Real-time Location Tracking and Visualization using Augmented Reality

By

Aadam
Hamza Saeed

Department of Information Technology

University of Haripur

Haripur, Pakistan

September, 2018

Real-time Location Tracking and Visualization using Augmented Reality

By

Aadam, Hamza Saeed
6396, 6391

Supervised by

Mudassar Ali Khan

Department of Information Technology

University of Haripur

Haripur, Pakistan

September, 2018

Real-time Location Tracking and Visualization using Augmented Reality

By

Aadam, Hamza Saeed

A Dissertation Submitted in Partial Fulfillment for the Degree of

BACHELORS OF SCIENCE

IN

COMPUTER SCIENCE

Department of Information Technology

University of Haripur

Haripur, Pakistan

September, 2018

Real-time Location Tracking and Visualization using Augmented Reality

By

Aadam, Hamza Saeed

CERTIFICATE

A THESIS SUBMITTED IN THE PARTIAL FULFILMENT OF THE
REQUIRMENTS FOR THE DEGREE OF BACHELORS
IN COMPUTER SCIENCE

We accept this dissertation as conforming to the required standards

Mudassar Ali Khan Project Supervisor	Dr. Yousaf Saeed Final Year Projects Coordinator
External Examiner	Mr. Muhammad Junaid Head of Department

Department of Information Technology

University of Haripur

Haripur, Pakistan

September, 2018

Declaration

I hereby declare that this dissertation is the presentation of my original work. Wherever contributions of others are involved, every effort is made to indicate this clearly with due reference to the literature and acknowledgment of collaborative research and discussions.

This work was done under the guidance of Mr. Mudassar Ali Khan, Department of Information Technology, University of Haripur, Haripur.

Date: 14th Sep, 2018 Aadam, Hamza Saeed

Dedicated to

our most respectful and beloved parents.

Acknowledgment

All glories to Allah Almighty, the most merciful, the most benevolent, who bestowed upon us his blessings and gave us the strength to face the challenges in this life and guided us to choose the right path along the way.

This is all made possible due to the prayers and utmost support of our parents. No matter what difficulties they faced, they kept us unaware of the problems and supported us, both morally and financially.

Last but not least, it was because of the valuable guidance of our project supervisor, Mr. Mudassar Ali Khan, who despite his busy schedule, gave us his valuable time and tolerated our mistakes and motivated us to be our best selves.

Abstract

As we know, Man is a social animal. There is always the need to connect to one's loved ones and to easily find them, especially in this crowded world. Although, there are several Apps out there that provide the services to keep track of a user's location, none of them support Augmented Reality in their App for now. In this report, we propose a simple, cheap but efficient architecture that will keep track of a user's location at any point in time and show his current location to his contacts in real-time, using Augmented Reality.

Augmented Reality (AR) is a technique that is used to show virtual objects on the real-world images. By integrating Augmented Reality into a location tracking App, the App will allow users to see and navigate towards their desired destination markers in the real-world. Thus, users will be able to easily visualize where their contacts are and quickly pick them out in a crowded space.

The goal of this thesis is to devise a cheap way to keep track of a user's location in real-time and use it to show a contact's location using Augmented Reality technology. We also plan to include messaging functionality in the App so the users could easily communicate with one another during the navigation.

Table of Contents

List of Figures

Chapter 1

Introduction

In this chapter, we will give a basic overview of the problem that we are trying to solve and why we chose to solve this problem. We will start off by explaining the purpose of the project and the document. Then we will talk about what AR is and, how it is different from VR, how it is impacting the way users interact with the technology and how it is making it easier to access information without cluttering the workspace of the user. Then we will move on to how it could be used in Location Tracking systems to give a better understanding of the whereabouts of the user. At the end, we will give a brief introduction of the technologies that we will use in the implementation of our proposed system.

1.1 Purpose of the Project

Man is a social animal. We have a strong desire to connect with others, share our experiences with one another. We care deeply for our loved ones and want to keep track of them, to know where they are at a particular time. Parents want to keep track of their children, and friends want to find one another in crowds easily. It makes us feel more secure and closer to each other.

Augmented Reality (AR) is a type of technology that superimposes virtual objects onto the real world. Although there are a number of apps already out there that provide the

feature of user tracking, none of them is using Augmented Reality to give a better user experience. By using AR in a location tracking app, the user will be able to better visualize where and how far someone is at the current moment. Imagine that you are in a crowded space and you quickly want to find your friend in the crowd. With AR supported app, you would open up the app, and it would point out the location of your friend in the crowd by showing a marker at his location, in the real-world.

Our proposed system will be using GPS (Global Positioning System), along with other technologies like WiFi, Cellular network etc, to get the current location of the user and update it in the remote real-time database. Once a user's location is updated, it will be reflected to all the contacts of the user so they could easily see where that user is at the current moment. The system will use AR (augmented reality) along with Google Maps, for the visualization of a user's location.

1.2 Purpose of this Document

This document serves to be a report describing and explaining the process and the procedures taken to develop a tracking system that manages a user's contacts and their locations in real-time and, along with Google Maps, uses augmented reality for the visualization of their location at a particular time.

1.3 Overview of this Document

This report is divided into 5 chapters:

- Chapter 1 introduces the main idea of the project, the purpose of the project/document. It also briefly describes the technologies that are used in the proposed system.
- Chapter 2 describes the devised design of the software system, its architecture, different modules involved; data flow throughout the system, data schema for the database,

and some design and implementation constraints.

- Chapter 3 gives the implementation details of the software system and the problem that we encountered during the development.
- Chapter 4 evaluates the achievements of this project and if it achieved its objectives.
- Chapter 5 gives the conclusion of the project as well as some guidelines to further develop and improve the project in future.

1.4 Objectives

The core objectives of the project are:

- Design and implement a system that tracks a user location and stores it in a remote database in real-time.
- Implement some social features so that a user can add other users into his contact list and keep track of their location at all times.
- Add real-time chatting functionality in the application for easier communication between users when navigating.
- Use Augmented Reality for the visualization of the user marker in the real-world at any specific interval.
- Add in-app chat-bot that provides guidance and assistance to the user in using the app effectively.
- The devised system should be economical, for the end-user as well as for the developers.

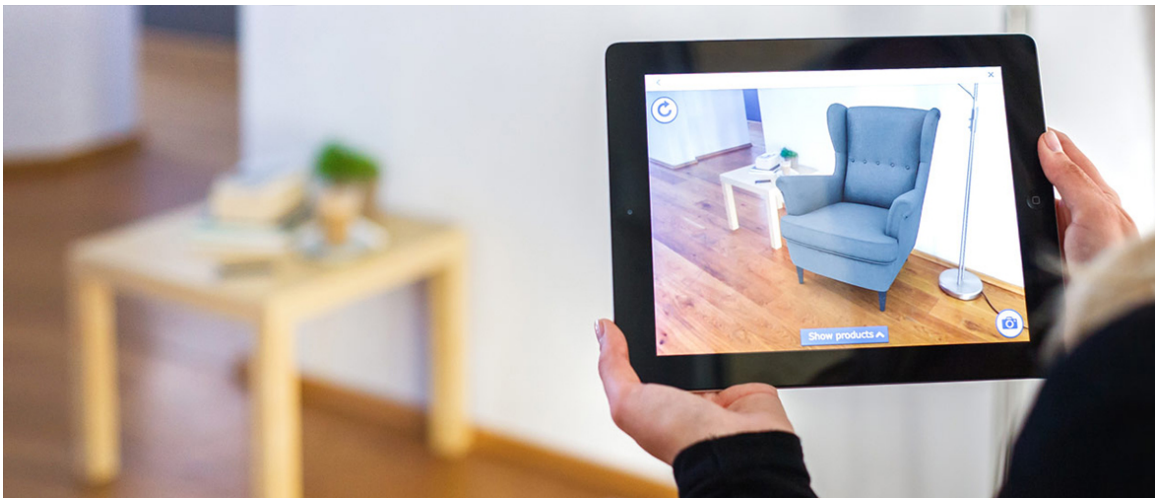

Figure 1.1: AR example with virtual sofa

1.5 Augmented Reality

Augmented Reality (AR) is a type of technology that is used to overlay virtual graphical content onto the real world [1]. It is a variation of Virtual Reality (VR) but instead of replacing the real world completely, as, in VR, it augments or supplements the real-world by superimposing virtual content onto the real world. In an ideal scenario, the user will not be able to tell the difference between the virtual and real objects in the same space. Figure 1.1 shows an example of such a scene where a virtual sofa and a lamp is superimposed on a real-world space, augmenting the experience of the user to show him what it would feel like if there really was a sofa like that present there.

