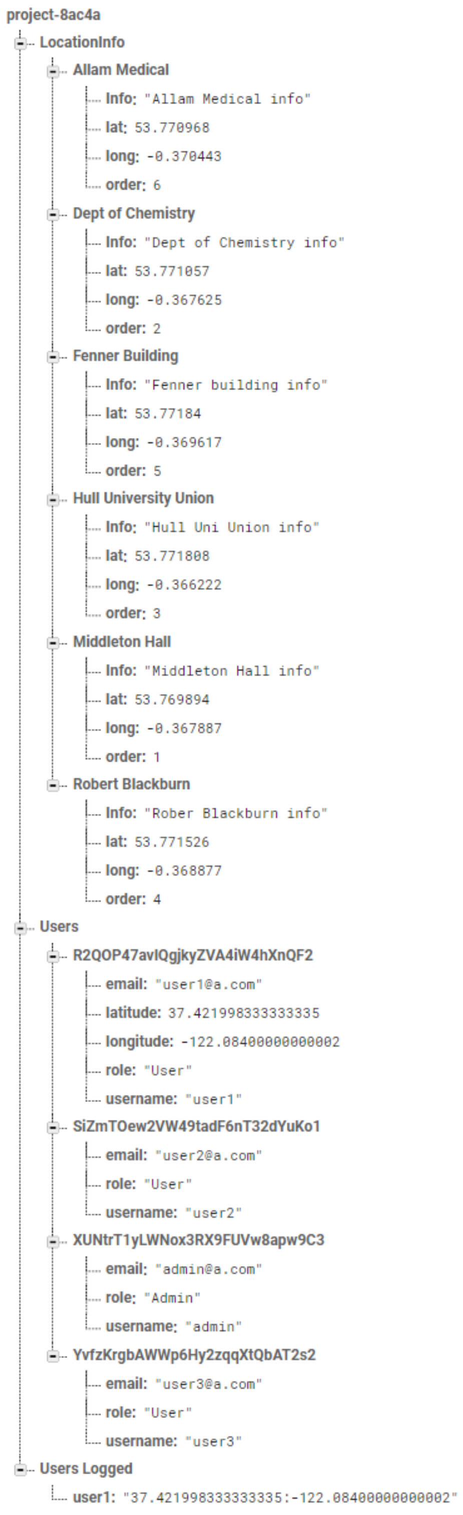
```
{
"rules": {
".read": false,
".write": false
}
}
```

To:

```
{
"rules": {
".read": true,
".write": true
}
```

Having changed the rules into the second format means that anyone would have read and write access to the database. Due to time constraints it will be kept like this.

The JSON format in which data is kept in the database is different from one entity to the other depending on functionality (**Fig 8**).



```
project-8ac4a
  LocationInfo
    Allam Medical
      Info: "Allam Medical info"
      lat: 53.770968
      long: -0.370443
      order: 6
    Dept of Chemistry
      Info: "Dept of Chemistry info"
      lat: 53.771057
      long: -0.367625
      order: 2
    Fenner Building
      Info: "Fenner building info"
      lat: 53.77184
      long: -0.369617
      order: 5
    Hull University Union
      Info: "Hull Uni Union info"
      lat: 53.771808
      long: -0.366222
      order: 3
    Middleton Hall
      Info: "Middleton Hall info"
      lat: 53.769894
      long: -0.367887
      order: 1
    Robert Blackburn
      Info: "Rober Blackburn info"
      lat: 53.771526
      long: -0.368877
      order: 4
  Users
    R2QOP47avlQgjkyZVA4iW4hXnQF2
      email: "user1@a.com"
      latitude: 37.421998333333335
      longitude: -122.08400000000002
      role: "User"
      username: "user1"
    SiZmTOew2VW49tadF6nT32dYuKo1
      email: "user2@a.com"
      role: "User"
      username: "user2"
    XUNtrT1yLWNox3RX9FUVw8apw9C3
      email: "admin@a.com"
      role: "Admin"
      username: "admin"
    YvfzKrgbAWWp6Hy2zqqXtQbAT2s2
      email: "user3@a.com"
      role: "User"
      username: "user3"
  Users Logged
    user1: "37.421998333333335:-122.08400000000002"
```

Fig 8: The Database

The database stores data about the Objectives, the users and about users who are logged into the application in a string format.

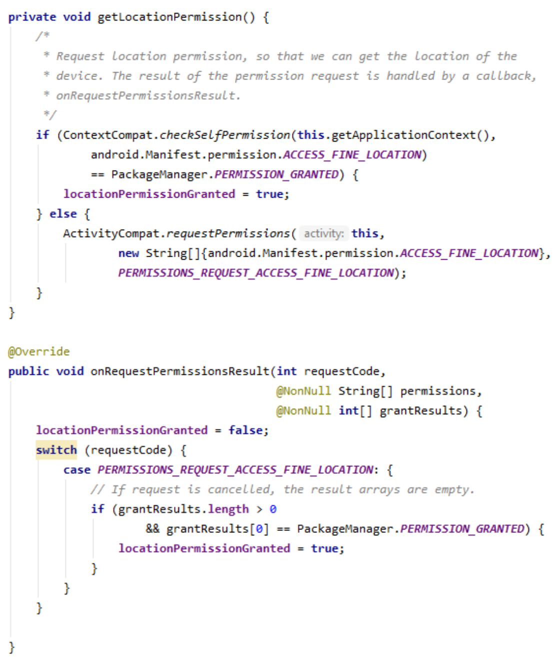
The Objectives part is named “LocationInfo” and contains the list of obejectives. Each Objective has four attributes holding the location information, location latitude, location longitude and location desired order. Order specifies the order in which the software will draw the polyline when the map is populated with objective markers.

It can clearly be seen that user1 and user2 have different information saved in the database. This is because the application, by default, stores the email, role and username when users are logged off. When a user logs in, the database extracts the location on the user and adds the extra information. It is visible that user2 is marked as logged in with coordinates saved as latitude and longitude. As soon as user2 logs out, the information about the location will be deleted. This information is only being used by the Admins who want to check the location of a group.

3.3.4 Login Activity

The purpose of the Log in activity is to enable users to use the application depending on the role they have. The activity is handled by the Firebase Authentication service which has been modified in order to meet the needs of the application.

Before the user can log in, the application will ask for several permissions. If the GPS is not enabled, the application will ask for permission to enable the GPS. If the Location services are not enabled the application will ask permission for Location on the phone (**Fig 11**).



```
private void getLocationPermission() {
    /*
     * Request location permission, so that we can get the location of the
     * device. The result of the permission request is handled by a callback,
     * onRequestPermissionsResult.
     */
    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
            android.Manifest.permission.ACCESS_FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED) {
        locationPermissionGranted = true;
    } else {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    locationPermissionGranted = false;
    switch (requestCode) {
        case PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                locationPermissionGranted = true;
            }
        }
    }

}
```

Fig 11: Permissions

If the user is already registered, all he needs to do is to enter the email in the email field and the password in the password field and then press the "LogIn" button. Once that is done, there are three possible outcomes:

- The information provided is valid and exists in the database and the user is successfully logged into the application and transferred to the correct part of the application depending on the authorisation.
- The information provided has a valid email that exists in the database but has the wrong password. A message, in the form of a Toast, will show on the screen telling the user that the authentication failed.
- The information provided is invalid and does not exist in the database informing the user that the authentication failed (**Fig 9**).

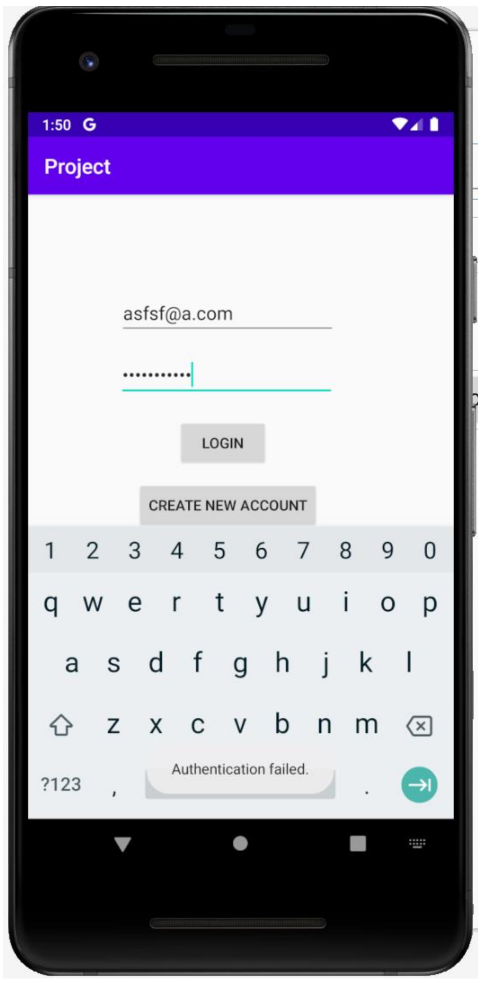
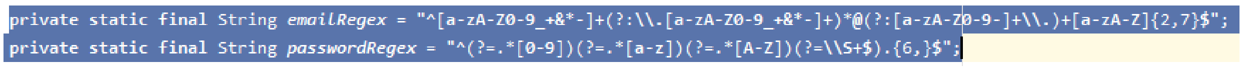


Fig 9: Failed Authentication

Input validation is handled with the help of Regular Expressions which are a sequence of characters that define a search pattern. The pattern for the email is different from the pattern of the password field. The email validation (**Fig 10**) checks for a classic email format while the password validation (**Fig 10**) checks for white spaces, at least 6 characters, at least one upper case, at least one lower case and at least one numeric character. Other types of characters can be used to form a password but those are the minimum requirements.



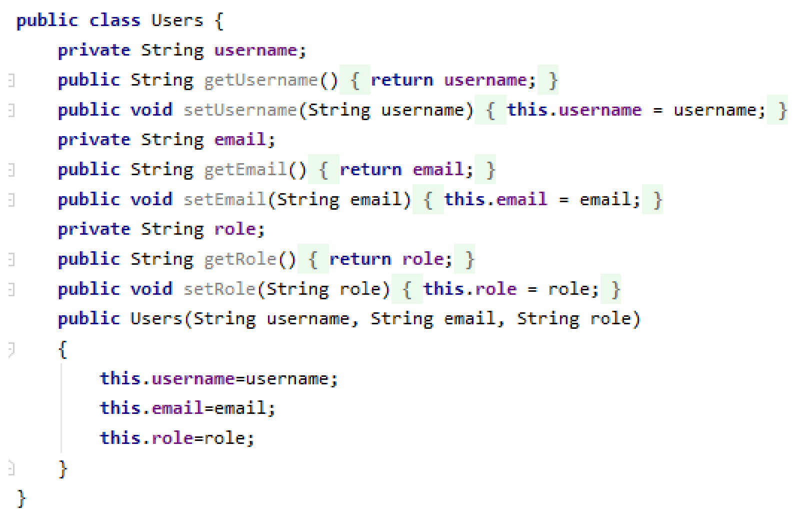
```
private static final String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";
private static final String passwordRegex = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=\\S+$).{6,}$";
```

Fig 10: Email and Password Regular Expressions

Once a user has been successfully logged in, the application will redirect towards the necessary activity. If the user needs to register an account, he needs to press the “Register Account” button and will be redirected to the account registration page.

3.3.5 AccRegister Activity

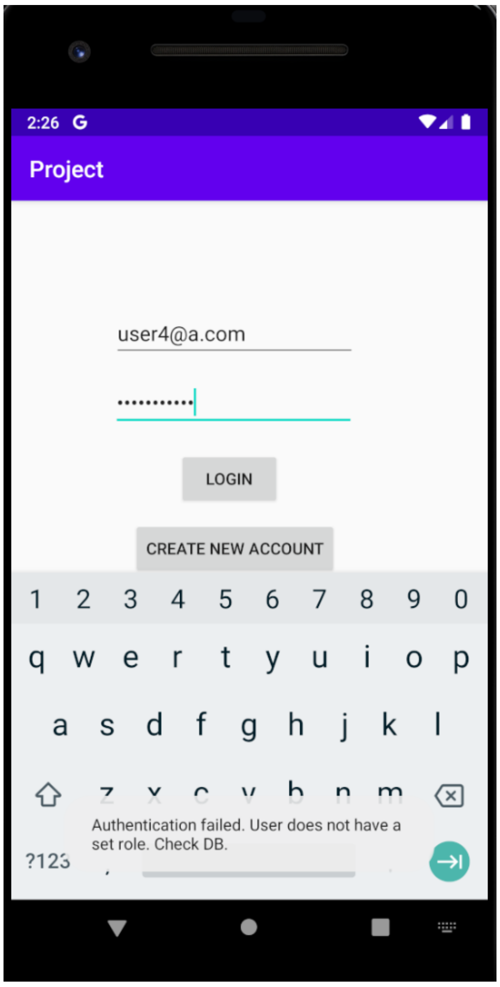
The Account registration activity (**Fig 14**) works with the Users class (**Fig 12**). The information that is being stored in the database is the username, the email and the password. All three variables are stored as strings.



```
public class Users {
    private String username;
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    private String email;
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    private String role;
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
    public Users(String username, String email, String role)
    {
        this.username=username;
        this.email=email;
        this.role=role;
    }
}
```

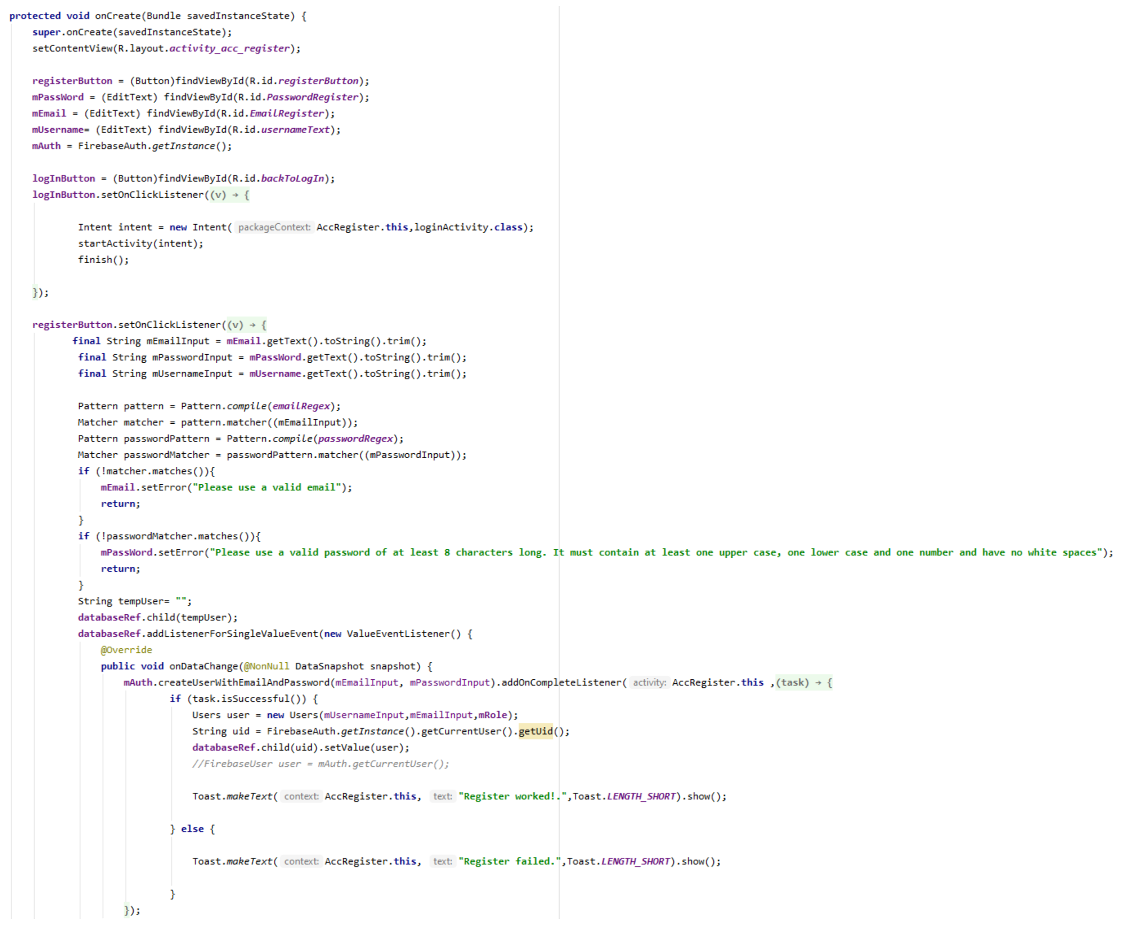
Fig 12: Users Class

The validation for the required fields follows the same rules as for Account login with the exception that a double password check is made in order to decide that the two password fields match. The other difference being that the username field requires no validation since anything can be inputted but, if left empty or default "Username", the email will be used to set the username inside the database. The first part of the email will be extracted and used. If the information provided is valid it will be added to the database and the role must be set by an admin that has access to the database. By default, the role is set as "To be set". A user without a valid role cannot use the application (**Fig 13**).