1.5.1 Motivation

We chose to use Augmented Reality in our application because of its utility and effectiveness in conveying the information, which is not readily available to a user in a real-world scenario, without secluding him from the reality.

Although Virtual Reality technology has its own benefits, for example, in providing an immersive experience which could be used for educational, tourism, simulation purposes,

Figure 1.2: Augmented Reality application in Medical field

etc., the main drawback of VR is that it completely secludes the user from real-world. In contrast, the AR technology does not remove the user from the real world, but it augments the real-world around the user in order to provide him with extra information that he could not detect with his senses. This dramatically increases the utility of the AR technology as it could be used for numerous applications without affecting the work-flow of the user [2].

1.5.2 Applications

Every interaction technology has its own benefits and applications. Although touch screens are quite easy and effective to use, they are not suitable for usage in every environment. Similarly, text-based interfaces are pretty outdated, but still, there are still some cases where they turn out to be the best way to interact with the technology. Here we will briefly describe some of the applications where it is more suitable to use Augmented Reality as it could greatly increase the usability and user experience [3].

1.5.2.1 Medical

Augmented Reality could be used by Doctors as a visualization and training aid for surgery, as shown in Figure 1.2. A novice surgeon could consult with a highly trained surgeon. Virtual instructions could remind him of the required steps, without the need to look away from the patient or consult a manual and all this could be done remotely without the need of the physical presence of the professional there. In her TED talk [4], Nadine Hachach-Haram showed how AR is currently revolutionizing the medical industry and how this could potentially benefit those with low infrastructural and educational facilities.

1.5.2.2 Manufacturing and Repair

Another application of Augmented Reality is the assembly, maintenance and repair of complex machinery as the instructions might be easier to understand if, instead of reading manuals, it was available as 3D drawings with step by step instructions as to how to perform some task, as shown in Figure 1.3.

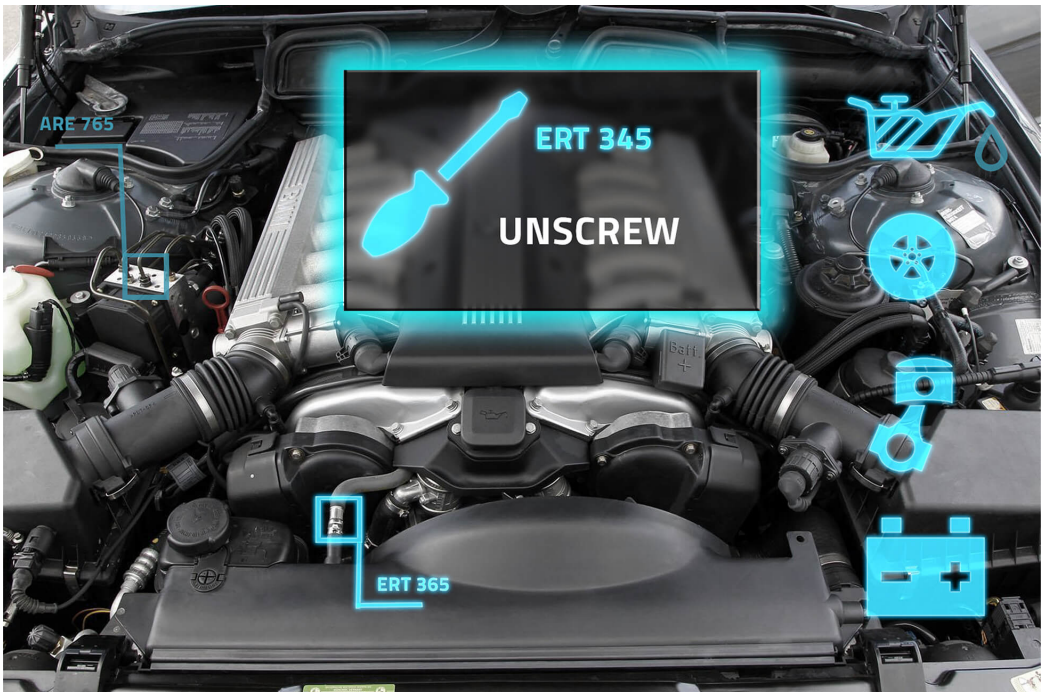

Figure 1.3: AR usage in Engine Repairing

Figure 1.4: Annotation of Nearby Places in AR

1.5.2.3 Annotation and visualization

AR could also be used for annotations and visualization purposes, where it annotates an object with some relevant information, as shown in Figure 1.4. For this purpose, it would fetch information from some public database, or it could also be some privately maintained database, as is the case in our application. Our use-case for the current project falls under this category as we are annotating or visualizing the current location of the user in the real-world using Augmented Reality.

1.5.2.4 Navigation

Augmented Reality could also be used for navigation purposes where some markers of lines are superimposed onto the paths that the user needs to follow. Mapbox [5], as well as Google [6], worked on this problem to make it more efficient and reliable to work on World-scale AR. In Figure 1.5, Google revealed how their Google Maps apps would use

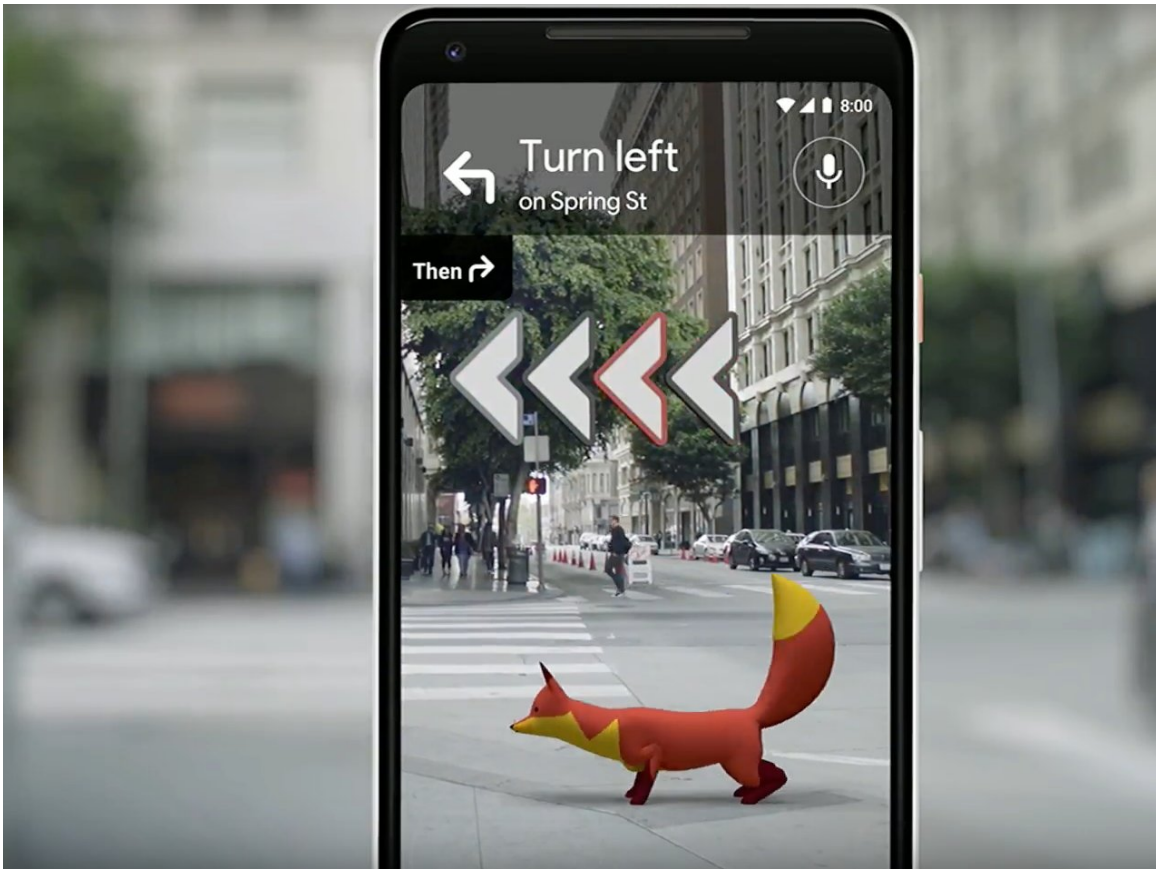

Figure 1.5: AR Navigation in Google Maps

Augmented Reality, both in Android as well as iOS, to help users navigate through the real world [6].

1.5.2.5 Entertainment

In the past couple of years, AR has been used extensively in this area, from Facebook and Snapchat Filters to games like Pokemon Go, as shown in Figure 1.6, it has helped a lot in publicity of Augmented Reality technology. With the advent of Mobile AR, numerous apps have been released that make use of this technology to entertain their users.

1.5.3 Mobile AR

There are some specific devices designed for AR usage, but they tend to be pricey and not yet economical and practical. Mobile AR is where AR technology is used on our

Figure 1.6: Use of AR in Pokemon Go

daily smart-phones, tablets, etc., instead of those specialized AR glasses and hardware [7]. There has been a surge in past couple of years in the usage of Mobile AR because of little to no investment on the part of end-users. Developers have been using it extensively to deliver new and unique experiences to their users, and tech-giants like Apple and Google are battling for the dominance in this area. There are over 2000 AR apps available in iOS Store and another 200-plus on Google Play [8].

Apple released their AR library, named ARKit [9], which was very well received by the developer's community. It was soon followed by ARCore [10], which was released by Google. Both libraries were focused on solving the same problem, introducing AR on a mobile platform and to dominate this area. As ARKit was released earlier, that is the reason behind a vast number of AR apps on iOS Store as compared to Google Play. Another thing is that ARCore can only run on some specific hardware devices. They have a list of supported devices that they update with time as the library matures and the hardware gets powerful enough to support AR [11].

We decided to use ARCore for our project as it was relatively new at that point and

the cost of development of Android apps is low as compared to iOS apps. Besides we already had experience building Android apps before, so it was more feasible for us just to use Android for the project and focus on learning to integrate AR into our app instead of learning to program for whole another platform.

1.6 Firebase

Firebase is a mobile and application development platform which was developed by Firebase, Inc but later was acquired by Google [12].

1.6.1 Services

Firebase offers a range of services to the developers that make app development easier for them. Some of the most popular services include [13]:

• **Google Analytics** Analyzes user attributions and behavior and provides a single dashboard for visualizing and managing it.

• **Authentication** Provides a simple way to authorize users through multiple methods, including email and password and third-party services like Google, Facebook, etc.

• **Cloud Functions** Custom back-end code that gets triggered by some specified events and executes and scales automatically without needing to manage your own servers.

• **Realtime Database** Store and sync data between different users and devices in realtime using NoSQL database, updated data syncs in milliseconds regardless of network connectivity.

• **Cloud Messaging** Send messages and notifications to users, either single devices or groups of devices, across platforms, Android, iOS, and web, for free.

1.6.2 Motivation

We chose Firebase because of the services it provides and because of the great documentation behind it and how deeply it is integrated into the Android development ecosystem. The main services that we used are **Authentication** and **Realtime Database**. We used Realtime Database to update a user's location in real-time and sync it with their contacts. We also used it to introduce real-time chatting into our application.

Another big reason why we chose Firebase is their Free-tier pricing plan [14] and automatic plan upgrade if the app needs more resources. One of the primary motivations behind this project was to make it more economical, use as little resources as possible and spend as little money on it as possible. This will result in providing better services to the users free of cost.

1.7 Summary

Our proposed system will track the location of the users in real-time and store it in Firebase which is a remote real-time database. It will use ARCore, which is an AR library for Android platform, for the visualization of a user's location in the real-world. It will also add support for real-time chatting as well as a chat bot, so the users could have an enjoyable and a rewarding experience.

Chapter 2

Specification and Design

This chapter will first specify what the software is *required to do* and then it will show *how to do it*, that is the design rationale behind it. It will also give a general overview of the working of the system, how the data flows in the system etc.

2.1 User Requirements

The software system should be able to register new users, using a number of ways like email-password and third-party service providers like Google, Facebook, and Twitter, etc. Users should be able to add and remove other users as their **Contacts** using their email addresses. The system should keep track of the location of the user by running a lightweight background service and update their location periodically in the Database Server. The system should update the location of the user's Contacts in real-time and show their location on the Map. If the device supports Augmented Reality, the user should be able to view the location of their Contacts in the real-world using AR. The system should also provide real-time in-app chatting capability so the users could be able to communicate easily with one another. An in-app chat-bot should also be included to help out users and to make their experience more efficient and enjoyable.

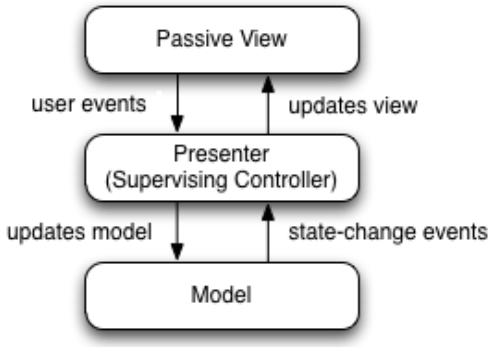

Figure 2.1: Google MVP architecture

2.2 Architecture

Our Android app will use **Firebase Authentication** for the authentication of users. It will then use Android background services and location services to retrieve the latest location update of the user and store it in **Firebase Realtime Database**. It will also use **Google Maps SDK** to show user markers on the Map, where their contacts are located, by retrieving them from the Firebase Database. If the device supports Augmented Reality, it will also fetch the user's contact's location details and show them in the real-world using **ARCore** library.

On the coding level, we will use **model-view-presenter (MVP)** architecture in our android application as it is engineered to facilitate automated unit testing and improve the separation of concerns in the presentation logic [15], as shown in Figure 2.1. In model-view-presenter:

Model defines the data to be displayed in the user interface.

View passive interface to display data and pass-out events to the presenter.

Presenter acts upon model and view by retrieving data and displaying it in view.

2.3 Modules

Separating the core logic of the application into different modules according to their functionality helps in managing, understanding and maintaining the code. Here we will briefly describe different modules being used by our application.

The external modules that are being used extensively are:

• **FirebaseAuth** It provides the authentication functionality for the app. By using this, we could register, log-in and log-out a user from an active session. If the user is not authenticated, he will not be able to access the Firebase Database and as a result, will not be able to utilize this app.

• **FirebaseDatabase** This module provides functionality to read, write, update and delete data from the remote Firebase Database repository. We use this to update the user's information, location, contacts information, messages, etc.

• **FirebaseUI** This module allows us to quickly connect common UI elements to Firebase APIs without having to write so much boilerplate code [16].

• **ButterKnife** It generates the boilerplate code for us to bind fields and methods to Android views.

• **GoogleMap** This module provides all the functionality needed to work with device location, from periodic location look-ups to marker display on the map.

• **ARCore** It is used for implementing Augmented Reality features into the application.

We have also separated our code base into separate modules according to their functionality. Different modules being used in our application are:

• **Activities** All the android activities are stored in this module.

• **Common** The common functionality that is shared between different classes and modules.

• **Fragments** Different fragments that are used in the activities.

• **Models** POJOs (plain old java objects) which are used to read and write information to the Database, as well as Android views, is stored in this module.

• **Services** Services that run in the background.

2.4 Data Flow

To understand how a system works, it is necessary to understand the data flow of the system first. How the data is being passed from one system to another, how it is being transformed, and how it is being processed inside the system. Here, we will show how the data flows in our application.