Authentication failed. User does not have a set role. Check DB.

Fig 13: Role Check



```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_acc_register);

    registerButton = (Button)findViewById(R.id.registerButton);
    mPassWord = (EditText) findViewById(R.id.PasswordRegister);
    mEmail = (EditText) findViewById(R.id.EmailRegister);
    mUsername= (EditText) findViewById(R.id.usernameText);
    mAuth = FirebaseAuth.getInstance();

    logInButton = (Button)findViewById(R.id.backToLogIn);
    logInButton.setOnClickListener((v) → {

            Intent intent = new Intent( packageContext: AccRegister.this,loginActivity.class);
            startActivity(intent);
            finish();

    });

    registerButton.setOnClickListener((v) → {
           final String mEmailInput = mEmail.getText().toString().trim();
            final String mPasswordInput = mPassWord.getText().toString().trim();
            final String mUsernameInput = mUsername.getText().toString().trim();

            Pattern pattern = Pattern.compile(emailRegex);
            Matcher matcher = pattern.matcher((mEmailInput));
            Pattern passwordPattern = Pattern.compile(passwordRegex);
            Matcher passwordMatcher = passwordPattern.matcher((mPasswordInput));
            if (!matcher.matches()){
                mEmail.setError("Please use a valid email");
                return;
            }
            if (!passwordMatcher.matches()){
                mPassWord.setError("Please use a valid password of at least 8 characters long. It must contain at least one upper case, one lower case and one number and have no white spaces");
                return;
            }
            String tempUser= "";
            databaseRef.child(tempUser);
            databaseRef.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    mAuth.createUserWithEmailAndPassword(mEmailInput, mPasswordInput).addOnCompleteListener( activity: AccRegister.this ,(task) → {
                            if (task.isSuccessful()) {
                                Users user = new Users(mUsernameInput,mEmailInput,mRole);
                                String uid = FirebaseAuth.getInstance().getCurrentUser().getUid();
                                databaseRef.child(uid).setValue(user);
                                //FirebaseUser user = mAuth.getCurrentUser();

                                Toast.makeText( context: AccRegister.this,  text: "Register worked!.",Toast.LENGTH_SHORT).show();

                            } else {

                                Toast.makeText( context: AccRegister.this,  text: "Register failed.",Toast.LENGTH_SHORT).show();

                            }
                    });
```

Fig 14: Account Registration Code

Once a user has been registered, he has to press the “LogIn” button to return to the Log In page where he will enter his information and attempt to log in.

3.3.6 Administrator Section

3.3.6.1 CheckUsersLocation Activity

As soon as a user has logged in and if he bears the role of “Admin” he will be redirected towards the CheckUsersLocation activity. This activity is a Goggle Maps activity fragment that has a four buttons menu (**Fig 15**).

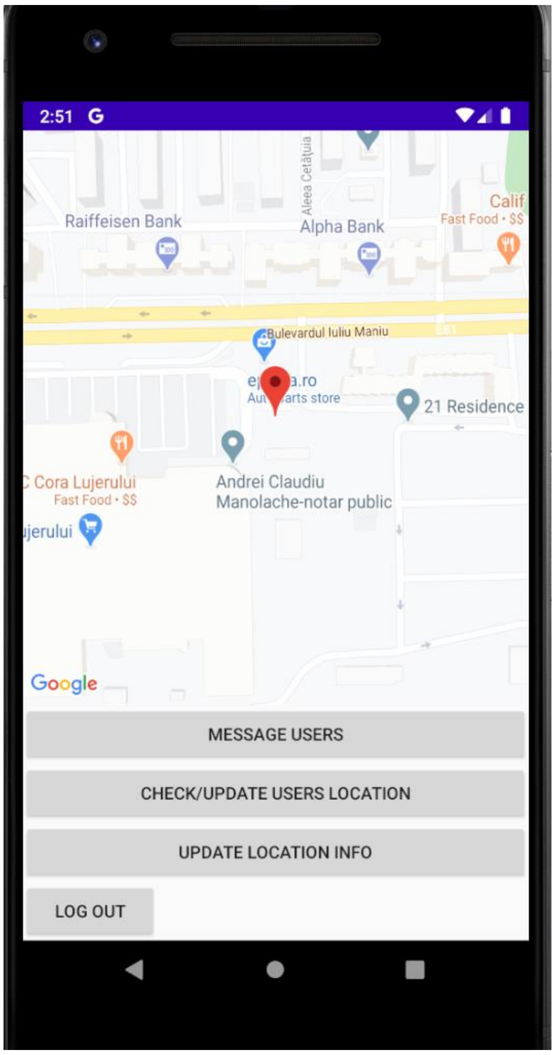


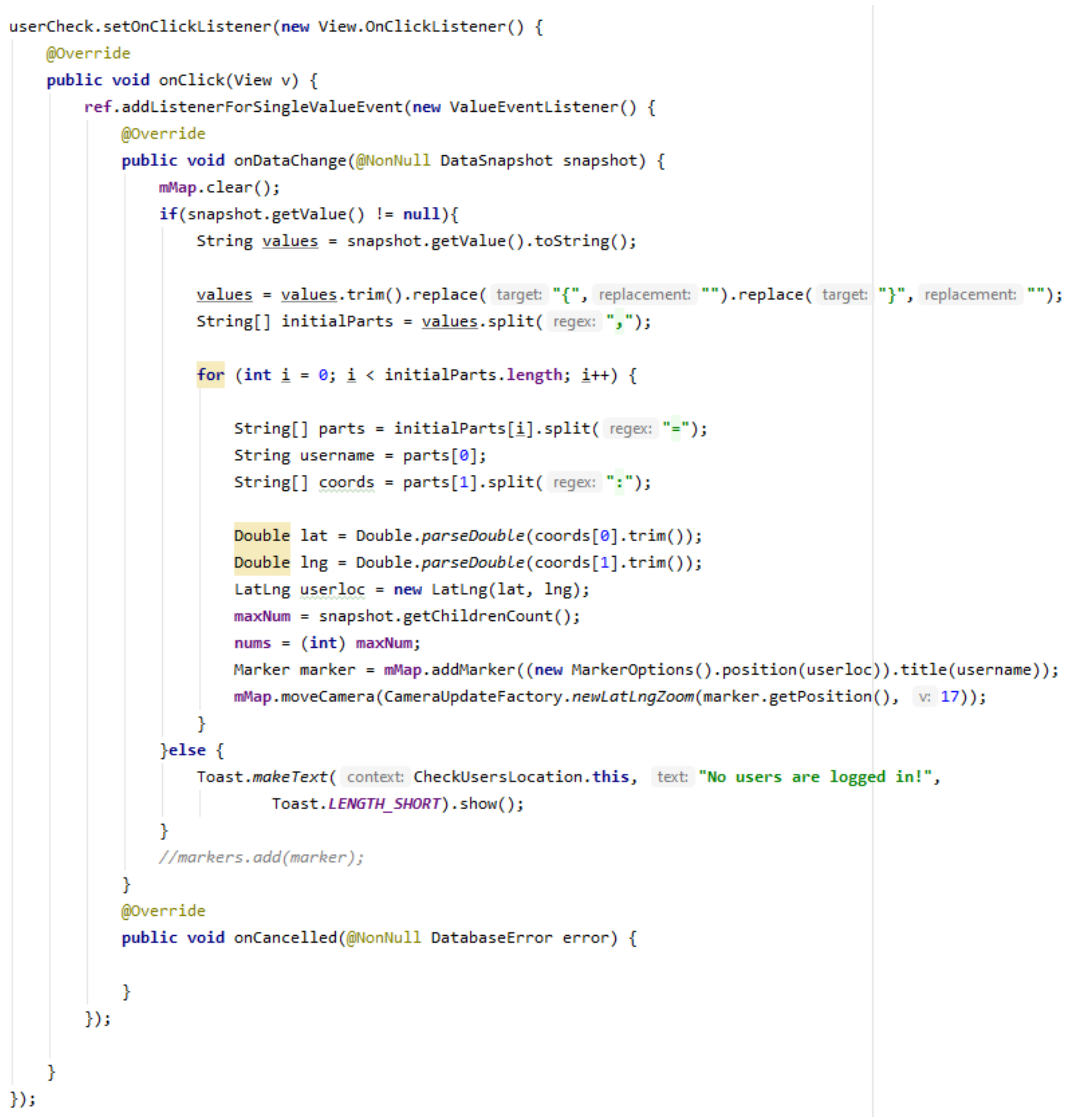
Fig 15: Admin – Location of a logged in user

The first button is “Message Users” and when pressed it starts a new activity called “Chat”. This is the way the admin sends messages to the users that are logged in.

The second button is “Check/Update Users Location” and it is coded as a listener button for changes in the database. When pressed it will query the database for logged in Guide users, will go through every user that is currently logged in, extract their location and username information, build markers with the extracted latitude and longitude and place the markers on the map. The markers bear information pointing out the username of the user. If the button is pressed again it will perform a map clear meaning it will remove all the markers from the map and will update the map with the new information from the database (**Fig 16**).

The marker building process consists of the application creating a query. Whatever is returned from the database is added to a string in a raw format.

The unwanted characters from the raw format are removed and then it is split into parts creating an array of strings. Each part represents a somewhat raw state of a marker. The function then iterates through the newly created array further splitting it into two parts and extracting the username from the first part. The second part contains information about position on the map. This part is further split and the latitude and longitude are extracted and added to a variable of type double which will be used to build the actual marker. Once the marker has been built it will be placed on the map and the camera will move on top of it.



```
userCheck.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ref.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                mMap.clear();
                if(snapshot.getValue() != null){
                    String values = snapshot.getValue().toString();

                    values = values.trim().replace( target: "{", replacement: "").replace( target: "}", replacement: "");
                    String[] initialParts = values.split( regex: ",");

                    for (int i = 0; i < initialParts.length; i++) {

                        String[] parts = initialParts[i].split( regex: "=");
                        String username = parts[0];
                        String[] coords = parts[1].split( regex: ":");

                        Double lat = Double.parseDouble(coords[0].trim());
                        Double lng = Double.parseDouble(coords[1].trim());
                        LatLng userloc = new LatLng(lat, lng);
                        maxNum = snapshot.getChildrenCount();
                        nums = (int) maxNum;
                        Marker marker = mMap.addMarker((new MarkerOptions().position(userloc)).title(username));
                        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(marker.getPosition(),  v: 17));
                    }
                }else {
                    Toast.makeText( context: CheckUsersLocation.this,  text: "No users are logged in!",
                            Toast.LENGTH_SHORT).show();
                }
                //markers.add(marker);
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

    }
});
```

Fig 16: Admin – User Check Code

The third button is called “Update Location Info” and when pressed it will open a new activity called “Admin”.

The fourth button is the “LogOut” button which when pressed will finish the current activity and will log the user out from the application and database but not before checking if the database has registered the user’s location. If it has it will delete the data from the database (**Fig 17**).