In Figure 2.2, we show a Level-0 Data Flow Diagram which maps out the general flow of information between the system and the user. The user will be able to manage his/her account, add or remove users from his/her contact list and send his/her location updates from the device to the application where they will be stored. The application will provide the user with contact's information and update him/her periodically with the location updates of his/her contacts.

In Level-1 DFD, Figure 2.3, we break down processes into further sub-processes to show how they interact with each other. There are several systems in our application that can act like standalone systems, like User Management System, Contact Management System, Location Tracking System etc. Here, we show how these systems interact with one another, how the user interacts with the system, and how different modules interact with the remote Firebase database.

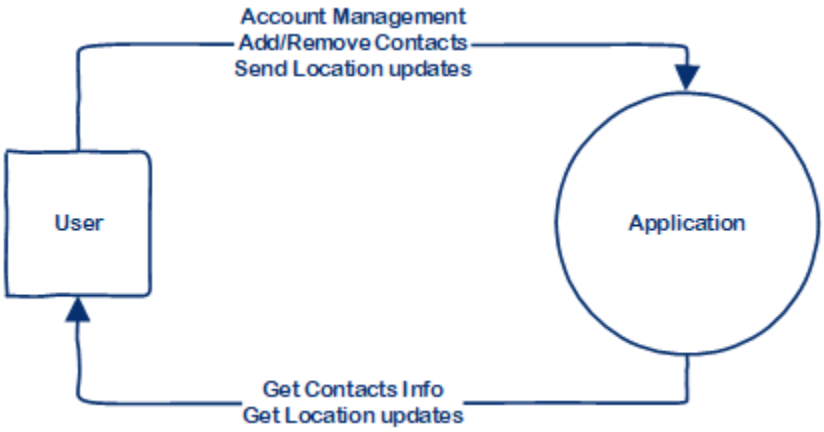

Figure 2.2: DFD Level 0

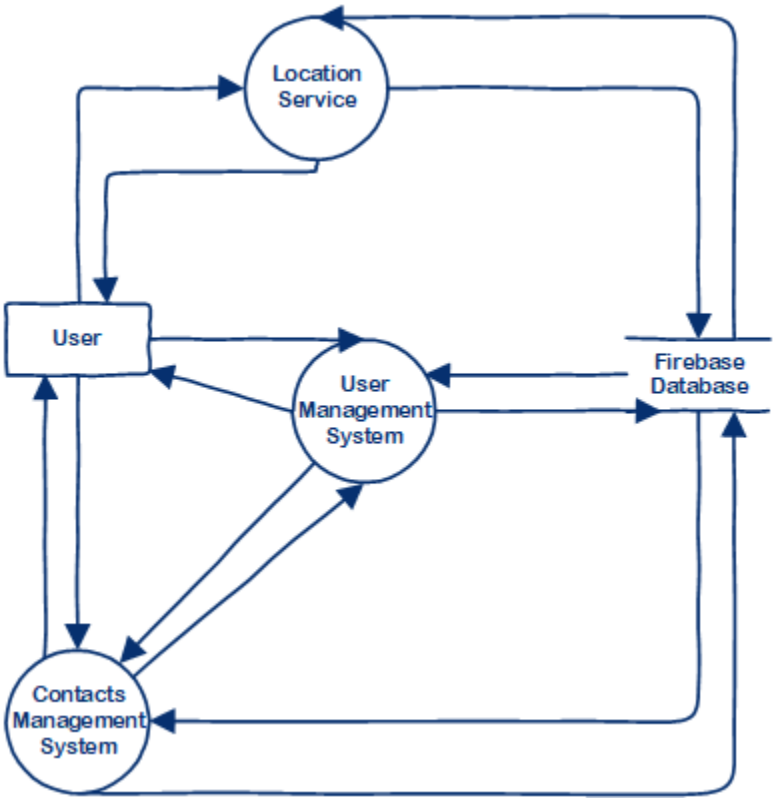

Figure 2.3: DFD Level 1

2.5 Data Schema

We are using Firebase Database, which is a **NoSQL database**. In a NoSQL database, data is modeled in means other than the tabular relations that are used in relational databases [17]. As compared to relational databases, NoSQL databases are more scalable, robust and provide superior performance. The data structures that are used by NoSQL databases (e.g., key-value, wide column, graph, or document) are different from those that are used by default in relational databases, making some operations faster in NoSQL.

Firebase Database is a NoSQL database that uses **JSON**. It has different optimizations and functionality compared to a relational database. It is a recommended practice to avoid nesting of data in Firebase DB because it might result in unnecessary data retrieval, e.g. if we want to get only the name of the user but our database schema is such that we have nested chat messages sent by that user into the user node, then it'll fetch all the messages as well, which is inefficient and a waste of network resources as well. So it is a best practice to avoid the nesting of data and only include as much as required for a particular scenario.

Another good practice is the denormalization of data, that is, data is split into separate paths. This helps in retrieving only the desired information at a particular time. For example, we might store chatting messages data in their separate nodes and location data in its separate node and user information in a separate node. When we need some specific data, we will be able to fetch only that.

Although Entity Relationship Diagram (ERD) is not a good fit for describing the schema of a NoSQL database, still we have provided it here, in Figure 2.4, to give you a general overview of what different fields are being stored and what is the relationship between them.

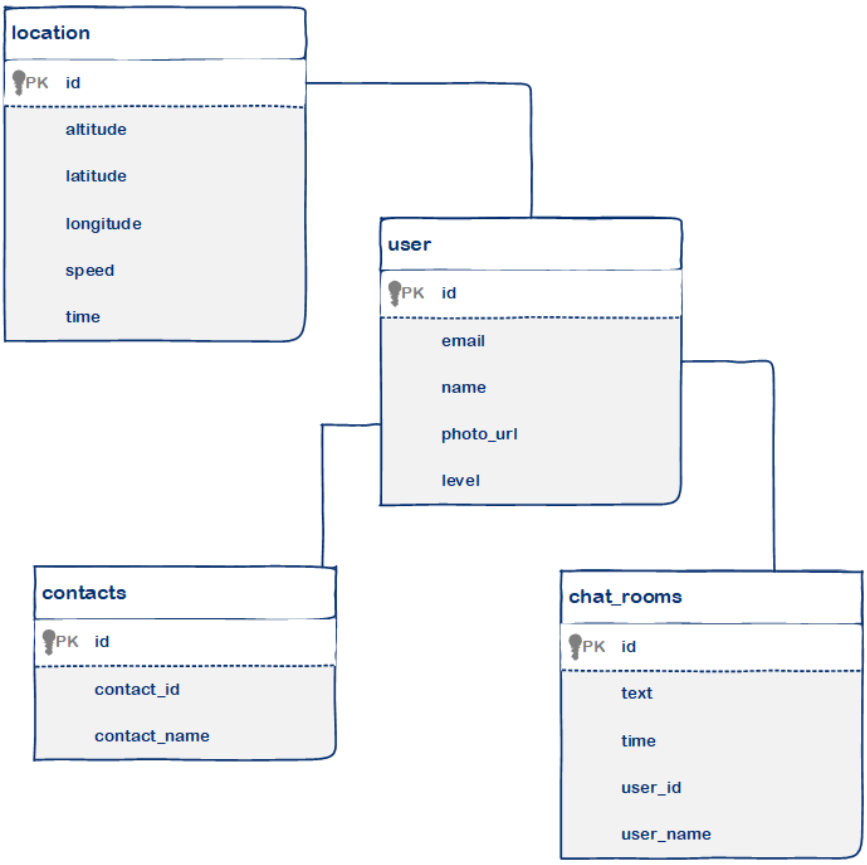

Figure 2.4: Entity Relationship Diagram

Now, we will talk about how the data is being stored in the Firebase database as a JSON tree and how different fields are connected to one another. At the top level, we have four separate nodes for different kinds of data.

```
app
{
    users: {}
    locations: {}
    contacts: {}
    chat_rooms: {}
}
```

We will explain the structure of each and every node in further detail below, and show

how we have followed best practices in defining the data schema for this project.