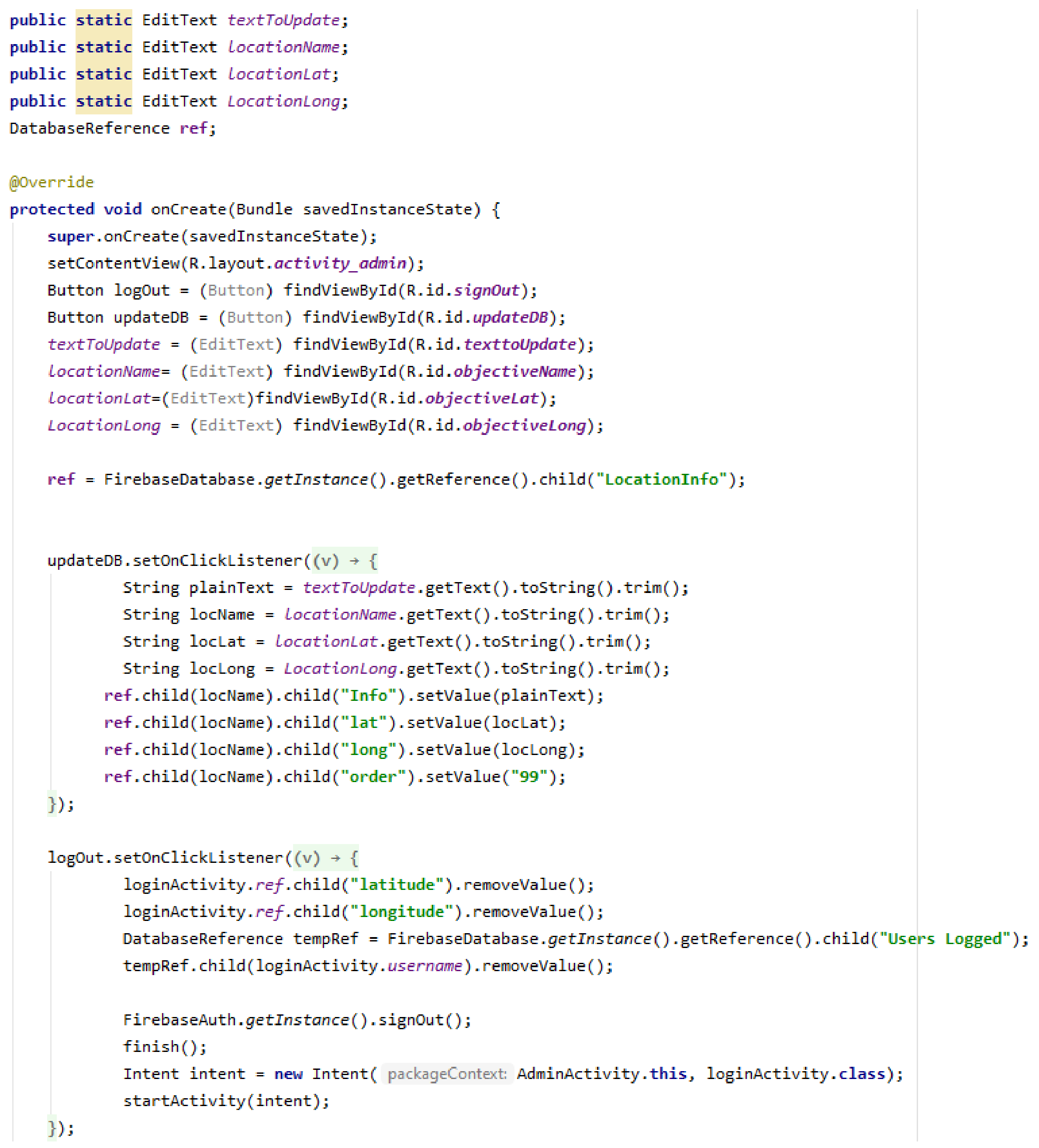
To mention is the fact that, except for the “LogOut” button, all other buttons do not finish the current activity thus saving the data in the current state in case the user wants to come back here from any other activity.

3.3.6.2 Admin Activity (Fig 17)

An admin can only get to this activity from the CheckUserLocation activity by pressing “Update Location Info” button. This activity has four text fields, one for the name of the objective that needs to be added or updated, one for the latitude of the objective, one for the longitude and one for the information regarding that objective. On top of the information from the fields the function adds and extra attribute to the created entry. The attribute is called “order” and it will be used later in populating the map with objective markers. It also has two buttons, one for updating the database and one for logging out. If the admin wants to return to the previous activity, he must press the back button on the.

This is a very simple class that has very basic functionality. The name of the objective, latitude, longitude, order and the information will be passed into four different string variables and once the “Update Database” button is pressed, a query will be sent to the database with the required information, updating the database.

The “LogOut” button (**Fig 17**) has the same functionality as in CheckUsersLocation activity.



```java
public static EditText textToUpdate;
public static EditText locationName;
public static EditText locationLat;
public static EditText LocationLong;
DatabaseReference ref;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_admin);
    Button logOut = (Button) findViewById(R.id.signOut);
    Button updateDB = (Button) findViewById(R.id.updateDB);
    textToUpdate = (EditText) findViewById(R.id.texttoUpdate);
    locationName= (EditText) findViewById(R.id.objectiveName);
    locationLat=(EditText)findViewById(R.id.objectiveLat);
    LocationLong = (EditText) findViewById(R.id.objectiveLong);

    ref = FirebaseDatabase.getInstance().getReference().child("LocationInfo");


    updateDB.setOnClickListener((v) → {
            String plainText = textToUpdate.getText().toString().trim();
            String locName = locationName.getText().toString().trim();
            String locLat = locationLat.getText().toString().trim();
            String locLong = LocationLong.getText().toString().trim();
        ref.child(locName).child("Info").setValue(plainText);
        ref.child(locName).child("lat").setValue(locLat);
        ref.child(locName).child("long").setValue(locLong);
        ref.child(locName).child("order").setValue("99");
    });

    logOut.setOnClickListener((v) → {
            loginActivity.ref.child("latitude").removeValue();
            loginActivity.ref.child("longitude").removeValue();
            DatabaseReference tempRef = FirebaseDatabase.getInstance().getReference().child("Users Logged");
            tempRef.child(loginActivity.username).removeValue();

            FirebaseAuth.getInstance().signOut();
            finish();
            Intent intent = new Intent( packageContext: AdminActivity.this, loginActivity.class);
            startActivity(intent);
    });
```

Fig 17: Admin Activity

3.3.7 Guide User Section

3.3.7.1 Maps Activity

The Maps Activity (**Fig 19**) is the most complex of all activities included in the application. It is a Google Maps Activity that contains a map fragment, a textView section, a TextDisplay section and a three buttons menu.

The methods contained in the activity check for permissions and GPS status (**Fig 18**), a method that calculates the distance between the first objective of the tour and user's location, a method that displays information about a selected objective, a method that starts a timer when the tour starts and a method that continuously checks for the guide’s location and updates it on the map.

The first thing that happens when a guide user logs in is being sent to this activity. Initially, the application fills the map with the set objective markers and draws a poly line between them to try and show the guide the visit order of the objectives (**Fig 20**). The process of filling the map with objective markers happens as follows:

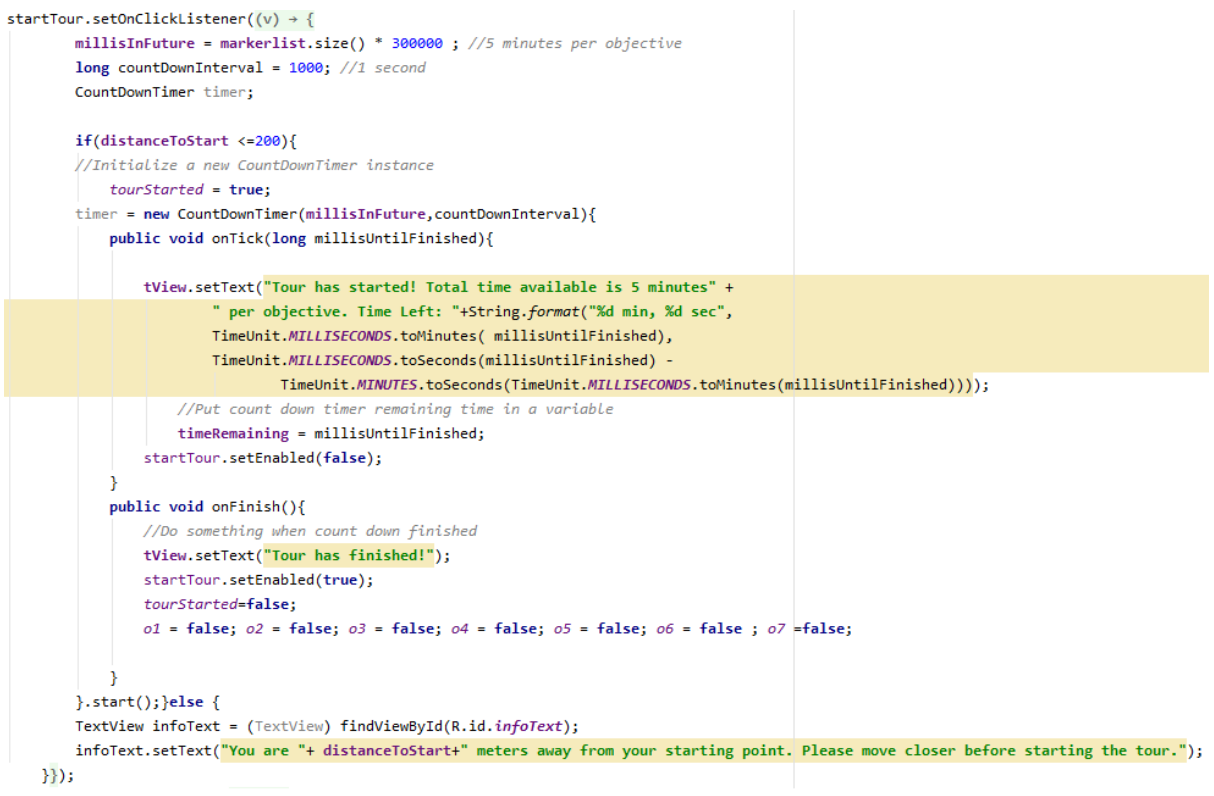
- First, the function creates a database reference on the “LocationInfo” node, and it reads everything that is contained in that node adding it to a variable of DataSnapshot type. Because Firebase stores data in alphabetical order by node name, the snapshot variable contains data in an unwanted order.
- The snapshot is then transferred to a string variable and an empty hash map is being initialized which will be used later to order the data as needed.
- The string containing the snapshot value is split at the desired points creating an array.
- The function then iterates through the array and replaces unwanted characters with wanted ones bringing the array to the point where is contains each objective info divided only by commas.
- Each part of the initial array is further split by commas and the resulting parts are added to variables each containing the desired information about the objective. The most important variable is the “objOrder” one that will be parsed as an Integer and will be used later for ordering the previously created map.
- A new string is being created containing information about the objective that will be used later.
- The function then creates the marker, adds it to a list of markers and adds the information about the objectives to another list and populates the map with the previously set Integer as a key and the previously built string as the value of that key.
- After the iteration has finished, the map is being ordered in ascending order using the key.
- The function iterates through the ordered map and adds each value of each key to a new list of strings which represents the raw markers data in the desired order.

- One final step, the function iterates through the list created on the previous step, splitting it and adding each part to a variable that will be used to create and populate the polyline which, is finally drawn.

The map moves its camera above the first objective of the tour until the user location has been established and updated for the first time, at which point it will move the camera on top of the guide and will start following it. The next thing it will do is check for permissions and GPS status and will send notifications and permissions requests if necessary (**Fig 18**). The application then checks for the user's location and calculates the distance between the first objective of the tour and the user (**Fig 21**). If the user is further than 200 meters from the first objective, the application will disable the "Start Tour" button and will send a notification which will be displayed in the TextView field informing the user about the impossibility of starting a tour from that distance and the distance in meters. Because the location is checked constantly, as soon as the guide approaches the first objective by an acceptable distance, the "Start Tour" button will become enabled.

On "Start tour" button being pressed three things will happen:

- A new distance check between the objective and guide's location will be made. If the distance is less or equal to 200 meters, a Boolean variable will set to true, flagging the fact that the tour has started. If the distance in not accepted a notification will be displayed in the TextView field. A Boolean variable, which by default is set to false, will be set to true, flagging an ongoing tour.
- the application will start a timer that is calculated based on the size of the markers list created on the previously explained step. For each objective, the function will add 5 minutes to the total time available. If there are 5 objective entries in the list, the timer will be 25 minutes, if there are 6, 30 minutes, etc. The timer will be displayed in the TextDisplay section, at the bottom of the UI (**Fig 22)**.
- The "Start Tour" button will be disabled if the tour starts successfully. When the tour has ended, the variable previously set to true will become false and the "Start Tour" button will be enabled.



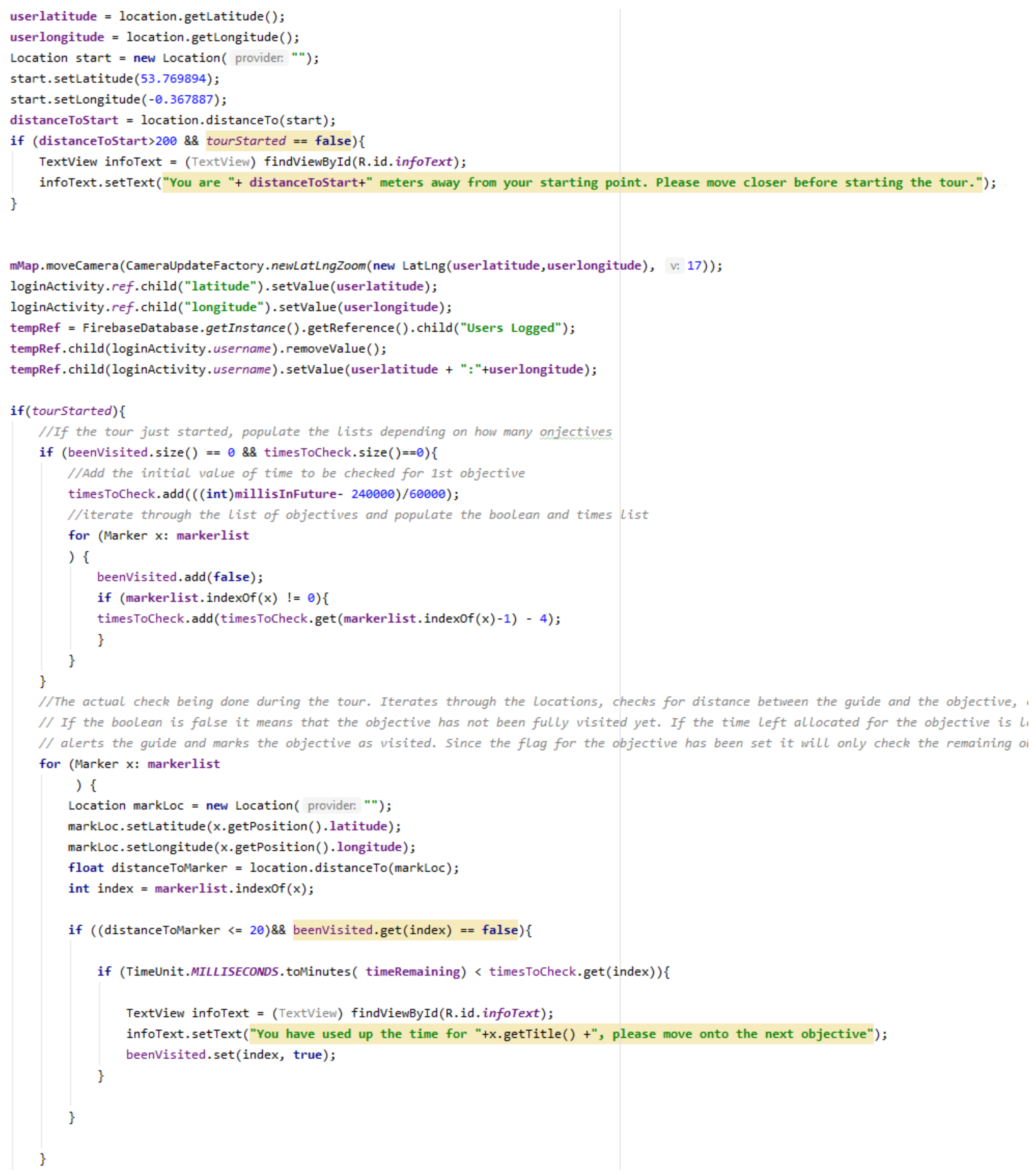
```
startTour.setOnClickListener((v) → {
        millisInFuture = markerlist.size() * 300000 ; //5 minutes per objective
        long countDownInterval = 1000; //1 second
        CountDownTimer timer;

        if(distanceToStart <=200){
        //Initialize a new CountDownTimer instance
            tourStarted = true;
        timer = new CountDownTimer(millisInFuture,countDownInterval){
            public void onTick(long millisUntilFinished){

                tView.setText("Tour has started! Total time available is 5 minutes" +
                        " per objective. Time Left: "+String.format("%d min, %d sec",
                        TimeUnit.MILLISECONDS.toMinutes( millisUntilFinished),
                        TimeUnit.MILLISECONDS.toSeconds(millisUntilFinished) -
                                TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(millisUntilFinished))));
                    //Put count down timer remaining time in a variable
                    timeRemaining = millisUntilFinished;
                startTour.setEnabled(false);
            }
            public void onFinish(){
                //Do something when count down finished
                tView.setText("Tour has finished!");
                startTour.setEnabled(true);
                tourStarted=false;
                o1 = false; o2 = false; o3 = false; o4 = false; o5 = false; o6 = false ; o7 =false;

            }
        }.start();}else {
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You are "+ distanceToStart+" meters away from your starting point. Please move closer before starting the tour.");
    }});
```

Fig 22: Timer

While the tour is active the Location manager continuously checks for the guide's location and compares it to the location of the objectives (**Fig 23**). The function logic flows as flows:

- First, the user's location is being checked and the parameters added to longitude and latitude variables.
- A new distance check to the first objective is made.
- A firebase update with the new user's location is made.
- If the "Start Tour" button has been pressed the "tourStarted" Boolean will be set to true and the main part of the function will be entered.
- In here, a check on the lists used to validate is being made. There are two lists that will be used, one containing Boolean values for each objective and one containing the time intervals at which the alerting service triggers. If the lists are empty a new entry will be added to the list containing the time intervals to be checked. This entry will be set as total available time minus four minutes. The reason behind setting to four and not five minutes is because when the guide is being alerted that the objective's allocated time is about to finish, he still has one minute to wrap things up and move on.

- Immediately after, the function iterates through the list of markers which was previously created and populates the Boolean list and the times list. The reason behind using the markers list in order to populate the other, dependant lists is because we want to have the same size on all three lists. This way we ensure no exception is being thrown if the list sizes do not match.
- The function exits the first if statement and enters another iteration. For each marker in the list of markers a new Location variable is being created. That variable copies the positioning values of the marker and a distance difference between the objective represented by the marker and the user location is being calculated and assigned to a float variable. The index of the current marker in the list is being assigned to an Integer variable for simplicity in the following steps.
- A check is made next verifying if the distance previously calculated is less than twenty meters coupled with a check against the Boolean value assigned to the marker we are iterating over. If either of them returns false, the iteration continues with the next marker until the marker list is exhausted. If both return true then a new check will be made against the remaining tour time and if the remaining time is less than the time previously assigned to the current objective the guide will be alerted and the Boolean value attached to the objective will be set to true ensuring that the checks will not happen again against this objective which is not flagged as visited.
- The function performs these checks every time the guide’s location is updated in the database.



```
userlatitude = location.getLatitude();
userlongitude = location.getLongitude();
Location start = new Location( provider: "");
start.setLatitude(53.769894);
start.setLongitude(-0.367887);
distanceToStart = location.distanceTo(start);
if (distanceToStart>200 && tourStarted == false){
    TextView infoText = (TextView) findViewById(R.id.infoText);
    infoText.setText("You are "+ distanceToStart+" meters away from your starting point. Please move closer before starting the tour.");
}


mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(userlatitude,userlongitude),  v: 17));
loginActivity.ref.child("latitude").setValue(userlatitude);
loginActivity.ref.child("longitude").setValue(userlongitude);
tempRef = FirebaseDatabase.getInstance().getReference().child("Users Logged");
tempRef.child(loginActivity.username).removeValue();
tempRef.child(loginActivity.username).setValue(userlatitude + ":"+userlongitude);

if(tourStarted){
    //If the tour just started, populate the lists depending on how many onjectives
    if (beenVisited.size() == 0 && timesToCheck.size()==0){
        //Add the initial value of time to be checked for 1st objective
        timesToCheck.add(((int)millisInFuture- 240000)/60000);
        //iterate through the list of objectives and populate the boolean and times list
        for (Marker x: markerlist
        ) {
            beenVisited.add(false);
            if (markerlist.indexOf(x) != 0){
            timesToCheck.add(timesToCheck.get(markerlist.indexOf(x)-1) - 4);
            }
        }
    }
    //The actual check being done during the tour. Iterates through the locations, checks for distance between the guide and the objective, 
    // If the boolean is false it means that the objective has not been fully visited yet. If the time left allocated for the objective is l
    // alerts the guide and marks the objective as visited. Since the flag for the objective has been set it will only check the remaining o
    for (Marker x: markerlist
         ) {
        Location markLoc = new Location( provider: "");
        markLoc.setLatitude(x.getPosition().latitude);
        markLoc.setLongitude(x.getPosition().longitude);
        float distanceToMarker = location.distanceTo(markLoc);
        int index = markerlist.indexOf(x);

        if ((distanceToMarker <= 20)&& beenVisited.get(index) == false){

            if (TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < timesToCheck.get(index)){

                TextView infoText = (TextView) findViewById(R.id.infoText);
                infoText.setText("You have used up the time for "+x.getTitle() +", please move onto the next objective");
                beenVisited.set(index, true);
            }

        }

    }
```

Fig 23: Tour Checks

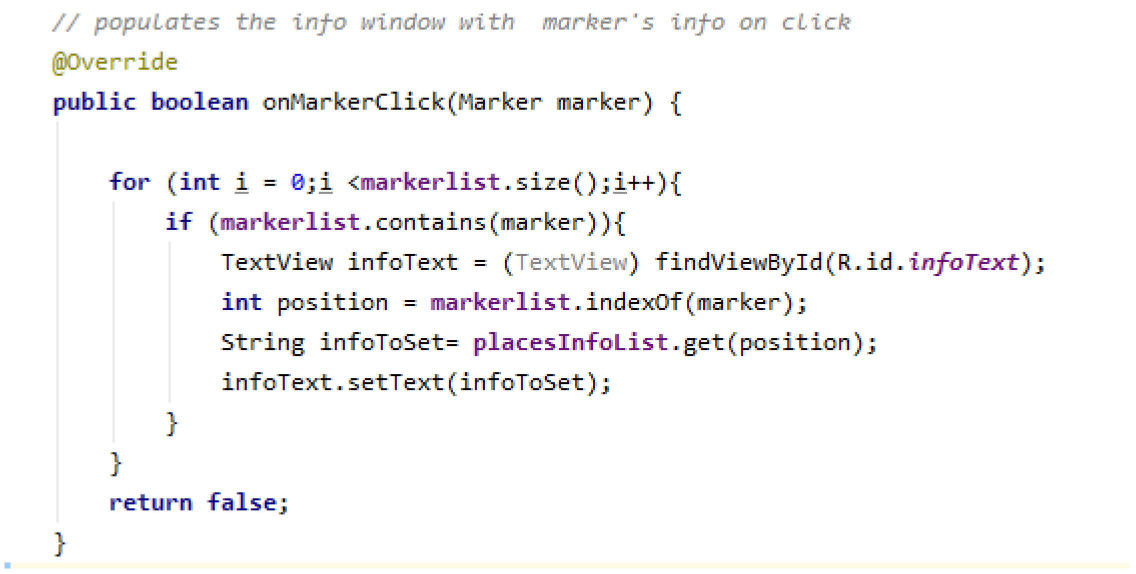
Each objective has a set time limit of 5 minutes.

If the user has passed the first objective, the application makes checks for the next objective. The Boolean used for the previous objective is being checked in order to know which objective is being visited now. The process is the same for all objectives.

The “Verify Messages” button starts a new Chat activity and allows the guide to check for incoming messages without interrupting an ongoing tour.

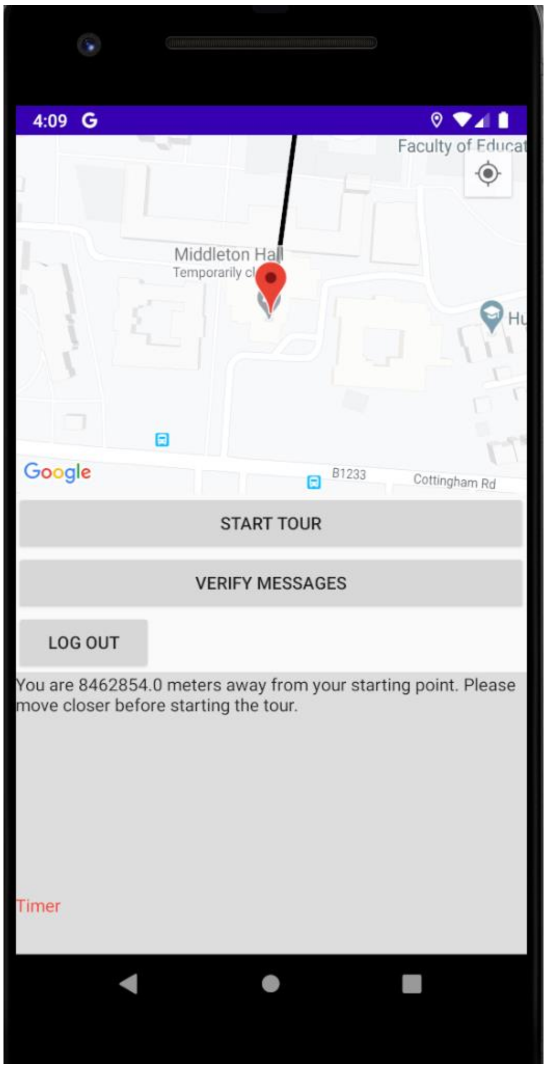
The “LogOut” button logs the user out from the application and from the database and deletes the information about his location.

Important note: By tapping any objective marker on the map, the application will extract information from the database, more precisely from the lists previously created, and will display it in the TextView field (**Fig 33)**. This feature is available regardless if the tour has started or not.



```
// populates the info window with  marker's info on click
@Override
public boolean onMarkerClick(Marker marker) {

    for (int i = 0;i <markerlist.size();i++){
        if (markerlist.contains(marker)){
            TextView infoText = (TextView) findViewById(R.id.infoText);
            int position = markerlist.indexOf(marker);
            String infoToSet= placesInfoList.get(position);
            infoText.setText(infoToSet);
        }
    }
    return false;
}
```

Fig 33: Populating Markers Info on Click



You are 8462854.0 meters away from your starting point. Please move closer before starting the tour.

Timer

Fig 19: Maps Activity

```
public void statusCheck() {
    final LocationManager manager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

    if (!manager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
        buildAlertMessageNoGps();

    }
}

private void buildAlertMessageNoGps() {
    final AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setMessage("Your GPS seems to be disabled, do you want to enable it?")
            .setCancelable(false)
            .setPositiveButton( text: "Yes", (dialog, id) → {
                    startActivity(new Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS));
            })
            .setNegativeButton( text: "No", (dialog, id) → {
                    dialog.cancel();
            });
    final AlertDialog alert = builder.create();
    alert.show();
}
```

Fig 18: Permissions and GPS

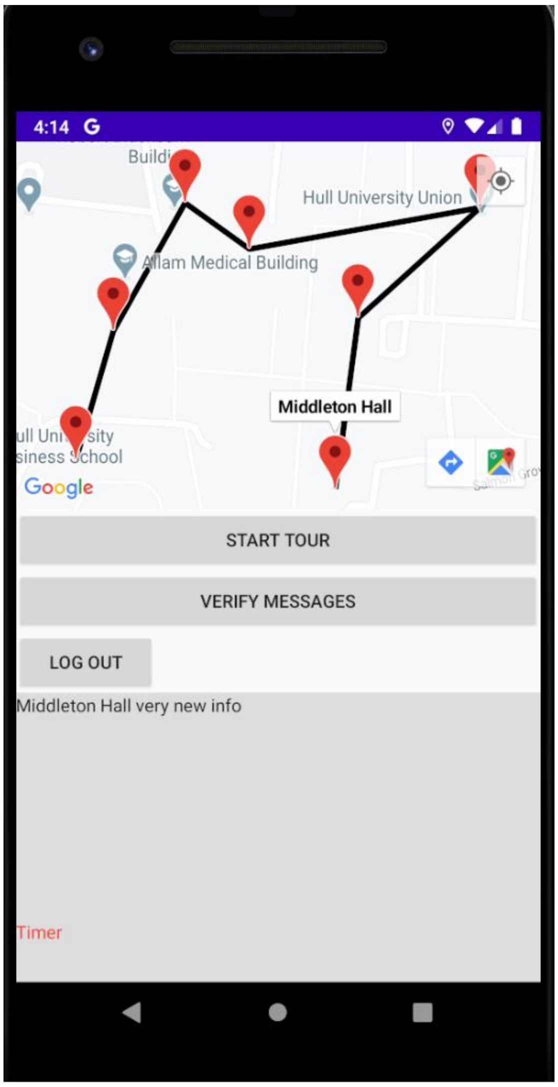
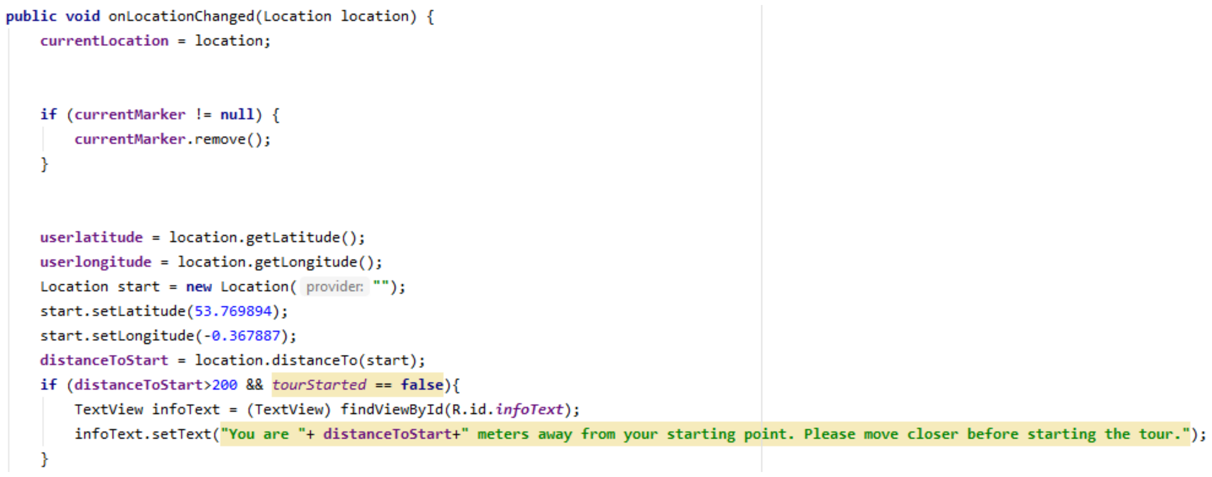


Fig 20: Objectives and PolyLine



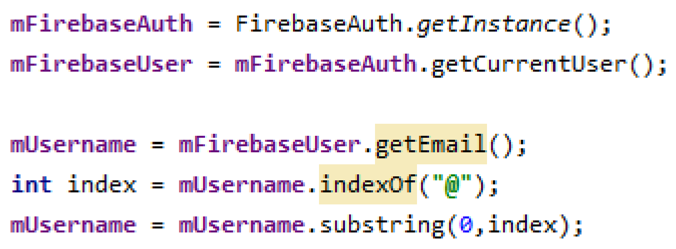
```
public void onLocationChanged(Location location) {
    currentLocation = location;


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


    userlatitude = location.getLatitude();
    userlongitude = location.getLongitude();
    Location start = new Location( provider: "");
    start.setLatitude(53.769894);
    start.setLongitude(-0.367887);
    distanceToStart = location.distanceTo(start);
    if (distanceToStart>200 && tourStarted == false){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You are "+ distanceToStart+" meters away from your starting point. Please move closer before starting the tour.");
    }
```

Fig 21: Distance check

3.3.7.2 Chat Activity

The chat activity is based on Firebase's Codelab FriendlyChat which has been simplified and edited in order to match the project's scope. Integration with the existing project and existing Firebase instance had to be done.