First of all, `users` node is used to store all the information about the user that is needed in the app.

```
"users" : {
    "tBEJnga0qOePwQhfhEVFP0uC1zB2" : {
      "email" : "aadimator@gmail.com",
      "name" : "Aadam",
      "photoUrl" : "https://link-to-photo/photo.jpg"
    }
}
```

At first, we have the user id (UID), generated when he registers for the first time and stored here for future reference, that is used to identify the user throughout the application. Then we store basic information like his email id, his name and profile picture that we use in the application on various points.

`locations`, as the name suggests, is used to store the location of a particular user.

```
"locations" : {
    "tBEJnga0qOePwQhfhEVFP0uC1zB2" : {
      "altitude" : 0,
      "latitude" : 33.9859268,
      "longitude" : 72.9703671,
      "speed" : 0,
      "time" : 1514393767560
    }
}
```

The same `id` is used to store the location as the one used to store the user information in the `users` node. This helps in easy retrieval of relevant information.

`contacts` node is used to store the information about the contacts of a particular user.

```
"contacts" : {
    "tBEJnga0qOePwQhfhEVFP0uC1zB2" : {
        "4eTIfKSASHfSNjqv5rr1Ay3uKuS2" : "Panda",
              "xwykXzzlMjXJvNVjUbk65ZZxvjt2" : "Po"
    }
}
```

We store the `uid` and `name` of the contact. We also store the name as it is mostly used in the application. The name of the contact is displayed in the list so instead of querying for full user information, we store his name here which reduces the number of queries to the database.

At the end, we have `chat_rooms`. The `id` for the `chat_rooms` node is generated by concatenating the two unique ids of the users that are participating in the current chat room. This ensures that it will always be unique and easily accessible.

```
"chat_rooms" : {
    "tBEJnga0qOePwQhfhEVFP0uC1zB2_oxCpx7sffgRfTc7SrLBTcW0LGW12" : {
        "-LI0dHZ6ff_GBJg3CNqO" : {
          "text" : "we are in the process of moving and I will",
          "time" : 1532257184164,
          "userId" : "oxCpx7sffgRfTc7SrLBTcW0LGW12",
          "userName" : "Po"
        },
        "-LI0eJcQzi6yx8Qn45Hl" : {
          "text" : "thanks for understanding",
          "time" : 1532257454754,
          "userId" : "tBEJnga0qOePwQhfhEVFP0uC1zB2",
```

```
            "userName" : "Aadam"
        }
        }
}
```

In the `userId` and `userName`, we store the details of the user that is sending that particular message.

By looking at this Data Schema, you can see how we followed best practices, that is, avoided nesting of data and tried to denormalized the data. Each separate path or node serves a particular purpose and only contains information for a specific scenario.

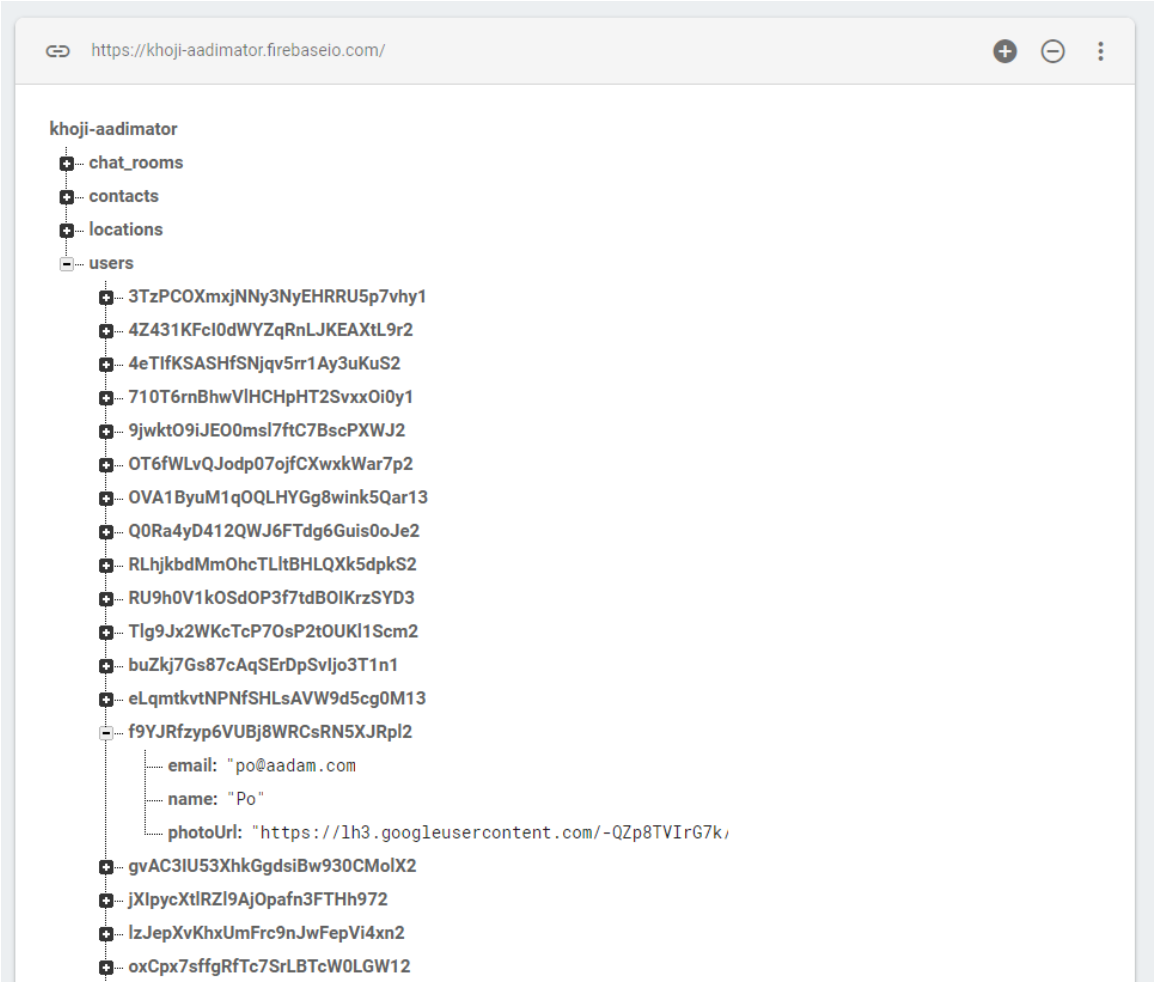

Figure 2.5: Firebase Database of our system

2.6 Design and Implementation Constraints

One of the major constraints is the availability of the internet. Although we have enabled offline persistence in our application, which will store the data locally in case of network failure and upload it immediately when the network is back on, the real-time nature of the app will be affected by the unavailability of a network connection.

Another thing to keep in mind is the Firebase Database Limits [18]. Although we can scale up to a bigger tier, it will cost more and given that one of the primary objectives of this project was to be more economical, we have to design our system accordingly.

We also have to keep in mind those users who do not have a compatible device for Augmented Reality features. We want to design our app in such a way that it is still accessible and usable for those users as well. If we only design for users with Augmented Reality supported devices, it would significantly reduce our user-base, thus affecting the usability of the app because most of the users will not be able to add their friends or family as their contacts.

We also have to keep in mind the background services constraint that has been implemented in Android v8.0 and above. Now, the apps can not use background services as frequently as they want but some time and frequency constraints are applied to them. Once the app is in the background, it can only listen for background updates a few times in an hour. This is used to improve the battery performance of the smartphone.

Chapter 3

Proposed System

The implementation aspect of this project requires the development of an infrastructure which can be used to fetch and receive location updates and chat messages in real-time and, where supported, use Augmented Reality to show the markers superimposed on the real world. In this chapter, we will talk about the implementation details, the unforeseen problems that we faced during the development phase and how we chose to overcome them.

3.1 Major Areas

We will divide our proposed system into major areas according to their functionality and responsibility. This will help us in focusing on a specific problem at a time and explaining it well so that you could understand it better. Following are some of the major functions of our application:

1. User Account Management
2. Location Tracking
3. Contacts Management
4. Contact's Location Markers

5. Real-time Chat

6. Augmented Reality

7. Chat Bot Integration

Now we will discuss each area separately, and show you how we implemented it in our application.