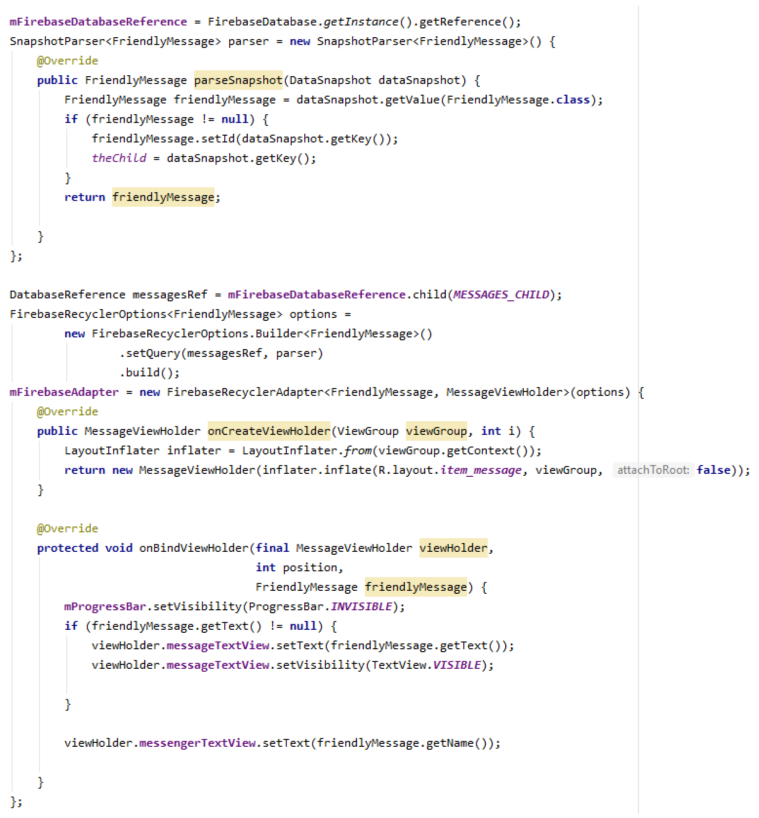
First, the project's gradle files had to be synced. Access to the database follows the same JSON rules as stated previously. Chat authentication implements the exact same authentication as the application, importing information from the login activity and the database (**Fig 25**).



```
mFirebaseAuth = FirebaseAuth.getInstance();
mFirebaseUser = mFirebaseAuth.getCurrentUser();

mUsername = mFirebaseUser.getEmail();
int index = mUsername.indexOf("@");
mUsername = mUsername.substring(0,index);
```

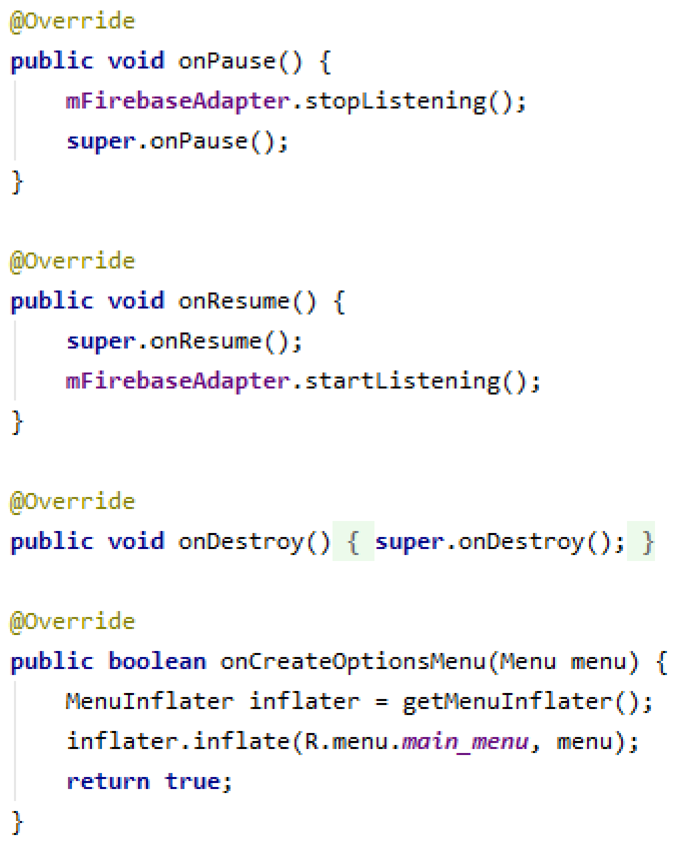
Fig 25: Chat

Reading messages from the database is done with the help of a Firebase format query that reads the database for all existing messages and adding them to the chat window. Then, it listens for new child entries under the correct path, in this case "messages", and when a new message is added to the database it extracts the information and adds it to the chat UI (**Fig 26**). Additional methods are implemented in order to be able to pause and resume the listening process (**Fig 27**).



```
mFirebaseDatabaseReference = FirebaseDatabase.getInstance().getReference();
SnapshotParser<FriendlyMessage> parser = new SnapshotParser<FriendlyMessage>() {
    @Override
    public FriendlyMessage parseSnapshot(DataSnapshot dataSnapshot) {
        FriendlyMessage friendlyMessage = dataSnapshot.getValue(FriendlyMessage.class);
        if (friendlyMessage != null) {
            friendlyMessage.setId(dataSnapshot.getKey());
            theChild = dataSnapshot.getKey();
        }
        return friendlyMessage;

    }
};

DatabaseReference messagesRef = mFirebaseDatabaseReference.child(MESSAGES_CHILD);
FirebaseRecyclerOptions<FriendlyMessage> options =
        new FirebaseRecyclerOptions.Builder<FriendlyMessage>()
                .setQuery(messagesRef, parser)
                .build();
mFirebaseAdapter = new FirebaseRecyclerAdapter<FriendlyMessage, MessageViewHolder>(options) {
    @Override
    public MessageViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        LayoutInflater inflater = LayoutInflater.from(viewGroup.getContext());
        return new MessageViewHolder(inflater.inflate(R.layout.item_message, viewGroup,  attachToRoot: false));
    }

    @Override
    protected void onBindViewHolder(final MessageViewHolder viewHolder,
                                    int position,
                                    FriendlyMessage friendlyMessage) {
        mProgressBar.setVisibility(ProgressBar.INVISIBLE);
        if (friendlyMessage.getText() != null) {
            viewHolder.messageTextView.setText(friendlyMessage.getText());
            viewHolder.messageTextView.setVisibility(TextView.VISIBLE);

        }

        viewHolder.messengerTextView.setText(friendlyMessage.getName());

    }
};
```

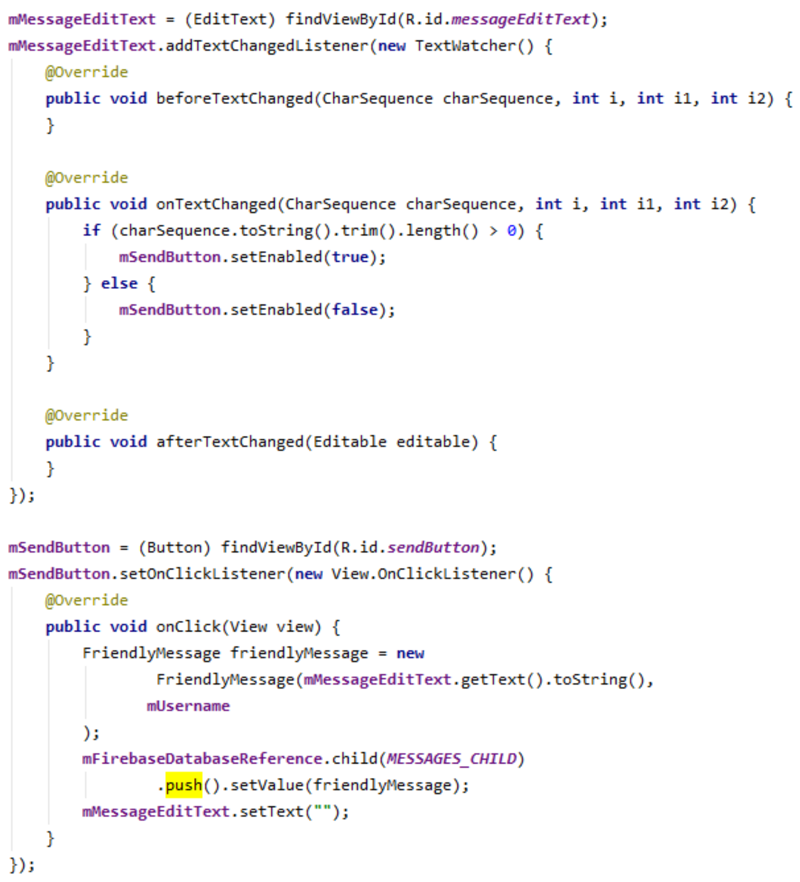
Fig 26: Chat read messages



```
@Override
public void onPause() {
    mFirebaseAdapter.stopListening();
    super.onPause();
}

@Override
public void onResume() {
    super.onResume();
    mFirebaseAdapter.startListening();
}

@Override
public void onDestroy() { super.onDestroy(); }

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}
```

Fig 27: Chat Pause/Resume

Sending messages follows similar path to reading messages. Listeners are added to the “Send” button that, when receive new information, create a FriendlyMessage object (**Fig 28**) filled with the contents of the input field which is then pushed to the database. The push() method (**Fig 28**) generates a sequential GUID attached to the object, ensuring that the new object will be added exactly at the end of the messages list.



```
mMessageEditText = (EditText) findViewById(R.id.messageEditText);
mMessageEditText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {
    }

    @Override
    public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
        if (charSequence.toString().trim().length() > 0) {
            mSendButton.setEnabled(true);
        } else {
            mSendButton.setEnabled(false);
        }
    }

    @Override
    public void afterTextChanged(Editable editable) {
    }
});

mSendButton = (Button) findViewById(R.id.sendButton);
mSendButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        FriendlyMessage friendlyMessage = new
                FriendlyMessage(mMessageEditText.getText().toString(),
               mUsername
        );
        mFirebaseDatabaseReference.child(MESSAGES_CHILD)
                .push().setValue(friendlyMessage);
        mMessageEditText.setText("");
    }
});
```

Fig 28: Chat Send Message

Additional methods have been added for optional sending files and pictures, but they have not been implemented due to time constrains and the features having not been part of the initial project.

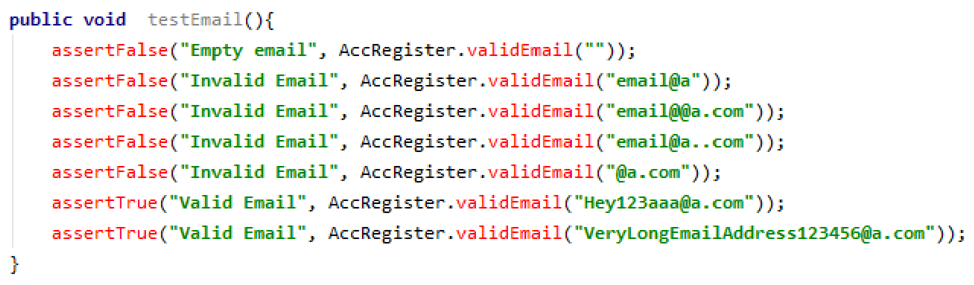
3.4 Testing

3.4.1 Unit Testing

Unit testing is defined as a certain level of application testing where individual, small parts of the application are tested against certain conditions. The aim of unit testing is to validate or invalidate whether each application component performs as designed and intended. Usually, the part that needs testing is isolated, required inputs are provided and an output is expected. The output is tested against a pre set output that matches the correct output value. Since this project uses an object-oriented programming language, the units that will be tested are methods. This project implemented unit testing in the only areas where a user must fill information that has to match a certain format: Account registration and Log in methods.

In Android Studio, unit testing is done locally without the need for extra tools or emulators. This saves a great deal of time that can be used to focus on other aspects of the project. It is done with the help of a built-in platform called Junit which is an automated framework that generates and runs scripts painless and quick without the need of manually inputting the required information. This, as well, saves valuable time.

Fig 29 shows an example of the testing implemented.



```
public void  testEmail(){
    assertFalse("Empty email", AccRegister.validEmail(""));
    assertFalse("Invalid Email", AccRegister.validEmail("email@a"));
    assertFalse("Invalid Email", AccRegister.validEmail("email@@a.com"));
    assertFalse("Invalid Email", AccRegister.validEmail("email@a..com"));
    assertFalse("Invalid Email", AccRegister.validEmail("@a.com"));
    assertTrue("Valid Email", AccRegister.validEmail("Hey123aaa@a.com"));
    assertTrue("Valid Email", AccRegister.validEmail("VeryLongEmailAddress123456@a.com"));
}
```

Fig 29: Unit Testing Example

The assertTrue and assertFalse options return Boolean values depending on the success of the test. If the test is successful, the returned value is “true” and if it is not successful the returned value is “false”.

Later in the project, when validation was replaced by Regular Expressions validation, the JUnit unit testing has been scrapped.

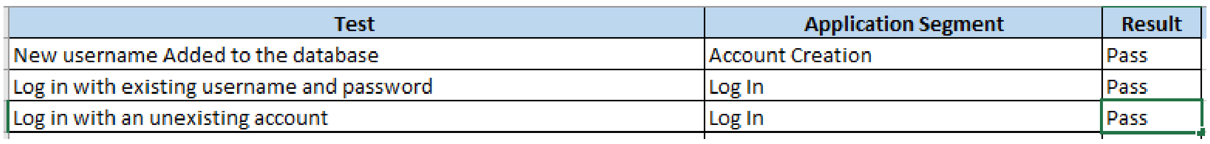
Another type of unit testing is Instrumented Testing. It uses a programmatically set emulator and performs tasks that Local Unit Testing cannot perform. This kind of testing was not implemented in the project because Acceptance Testing was a much more suited method of testing when it comes to the project’s scope and the available timeframe.

3.4.2 Acceptance Testing

Acceptance Testing is defined as “a formal description of the behaviour of a software product, generally expressed as an example or a usage scenario. A number of different notations and approaches have been proposed for such examples or scenarios. In many cases the aim is that it should be possible to automate the execution of such tests by a software tool, either ad-hoc to the development team or off the shelf.

Similar to a unit test, an acceptance test generally has a binary result, pass or fail. A failure suggests, though does not prove, the presence of a defect in the product." (Agile Alliance, 2020)

This form of testing is the main approach used in this project and was performed as soon as a new feature would be implemented, hence providing the developer with immediate feedback regarding how the feature performs and interacts with other elements of the project. This influenced later changes and design decisions.



Test	Application Segment	Result
New username Added to the database	Account Creation	Pass
Log in with existing username and password	Log In	Pass
Log in with an unexisting account	Log In	Pass

Fig 30: Acceptance Snippet

The above segment (**Fig 30**) is extracted from the full Acceptance Testing Table (**Appendix B**) seen below.

Executing the testing became, at some point during development, an repetitive process that was implemented as soon as a new feature was implemented or as soon as changes to a features have been made in order to make sure that, while the old state of the feature worked as intended, the new state is performing at the same level.

3.4.3 Testing Environment

While the application has been thoroughly tested on the developer's machine and Android phone it has also been tested on other people's devices and in different locations of the globe. Bucharest, Romania is one of those locations. This has been done in order to check that application's functionality and layout was preserved no matter the circumstances and that the user experience was not affected. Relative Layouts played a decisive role in achieving this. The XML objects describing the Layout decided how the layout will look depending on the device used. No value in the Layout has been entered manually. The devices used for testing are listed in the following table.

Device	Screen Size	Screen Resolution	Operating System
Pixel 3 XL	6.3"	2960 x 1440	Android 10
Pixel 2	5"	1920 x 1080	Android 8
Pixel 3 XL	6.3"	2960 x 1440	Android 9
Samsung Galaxy S10	5.8"	2280 x 1080	Android 9
Samsung Galaxy Note 8	6.3"	1440 x 2960	Android 7.1.1

Fig 31: Testing Environments

Fig 31 explicitly shows how the application behaves when tested against different phones with different screen sizes and different versions of Android. Since The application was designed with smartphones as the target platform, it has not been tested on other kind of devices like SmartTVs or SmartWatches. The last two options were considered but were not included due to no access to such devices and, most importantly, no time.

3.4.4 Review

While acceptance and personal testing within different environments provided the most rich and helpful insight into how the application behaves, unit testing has fallen short because the features applicable were replaced with Regular Expressions checking the validity of the inputs. Since the project already went through acceptance testing, unit testing became somewhat obsolete.

4. Evaluation

4.1 Potential Risks

4.1.1 Project

In addition to the risks described in Appendix A, another risk that could be added to the list is not adhering and following the time plan due to inaccuracies in predicting the necessary time needed to complete the tasks, other work commitments and, as it turned out, a viral outbreak. Due to the viral outbreak, which was highly unlikely but still happened, the development of the project was considerably delayed. A different risk would be the risk of spending more than needed time tweaking features to the last detail before the main key features would be implemented in the application.

4.1.2 Product Design and Deployment

Since the risk of not implementing all proposed features that were initially planned was high, extra time was booked for features that have been predicted as holding a higher difficulty in order to allow enough time for extra research when needed. If the time proved to still be insufficient, in this case, the developer made realistic cuts to consider an incomplete or half developed feature.

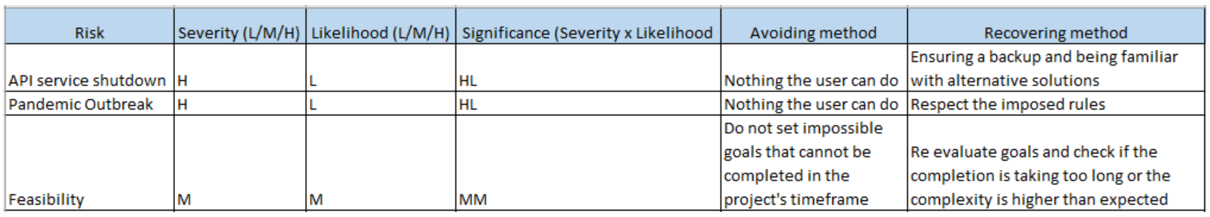
A high-risk concerning product design was that the application users will not feel comfortable with navigating through the software and will fail to use the application in the way it was designed to. This risk was overcome by making sure that, in terms of Application Design, the Android Developer rules have been adhered and respected. The result of this would be a natural feel and familiarity for the regular user when using the application.

4.2 User Data

The security of stored user data was another concern. This project uses Firebase database to store data and implicitly Firebase authentication as previously discussed. Because it does not use a local database to store any kind of data, the security concerns are somewhat lower. While in a real life scenario the sensitive user data should only be accessed by the user itself, at the moment the read/write permissions are open and will remain like this because this project is supposed to be a working prototype project which is not intended to be released for public use hence user data will only be accessed by the developer.

4.3 Risk Analysis

In addition to the risks already predicted and listed in the Appendix A, some risks were established during the process of implementation. Below is an example of risks that were not initially thought of but added later to the list.



Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Severity x Likelihood	Avoiding method	Recovering method
API service shutdown	H	L	HL	Nothing the user can do	Ensuring a backup and being familiar with alternative solutions
Pandemic Outbreak	H	L	HL	Nothing the user can do	Respect the imposed rules
Feasibility	M	M	MM	Do not set impossible goals that cannot be completed in the project's timeframe	Re evaluate goals and check if the completion is taking too long or the complexity is higher than expected

4.4 Project Achievements

By evaluating the final time plan for this project in comparison to how the actual implementation of the project developed and considering the current public health circumstances, the project turned out to be a success. With the estimation regarding the total project completion being severely affected by reasons out of developer’s control, realistically, all the initially proposed project features were implemented successfully.

However, when it comes to discussing the following of the timeframe dedicated to certain aspects of the application, inaccuracies can be found mainly due to current state of public health created by the Covid-19 outbreak. Another reason is not having knowledge about the complexity of a feature before implementing it and making inaccurate prediction regarding the time needed for it to be developed. For example, the implementation of Maps Activity lasted longer than expected due to the low knowledge about how Google Maps API works, and the time needed to be taken for research into the field. This kind of situation made the feature implementation to follow a rather erratic pattern.

In terms of consistency, the most time that was spent on development was during the Christmas period and during the summer months when commitments from other areas were minimal.

Since the developer has never taken and completed a project of this magnitude, in combination with limited knowledge of the platform and of the APIs, prioritizing workload and good time management were crucial factors that drove the success of the project. Extensive research, creating weekly objectives and guidance from the outside were major factors influencing the progress of the project during crucial periods like sessions and exams when workload reached its peak.

4.5 Objective Evaluation

4.5.1 Objective 1 – The GUI

Extensive research of the official documentation provided by Android regarding the creation of a UI greatly helped towards building an UI that adheres to the developer's guidelines. The UI is fully functional and since it uses a Relative Layout it allows scalability towards multiple screen sizes and resolutions. The colour scheme is a uniform one, making sure that different pages of the UI look similar to the previous ones. The arrangement in page is as simple as possible to enable ease of understanding and ease of use. All input boxes contain hints and are regulated by input validation methods in order to help the user and avoid any errors that might occur.

Percentage Completed: 90%

4.5.2 Objective 2 – Registration, Authentication and Authorisation

Google's Firebase Authentication made this objective's completion easier than expected. Firebase Authentication has full documentation and step by step setup available. Also, Firebase is integrated with Android Studio making the task even easier to complete. Users can successfully register an account and log into the application. The application successfully decides whether a user is admin or a regular user and initiates the correct activities created for each user type. The application securely stores all user data in the database. The application logs out users from the database whether they choose to press the logout button or just close the application.

Percentage Completed: 100%

4.5.3 Objective 3 – The GPS tracking service

The GPS tracking service uses Google Maps API. Studying and learning official documentation regarding Google Maps API made the completion of this task possible. When a user logs into the app, their location is checked and transferred to the database where it will sit waiting either for a change, a value query or deletion. The feature successfully implements listener methods in order to constantly track a user's location. The Admin can check the location of the users, at any time, just by pressing one button.

The feature successfully extracts and deposits information into the database, tracks a user's location and executes the correct measurements when a tour is ongoing.

A problem that occurred during the development of this objective is the fact that Android Studio's emulation service does not automatically emulate GPS tracking. The developer spent a significant amount of time trying to fix this but decided to use a real device in the end.

Percentage Completed: 100%

4.5.4 Objective 4 – The Database

With the help of Firebase Database, the project has successfully completed this task. Choosing to use Firebase instead of a custom database made things so much simpler that the time allocated for developing the database was cut in half. The application would successfully send and receive information to and from the database using a JSON format. The database is, in theory, split into four parts.

First part would be the Location Info where the tour objectives names are saved as key variables to which a description about the objectives, location and order are attached as a string variable.

Second part of the database is the part where user's information is stored. As soon as a user logs in, the application sends that user's information to the database where it is stored. Some information regarding the user, like the user's current location is saved in two variables, latitude and longitude, and it`s only being stored while the user is logged in. As soon as the user presses the log out button, the database erases the location information.

The third part is the part used for tracking the location of logged users. As soon as a user logs in, the information about their location along with the username is sent to the database and saved in a different part called "Logged Users". This is to avoid confusion regarding who is logged in and active and who is not. As soon as the user logs out, the database removes this information.

The fourth part is the chat part which is updated dynamically depending on user's chat activity. The database keeps track of all the messages and a manual deletion of the history if required. Only an admin with access to the database can perform this action.

Percentage Completed: 100%

4.5.5 Objective 5 – The communication channel

This objective is basically a chat feature that uses Firebase database and Codelab's FriendlyChat. Initially it was thought to be implemented as some sort of popup or notification feature but, after serious consideration, developing a full functional chat seemed like the better option. For example, if an admin sends a general urgent message to the users to cancel the tours due to security concerns, using the feature as a chat, enables the users to answer back and update the admins with their status. Or if an accident would happen while the tour is active, the users can immediately inform the supervisors and maybe measures could be taken more rapidly.

User's authentication is done through Firebase's authentication while message handling is performed by listeners that upload or download the typed text to and from the database into a chat layout. The database will keep all massages stored, for future reference, until an Admin decides they are not needed any longer and deletes them.

Successful completion of this task was made possible by researching into various documentation about chat applications, studying some of the available chat application online and allocating enough time for all these to happen.

While the task was fully completed per project spec and scope, additional optional features could have been added, like adding attachments options to the chat. The methods for this option have been written but not implemented.

Percentage Competed: 100%

4.5.6 Secondary Objectives

Initially, this task was meant to be used by an Admin by manually logging into the database and updating information as necessary, but, after consideration, it was decided that a more logical and easier to use solution would be to implement an activity that would allow the admin to communicate with the database directly from their mobile phone or tablet.

This task was successfully implemented, and it was one of the quickest tasks when it comes to developing time.

Percentage Completed: 100%

4.6 Refinements and Additions

4.6.1 – User Interface

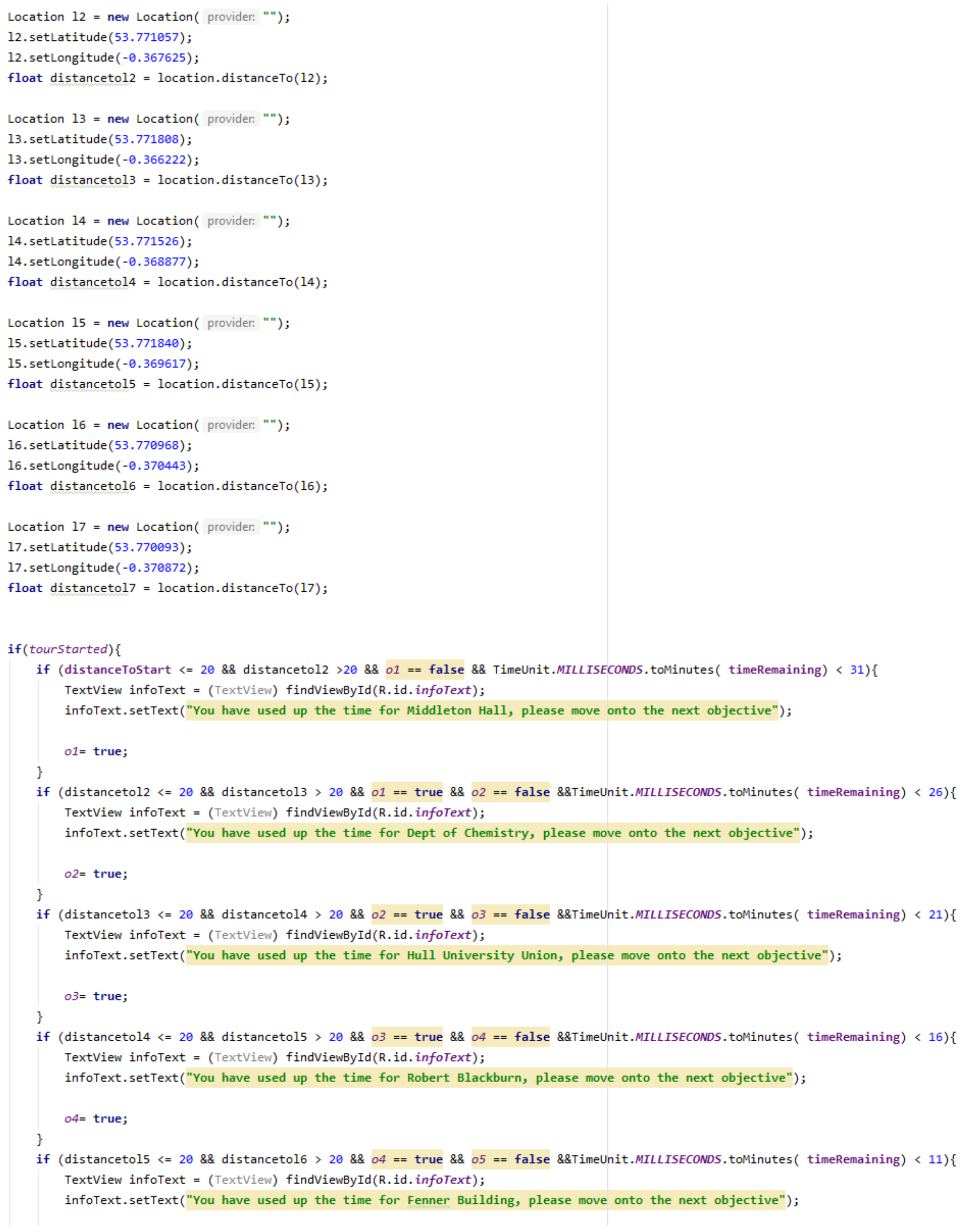
One of the refinements would be an improved user interface. The current interface serves the project well and accomplishes all the needed conditions described in Android but lacks the niche refinements that would make the UI not good but excellent. Custom images and graphics could be used to make the application more appealing to the public but since this application is not intended to be released and it's merely a working prototype, the extra mile was not needed.