3.2 User Account Management

We have discussed earlier that we will use **Firebase Authentication** services for user authentication and account registration. Here, we will show the whole process of how we implemented it, and how we are managing the user's account in our database.

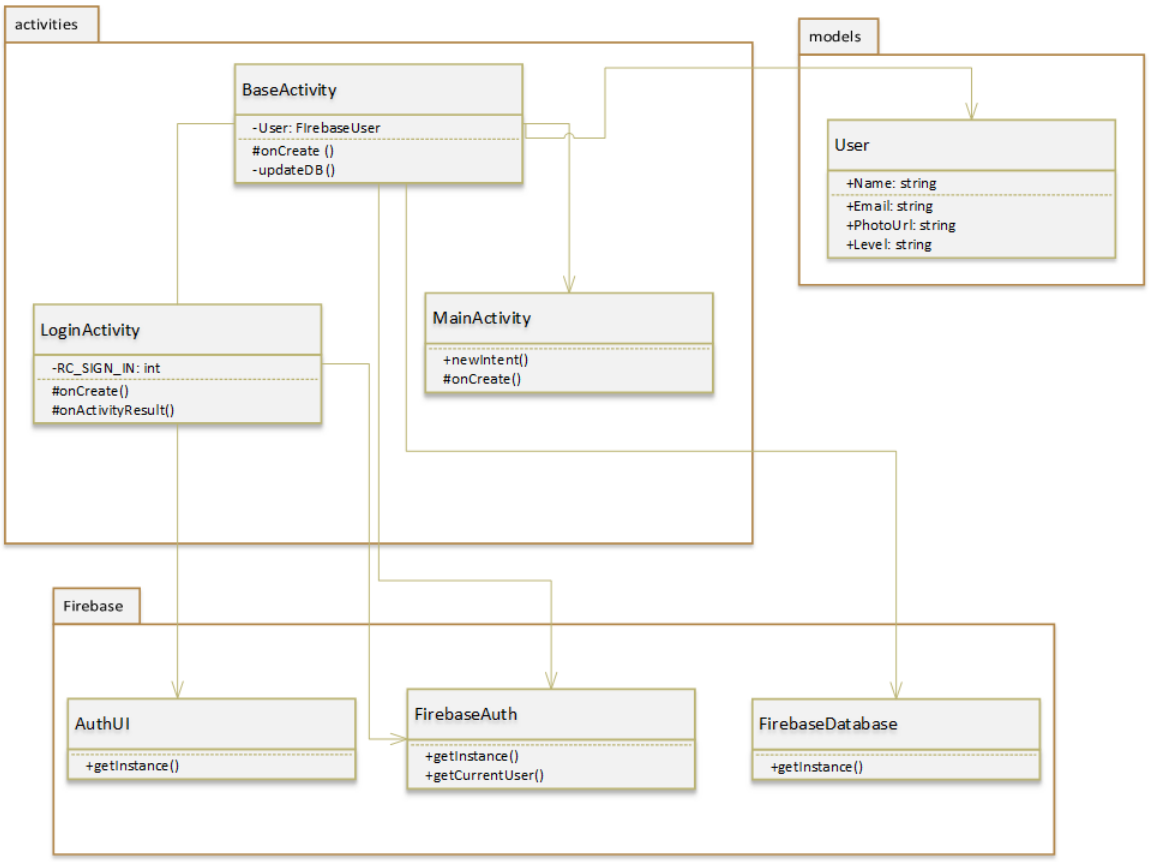

Figure 3.1: UML for Account Management

In Figure 3.1, we show the UML diagram of the account management system, without going into much detail about other functionality in these classes, just the ones related to the account management. The `User` class in the `models` package has private member variables, but with `getters` and `setters` for each and every variable, so instead of cluttering the diagram with redundant functions, we have shown them as public member variables here.

As we are developing an Android application, it always starts with a launcher activity. In our case, the launcher activity is the `BaseActivity`. When the application starts, it launches the `BaseActivity` and before doing anything else, we see if the user is already logged in. If he is already logged in, we proceed to the `MainActivity` where the user can freely interact with the application and perform his desired tasks.

However, if the user is not logged in, then we open up the `LoginActivity`. Here, we use `FirebaseAuthUI` which is a library developed on top of **Firebase**. It provides some general functionality regarding account registration and log-in, so we do not have to write much boilerplate code. In `LoginActivity`, we call the methods of `AuthUI` to *Register* or *Log in* a user. This opens up a separate screen where the user can register himself for a new account using various options; email-password, Google, Facebook, Twitter, as shown in Figure 3.2.

Upon registering, the Firebase Authentication saves the credentials of the user in the cloud and returns the user details to our application. We relaunch the `BaseActivity` class and save the details of the user in our **Firebase Database** as well, using `updateDB()` function, because we will be using that throughout the application.

The user details that were returned by the Firebase Authentication also contain a unique identifier for that user, `uid`. We use that `uid` to create a new node in the Firebase Database under `users` node. Now, we have a unique way of referring to that user whenever we want.

After storing the credentials of the user in the Firebase Database, we forward him to the `MainActivity` where he can carry out his usual tasks.

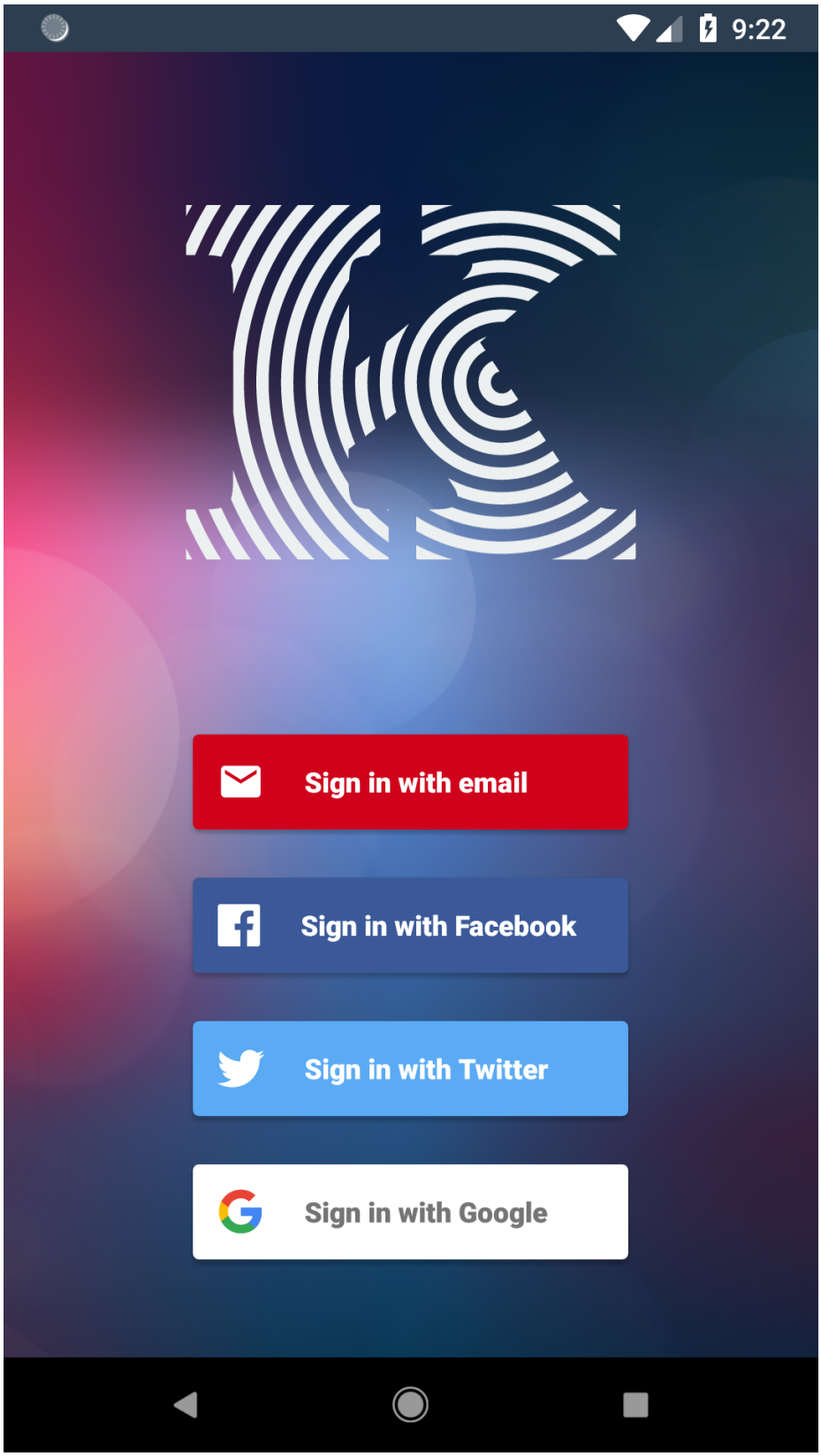

Figure 3.2: UI for the sign-in screen

3.3 Location Tracking

After the user has logged in, the first thing that needs to be done is to start a Location Service, which runs in the background and periodically send updates to his location.