4.6.2 – The GPS tracking system

The GPS tracking system could be further developed and eventually changed into Google Maps Directions using Google Maps not only as it is used in this app, for the map layout, but also taking advantage of the navigation system. Unfortunately, the timeframe allocated for the project is not enough for such a major upgrade. It would imply extensive research into this functionality and time is simply short.

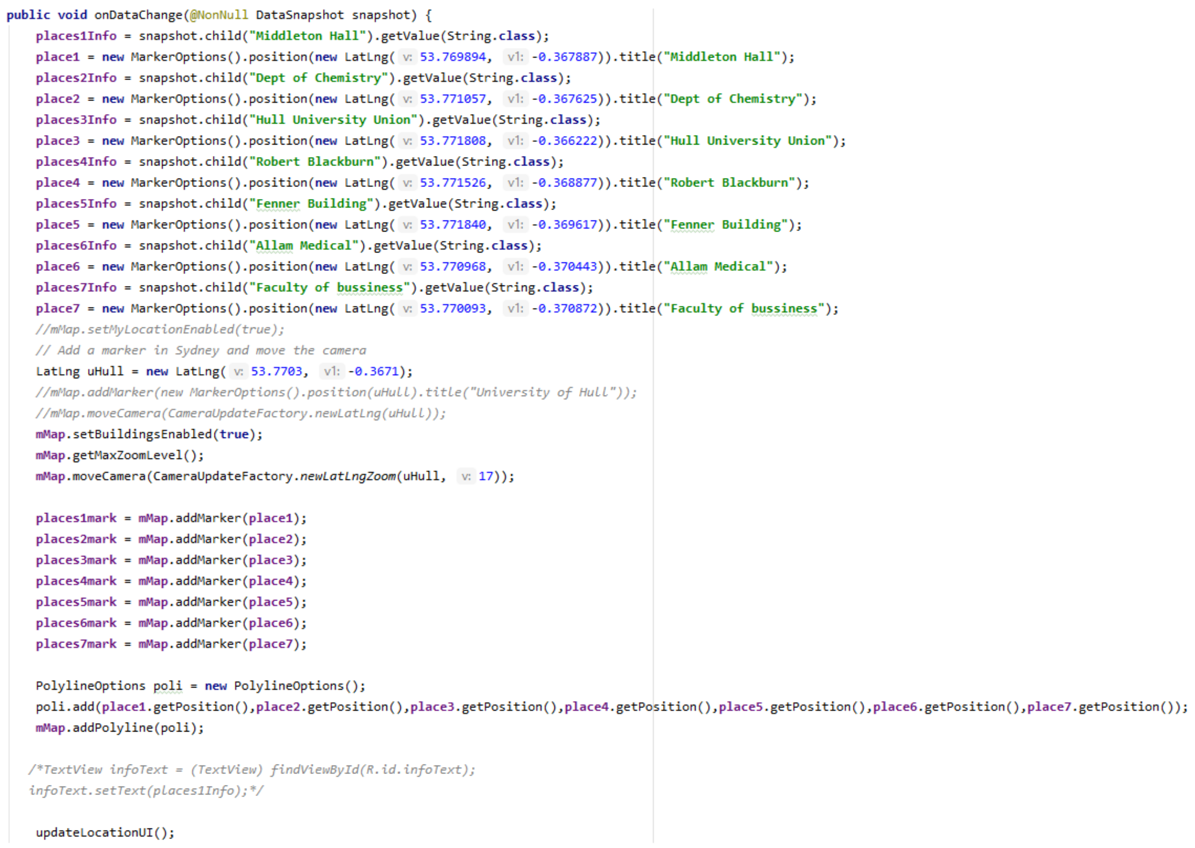
Another refinement or improvement would be to be able to track the user's location, live on the admin side. Now it requires a button to be pressed in order to update but, as mentioned above, extensive research and more time would be needed for the transition to take place.

A major refinement which has been successfully completed is with regards to the way objectives are placed on the map and the way the user is tracked and alerting service works. Initially, the objective locations and information about them have been hardcoded into the application (**Fig 34, Fig 35, Fig 36**). This was a major inconvenience in case an admin would want to add or remove any objectives from the database. It would require a developer with access to the source code in order to modify the objectives list. After consideration, the methods have been rewritten and the logic had to be thought through from zero. Explanation on how the functions work has been given above in the allocated paragraph with screenshots for support. Because of this change all the other depending functions and activities had to be rewritten, Admin Activity including. Maps activity received a major overhaul. Please see below how some of the function looked before.



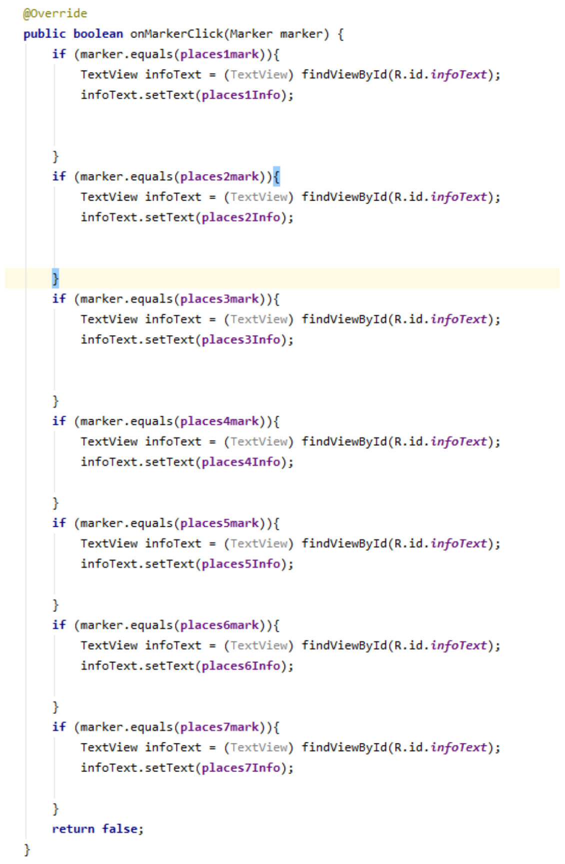
```
Location l2 = new Location( provider: "");
l2.setLatitude(53.771057);
l2.setLongitude(-0.367625);
float distancetol2 = location.distanceTo(l2);

Location l3 = new Location( provider: "");
l3.setLatitude(53.771808);
l3.setLongitude(-0.366222);
float distancetol3 = location.distanceTo(l3);

Location l4 = new Location( provider: "");
l4.setLatitude(53.771526);
l4.setLongitude(-0.368877);
float distancetol4 = location.distanceTo(l4);

Location l5 = new Location( provider: "");
l5.setLatitude(53.771840);
l5.setLongitude(-0.369617);
float distancetol5 = location.distanceTo(l5);

Location l6 = new Location( provider: "");
l6.setLatitude(53.770968);
l6.setLongitude(-0.370443);
float distancetol6 = location.distanceTo(l6);

Location l7 = new Location( provider: "");
l7.setLatitude(53.770093);
l7.setLongitude(-0.370872);
float distancetol7 = location.distanceTo(l7);


if(tourStarted){
    if (distanceToStart <= 20 && distancetol2 >20 && o1 == false && TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < 31){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You have used up the time for Middleton Hall, please move onto the next objective");

        o1= true;
    }
    if (distancetol2 <= 20 && distancetol3 > 20 && o1 == true && o2 == false &&TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < 26){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You have used up the time for Dept of Chemistry, please move onto the next objective");

        o2= true;
    }
    if (distancetol3 <= 20 && distancetol4 > 20 && o2 == true && o3 == false &&TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < 21){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You have used up the time for Hull University Union, please move onto the next objective");

        o3= true;
    }
    if (distancetol4 <= 20 && distancetol5 > 20 && o3 == true && o4 == false &&TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < 16){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You have used up the time for Robert Blackburn, please move onto the next objective");

        o4= true;
    }
    if (distancetol5 <= 20 && distancetol6 > 20 && o4 == true && o5 == false &&TimeUnit.MILLISECONDS.toMinutes( timeRemaining) < 11){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText("You have used up the time for Fenner Building, please move onto the next objective");
```

Fig 34: Before 1



```
public void onDataChange(@NonNull DataSnapshot snapshot) {
    places1Info = snapshot.child("Middleton Hall").getValue(String.class);
    place1 = new MarkerOptions().position(new LatLng( v: 53.769894,  v1: -0.367887)).title("Middleton Hall");
    places2Info = snapshot.child("Dept of Chemistry").getValue(String.class);
    place2 = new MarkerOptions().position(new LatLng( v: 53.771057,  v1: -0.367625)).title("Dept of Chemistry");
    places3Info = snapshot.child("Hull University Union").getValue(String.class);
    place3 = new MarkerOptions().position(new LatLng( v: 53.771808,  v1: -0.366222)).title("Hull University Union");
    places4Info = snapshot.child("Robert Blackburn").getValue(String.class);
    place4 = new MarkerOptions().position(new LatLng( v: 53.771526,  v1: -0.368877)).title("Robert Blackburn");
    places5Info = snapshot.child("Fenner Building").getValue(String.class);
    place5 = new MarkerOptions().position(new LatLng( v: 53.771840,  v1: -0.369617)).title("Fenner Building");
    places6Info = snapshot.child("Allam Medical").getValue(String.class);
    place6 = new MarkerOptions().position(new LatLng( v: 53.770968,  v1: -0.370443)).title("Allam Medical");
    places7Info = snapshot.child("Faculty of bussiness").getValue(String.class);
    place7 = new MarkerOptions().position(new LatLng( v: 53.770093,  v1: -0.370872)).title("Faculty of bussiness");
    //mMap.setMyLocationEnabled(true);
    // Add a marker in Sydney and move the camera
    LatLng uHull = new LatLng( v: 53.7703,  v1: -0.3671);
    //mMap.addMarker(new MarkerOptions().position(uHull).title("University of Hull"));
    //mMap.moveCamera(CameraUpdateFactory.newLatLng(uHull));
    mMap.setBuildingsEnabled(true);
    mMap.getMaxZoomLevel();
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(uHull,  v: 17));

    places1mark = mMap.addMarker(place1);
    places2mark = mMap.addMarker(place2);
    places3mark = mMap.addMarker(place3);
    places4mark = mMap.addMarker(place4);
    places5mark = mMap.addMarker(place5);
    places6mark = mMap.addMarker(place6);
    places7mark = mMap.addMarker(place7);

    PolylineOptions poli = new PolylineOptions();
    poli.add(place1.getPosition(),place2.getPosition(),place3.getPosition(),place4.getPosition(),place5.getPosition(),place6.getPosition(),place7.getPosition());
    mMap.addPolyline(poli);

   /*TextView infoText = (TextView) findViewById(R.id.infoText);
   infoText.setText(places1Info);*/

    updateLocationUI();
```

Fig 35: Before 2



```
@Override
public boolean onMarkerClick(Marker marker) {
    if (marker.equals(places1mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places1Info);


    }
    if (marker.equals(places2mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places2Info);


    }
    if (marker.equals(places3mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places3Info);


    }
    if (marker.equals(places4mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places4Info);

    }
    if (marker.equals(places5mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places5Info);

    }
    if (marker.equals(places6mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places6Info);

    }
    if (marker.equals(places7mark)){
        TextView infoText = (TextView) findViewById(R.id.infoText);
        infoText.setText(places7Info);

    }
    return false;
}
```

Fig 36: Before 3

4.6.3 – The chat

While some of the methods for implementing file and picture transfers have been written, the extra, unplanned feature has not been fully developed.

Further improvements would be adding pictures attached to the users in chat and making so that the messages displayed on left and right side of the screen depending if it's user's own message or other users' messages.

While initially thought to be designed as a one-way communication feature, meaning the admin would be able to send messages to the users and not the other way around, the development led to a nice addition of a fully working chat feature for the application.

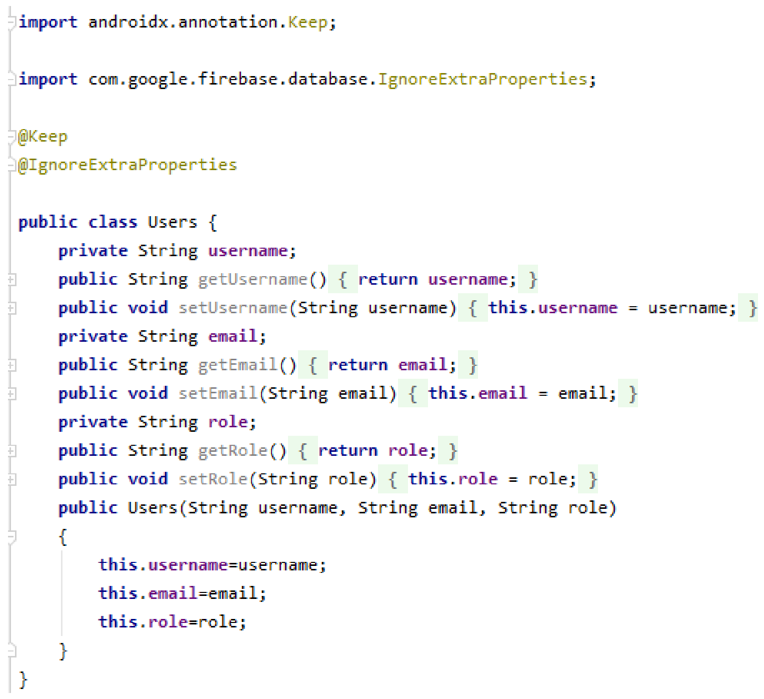
4.6.4 – More functionality to the Admin Activities

The secondary proposed objective has been fully completed but it would have been nice if the admin would have the possibility to have the users logged in displayed in a list and would be able to see the entries about objectives directly from the application. While these tasks would not require a huge amount of time to be spent on, they are unnecessary when looking at the project's scope.

This objective had to be updated due to Maps Activity receiving a major overhaul. Extra fields have been added. The admin can now also manually input the latitude and longitude of the objective and save it into the database, information which will be used by the application to dynamically build an objective map. Validation has been added to all the fields on the page, see below (**Fig 37**).