When we started working on this problem, before the release of Android 8.0 (API level 26), the way to solve this problem was entirely different. Before Android 8.0, we could query for Location updates as frequently as we wanted to. The API for listening to Location updates was also quite simple as compared to what it is now.

However, midway into developing this application, Android 8.0 was released which changed the way to retrieve location quite radically. They limited how frequently background apps can retrieve the user's current location [19]. When the app was in the foreground, we could ask for location updates as frequently as wanted, but when it was running in the background, we could not. We had to query for location update to the Android system, and it would only return the update only a few times each hour, no matter how frequently we requested it.

Because of this limitation, we had to redesign our whole app, keeping in mind the limitations imposed by the new standards as well as the new API that was recommended for this. Now, when the app is in foreground, that is the user is actively interacting with it, we use a different location retrieval logic which asks for location updates every few seconds. However, when the app goes to the background, we start a background Job service, which updates the location every few times in an hour. We do not have much of a control over how frequent the updates will be in the background as the Android OS takes it in his own hands to handle background services so as to increase the battery performance of the system.

As shown in Figure 3.3, in `MainActivity` we start listening for location updates. This will listen to the location updates in the foreground, and as soon as we receive an update, it will store that location in the Firebase Database under the `locations` node. The `uid` will store each location of the currently logged in user. `UserLocation` class is

used to model the location and storage and retrieval of the location for the database. Again, the public member fields shown in the figure are not public, they are **private** but with getter and setter methods.

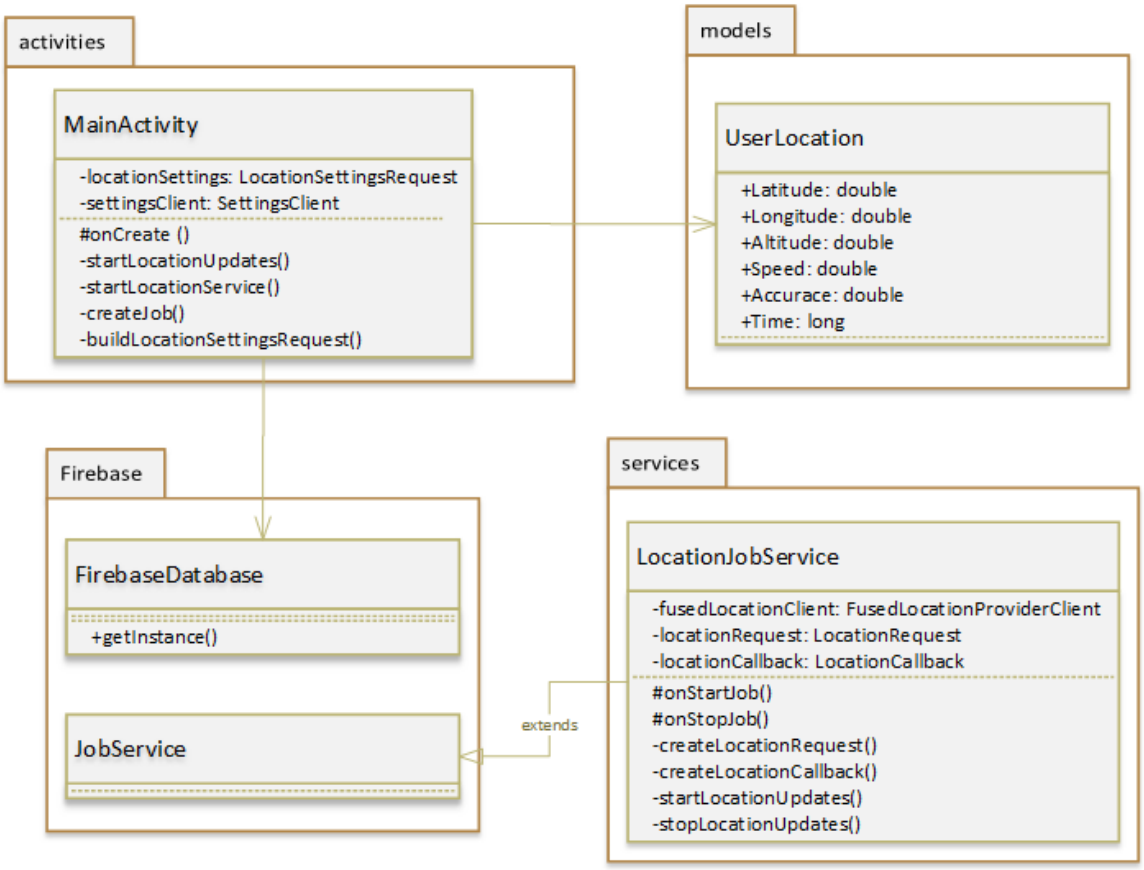

Figure 3.3: UML diagram for Location Tracking

We start location service in the `MainActivity` as well, which start the background service for listening to the location updates. It creates a `LocationJobService`, which `extends JobService` which runs in the background and whenever it receives a location update, it updates it in the database.

There is also boilerplate code present for requesting location tracking permissions, following best practices for Android 6.0 and higher [20]. In `MainActivity`, we also keep track of whether the user has enabled the location service on his device or is it turned off. If it is turned off, we prompt him to enable the location service, as shown in Figure 3.4, so he could use the app effectively.