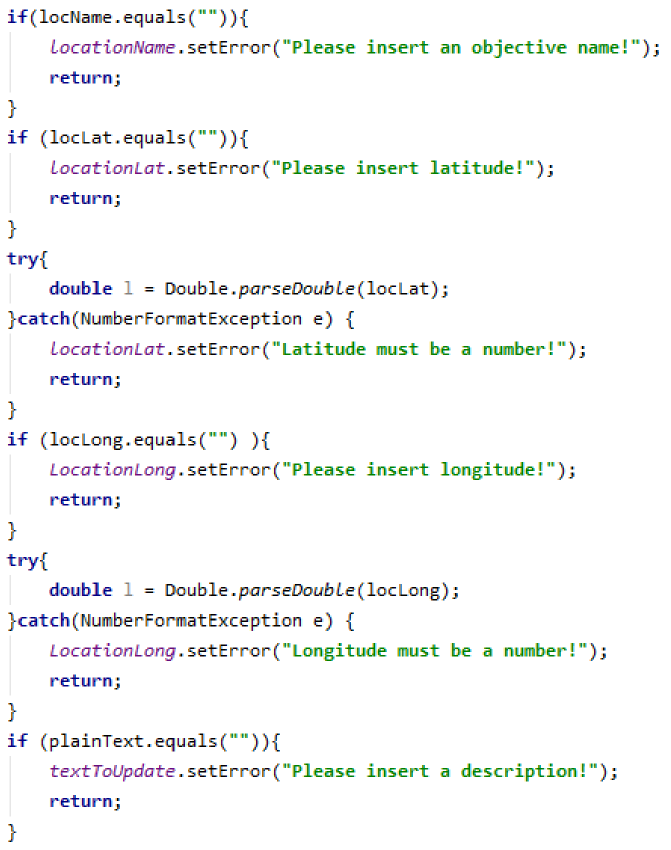
4.6.4 - ProGuard

ProGuard is a tool that optimizes, shrinks and obfuscates when building for release. Due to it meddling with the code, the application would behave differently in developer mode from release mode. For example, at account creation, the user data would be saved differently in the database and not in the designed format. ProGuard had to be configured to keep the original format of the targeted classes by adding @Keep and @IgnoreExtraProperties (**Fig 38**).



```
import androidx.annotation.Keep;

import com.google.firebase.database.IgnoreExtraProperties;

@Keep
@IgnoreExtraProperties

public class Users {
    private String username;
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    private String email;
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    private String role;
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
    public Users(String username, String email, String role)
    {
        this.username=username;
        this.email=email;
        this.role=role;
    }
}
```

Fig 38: ProGuard



```
if(locName.equals("")){
    locationName.setError("Please insert an objective name!");
    return;
}
if (locLat.equals("")){
    locationLat.setError("Please insert latitude!");
    return;
}
try{
    double l = Double.parseDouble(locLat);
}catch(NumberFormatException e) {
    locationLat.setError("Latitude must be a number!");
    return;
}
if (locLong.equals("") ){
    LocationLong.setError("Please insert longitude!");
    return;
}
try{
    double l = Double.parseDouble(locLong);
}catch(NumberFormatException e) {
    LocationLong.setError("Longitude must be a number!");
    return;
}
if (plainText.equals("")){
    textToUpdate.setError("Please insert a description!");
    return;
}
```

Fig 37: Admin Validation

4.6.5 – The Registration

The registration has been updated with a second password input field used to check against the first password field in order to make sure that the registering user does not make a typo in the password he wants to set.

While at first the username field would go without checking, the code has been updated in such a way that if the user does not choose an username or leaves it as default “Username” the application will split the email provided and use the first part of the email, before “@”, as the username.

5 Conclusion

Ultimately, the development of the project has been successful, all proposed objectives have been met, with extra features added along the way.

The main features presented in the project are:

- Online database
- Authentication and Authorisation
- User Accounts
- GPS tracking system
- Dynamic information extraction and insertion into the database
- Chat functionality
- Timer functionality
- Tour objectives information
- Alerting service

The proposed methodology was initially set to be represented by the waterfall model but, due to circumstances that could not be controlled, the methodology shifted towards the Agile approach.

5.1 Learning Outcomes

The development of the project has been very challenging, engaging, enriching, fun and annoying at the same time. Challenges have been countered by ambition and perseverance and have been overcome. While Java was not one of the developer’s skillsets, having priory learned C#, helped immensely with learning and understanding of the new programming language.

Some of the skills that have been picked up or improved upon during the development process of this project:

- Java
- XML
- Google’s Firebase
- JSON / Parsing
- Android Studio
- API implication and usage
- Time management
- Report writing

6 References

Android Developers. 2020. *Terms And Conditions | Android Developers.* [online] Available at: https://developer.android.com/studio/terms [Accessed 3 March 2020].

StatCounter Global Stats. 2020. *Mobile Operating System Market Share Worldwide | Statcounter Global Stats.* [online] Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed 6 March 2020].

Microsoft. 2020*. Xamarin | Open-Source Mobile App Platform For .NET.* [online] Available at: https://dotnet.microsoft.com/apps/xamarin [Accessed 13 March 2020].

Android Authority. 2020. *HTML5 Vs Native Android App.* [online] Available at:

https://www.androidauthority.com/html-5-vs-native-android-app-607214/ [Accessed 23 March 2020].

Docs.parseplatform.org. 2020. *Parse.* [online] Available at: https://docs.parseplatform.org/parse-server/guide/ [Accessed 4 May 2020].

Google Developers. 2020. *Overview | Places API | Google Developers.* [online] Available at: https://developers.google.com/places/web-service/intro [Accessed 16 May 2020].

M. and Boyle, M., 2020. *Mobile Internet Statistics 2019 | Finder UK*. [online]

Available at: https://www.finder.com/uk/mobile-internet-statistics [Accessed 23 May 2020].

Statista. 2020. *Google Play Store: Number Of Apps 2019 | Statista.* [online] Available at: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ [Accessed 8 April 2020].

Oracle.com. 2020. *Code Conventions For The Java Programming Language: 9. Naming Conventions.* [online] Available at: https://www.oracle.com/technetwork/java/codeconventions-135099.html [Accessed 21 February 2020].

En.wikipedia.org. 2020. *Windows Phone.* [online] Available at:

https://en.wikipedia.org/wiki/Windows_Phone [Accessed 2 May 2020].

Support.microsoft.com. 2020. *End Of Support.* [online] Available at: https://support.microsoft.com/en-us/help/4485197/windows-10-mobile-end-of-support-faq [Accessed 3 May 2020].

Statista. 2020. *Time Spent On Smartphones In The UK 2017 | Statista.* [online] Available at:https://www.statista.com/statistics/717110/smartphones-daily-time-spent-using-phone-in-the-uk/ [Accessed 18 May 2020].

Metev, D., 2020. *39+ Smartphone Statistics You Should Know In 2020*. [online] Review42. Available at: https://review42.com/smartphone-statistics/ [Accessed 14 July 2020].

Beccari, D., 2011. *Five Big Problems With Mobile Travel Applications (And A Bonus One For Android Users) | Phocuswire*. [online] Phocuswire.com. Available at: https://www.phocuswire.com/Five-big-problems-with-mobile-travel-applications-and-a-bonus-one-for-Android-users [Accessed 20 June 2020].

L, H., 2015. *Why, When, And How To Use The Google Map API*. [online] Medium. Available at: https://medium.com/@helennnsays/why-when-and-how-to-use-the-google-map-api-f5dfa35986dc [Accessed 8 February 2020].

Amin, N., 2019. *10 Best Programming Languages For Mobile App Development - Topofstack Software Limited*. [online] TopOfStack Software Limited. Available at: https://topofstacksoftware.com/2019/01/10-best-programming-languages-for-mobile-app-development/ [Accessed 10 April 2020].

Sinicki, A., 2020. *I Want To Develop Android Apps — What Languages Should I Learn?*. [online] Android Authority. Available at: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/ [Accessed 4 April 2020].

GeeksforGeeks. 2020. *Top Programming Languages For Android App Development - Geeksforgeeks*. [online] Available at: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/ [Accessed 12 May 2020].

India, Y., 2020. *5 Best Free IDE's To Speed Up Android App Development*. [online] Blog - Your Team in India. Available at: https://www.yourteaminindia.com/blog/best-free-ide-android-app-development/ [Accessed 8 July 2020].

ADMEC Multimedia. 2020. *Top 10 Features Of Android Studio - ADMEC Multimedia*. [online] Available at: https://www.admecindia.co.in/miscellaneous/top-10-features-android-studio-developers-not-miss/ [Accessed 13 December 2019].

AltexSoft. 2018. *The Good And The Bad Of Android App Development*. [online] Available at: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-android-app-development/ [Accessed 14 December 2019].

Williams, J., 2019. *Features Of Xamarin Platform For Hybrid App Development*. [online] Medium. Available at: https://medium.com/@justinwilliams5444/features-of-xamarin-platform-for-hybrid-app-development-4dfc4e24b724 [Accessed 5 January 2020].

Android Authority. 2020. *What Is Android? Here's Everything You Need To Know*. [online] Available at: https://www.androidauthority.com/what-is-android-328076/ [Accessed 17 December 2019].

Esplin, C., 2016. *What Is Firebase?*. [online] Medium. Available at: https://howtofirebase.com/what-is-firebase-fcb8614ba442 [Accessed 17 January 2020].

Oracle.com. 2020. *What Is A Database?*. [online] Available at: https://www.oracle.com/in/database/what-is-database.html [Accessed 14 December 2019].

Miller, J., 2016. *What Are The Advantages And Disadvantages Of Using Parse As A Mobile App Backend? - Quora*. [online] Quora.com. Available at: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-Parse-as-a-mobile-app-backend [Accessed 14 February 2020].

Bush, T., 2018. *5 Powerful Alternatives To Google Maps API | Nordic Apis |*. [online] Nordic APIs. Available at: https://nordicapis.com/5-powerful-alternatives-to-google-maps-api/ [Accessed 15 February 2020].

Sudarma, A., 2013. *(PDF) Design And Implementation Of Geographic Information System On Tourism Guide Using Web-Based Google Maps*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/272566787_Design_and_Implementation_of_Geographic_Information_System_on_Tourism_Guide_Using_Web-Based_Google_Maps [Accessed 10 June 2020].

Rehan, A., 2020. *12 Best Geocoding And Maps API Solution For Your Applications - Geekflare*. [online] Geekflare. Available at: https://geekflare.com/geocoding-maps-api-solution/ [Accessed 16 June 2020].

Habr.com. 2019. *Top 10 Chat, Audio & Video Calling API & SDK Providers For Enterprise Business*. [online] Available at: https://habr.com/en/post/453374/ [Accessed 6 June 2020].

Agile Alliance. 2020. *What Is Acceptance Testing?*. [online] Available at: https://www.agilealliance.org/glossary/acceptance/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'acceptance*20test))~searchTerm~'~sort~false~sortDirection~'asc~page~1) [Accessed 12 June 2020].

Firebase. 2020. *Firebase Realtime Database*. [online] Available at: https://firebase.google.com/docs/database [Accessed 14 April 2020].

Google Developers. 2020. *Google Maps Platform | Google Developers*. [online] Available at: https://developers.google.com/maps/documentation [Accessed 16 March 2020].

Codelabs.developers.google.com. 2020. *Firebase Android Codelab - Build Friendly Chat*. [online] Available at: https://codelabs.developers.google.com/codelabs/firebase-android/#9 [Accessed 9 March 2020].

7 Appendices

7.1 Updated Risk Table – Appendix A

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Severity x Likelihood	Avoiding method	Recovering method
API service shutdown	H	L	HL	Nothing the user can do	Ensuring a backup and being familiar with alternative solutions
Pandemic Outbreak	H	L	HL	Nothing the user can do	Respect the imposed rules
Feasibility	M	M	MM	Do not set impossible goals that cannot be completed in the project's timeframe	Re evaluate goals and check if the completion is taking too long or the complexity is higher than expected
Code Loss	H	L	HL	Use Source Control	Source Control
Mental Health	L	M	ML	Keeping on Top	Visit a specialist
Obsolete API	H	L	HL	Check updated API	Use updated API
Skill risk (Not having previous experience)	M	M	MM	Conduct research into skills needed prior to development.	Assign extra time to learn skills needed for the project.
UI not compatible with screen resolution	L	M	LM	Create icons for all types of screen DPI	Redesign icons or use standard icons but be aware of reduced quality for larger resolutions.
Inconsistent UI	M	L	ML	Make sure to use similar fonts and follow consistent design patterns with buttons, logo etc.	Redesign UI and change minor things such as fonts or colour schemes or follow a built-in template if time constraints.
External Database Legal issues	H	L	HL	Make sure to read the terms and conditions of the company that will be providing the database and check if it can be implemented.	Find another database online that will accommodate the app legally.
Fail to meet Time Plan Deadline for Task	M	H	MH	Keep working to schedule and if needed put in extra work to try and meet deadline on time.	Put it extra time to try to meet Task taking into account the priority, if it is an extra feature and Key tasks still need implementing, move on.
Physical Health	M	M	MM	Sleep and Eat healthy	See the GP

7.2 Acceptance Test Table – Appendix B

Test name	Module	Conclussion
White Space checker on input boxes	All	Pass
Password format is acceptable	Registration/Login	Pass
Account Creation Works	Registration	Pass
Email Format is acceptable	Registration	Pass
Correct account data is saved to the database	Registration	Pass
Correct info at log in page	Login	Pass
Login is functional	Login	Pass
On screen back button exits the application if the previous page was login	All	Pass
Pressing back button does not cause the application to forget state if the previous page is not the login page	All	Pass
App is functional on multiple devices with different display types	All	Pass
Landscape mode disabled works as intended	All	Pass
Markers are populated correctly	Maps Activity	Pass
Information about objectives is correctly extracted and displayed	Maps Activity	Pass
The tour starts only when correct conditions are met	Maps Activity	Pass
Start Tour button is disabled once the tour has started	Maps Activity	Pass
Start Tour button is enabled once the tour has finished	Maps Activity	Pass
Guide receives notifications if he is late	Maps Activity	Pass
Timer is displayed correctly once the tour starts	Maps Activity	Pass
Logout button functions correctly	All	Pass
Verify message button opens the correct activity	Maps Activity	Pass
User can send and receive message correctly	Chat Activity	Pass
On logout the databse is cleared of unnecessary information	All	Pass
On login, depending on authorisation, the user is sent to correct activity	Login	Pass
Admin map fragment loads correctly	CheckUserLocations	Pass
Admin can send and receive messages in chat	Chat Activity	Pass
Admin can update the database with correct info	Admin Activity	Pass
Admin can view markers for guide's location	CheckUserLocations	Pass
Messages are loaded correctly from the database	Chat Activity	Pass
Markers contain information about the currently logged users	CheckUserLocations	Pass
Markers updating the correct information from database, dynamically	Maps Activity	Pass