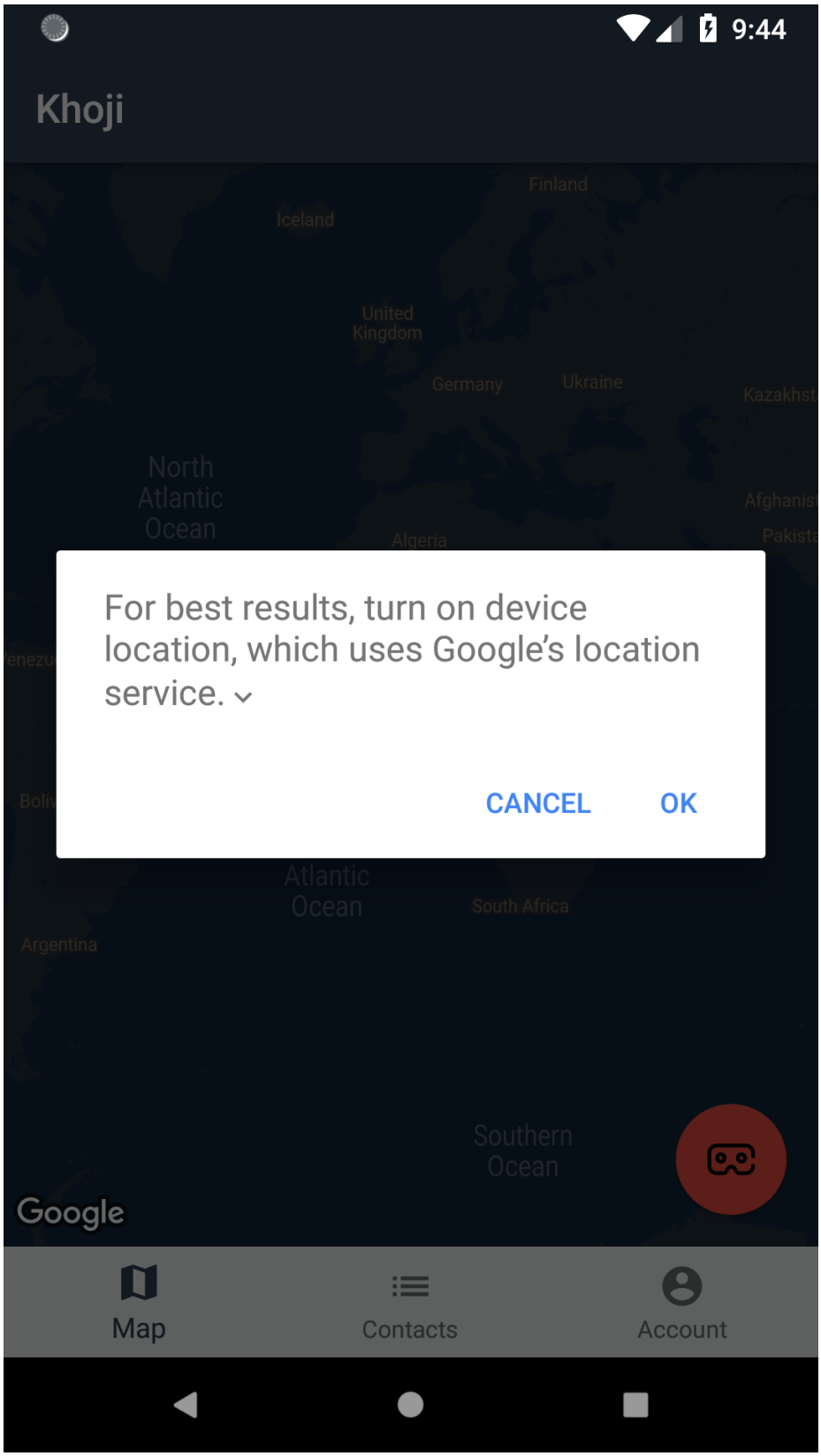

Figure 3.4: UI for enabling the location permission, if disabled

3.4 Contacts Management

Another best practice in Android development is to use `Fragments`. Fragments represent a behavior or a portion of the user interface in an activity. They are always hosted inside an Activity, and when that activity is closed or gets destroyed, they are also destroyed along with it. Fragments increase the modularity and re-usability of the application [21].

We have made use of fragments in our application extensively. `MainActivity` acts as a placeholder for different fragments. When the user presses a button to change screen, the **fragment container** in the `MainActivity` gets updated so that the old fragment is replaced with the new fragment in the container. This helps in keeping the code modular and making the user interface responsive.

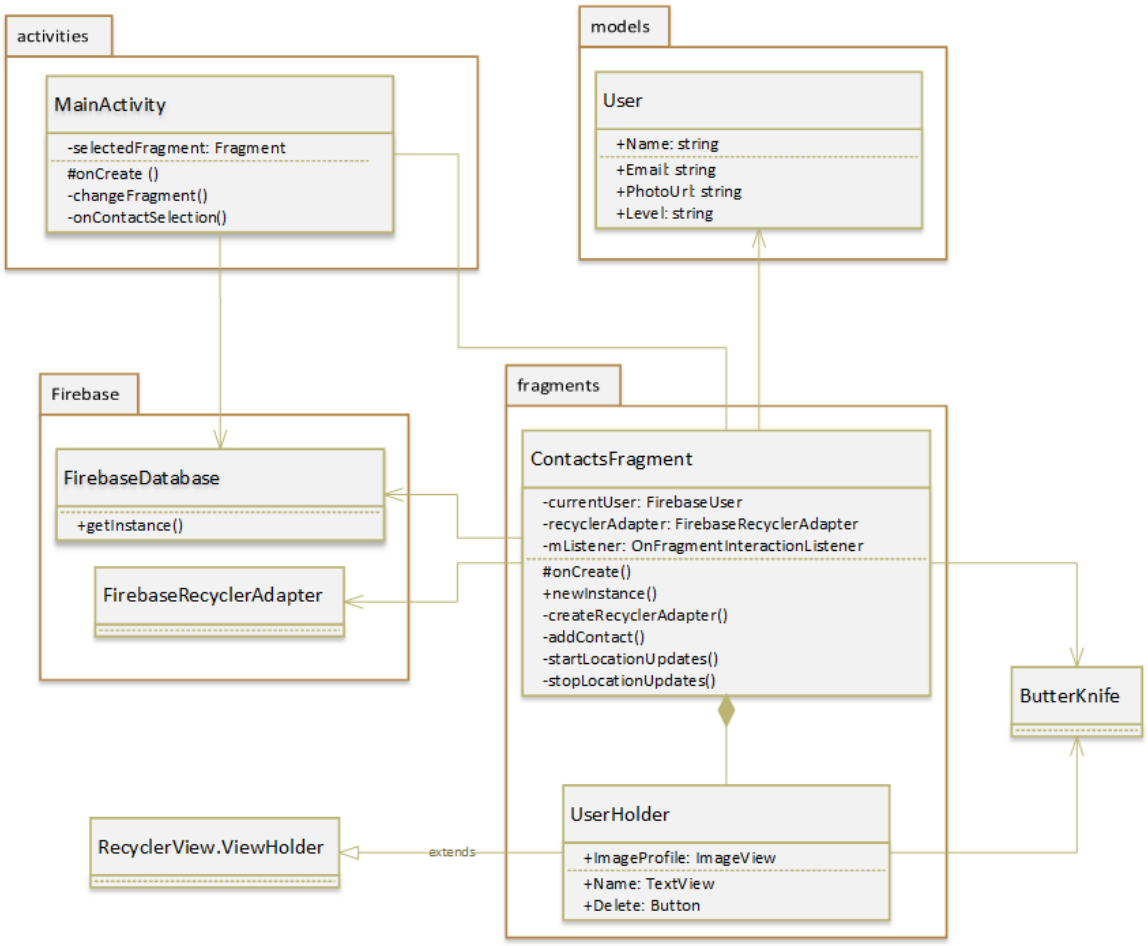

Figure 3.5: UML diagram for Contacts Management

As shown in Figure 3.5, when the user presses the button to change the fragment to `ContactsFragment`, we update the fragment container in the `MainActivity` to load `ContactsFragment`. In ContactsFragment, the user can see the list of all of his contacts, remove the ones he does not need anymore and add additional contacts to his contact list.

To add a new contact to his contact list, a user must type the email address of the user that he wants to add. If the email address is present in the database, then that contact will be added to his contact list, and the contact list of both the users will be updated to show the newly added contacts.

We use `FirebaseRecyclerAdapter` to populate the list of all the contacts a user have. This class automatically fetches the list of contacts from the Firebase Database, maps them into `User` class and populates the `RecyclerView` with the layout that we specify in the options. This reduces much boilerplate code.

Another library that we use heavily throughout the project is `ButterKnife`. This library binds the views from the layout files to the java source code files so we could easily access them in the code. By using this library, we do not have to type out a whole lot of redundant code as it has been taken care of by this library.

`UserHolder` class is an inner class of ContactsFragment which only holds the views that need to be populated in the list once the data is retrieved from the database. This class extends `RecyclerView.ViewHolder` class and we use `ButterKnife` to bind the views from the layout file to the field variables.

The user interface for the contacts management is shown in Figure 3.6. To add a new user to the contact list, one must have to add the email address of the user that he wants to add in the field marked by **1** and then press the add contact icon, marked by **2** in the image. If the user is available in the system, he will be added to the contacts list of the current user and the list shown below will be updated immediately to reflect the new addition.

By clicking on the user name or user image in the list, as shown by the marking **3** in the

figure, the current user will be taken to the location marker of that contact in the Google Maps view. If he clicks on the chat icon, marked by **4** in the figure, a new chat window will appear so that the user can start chatting with that person. By clicking on the trash icon, marked by **5** in the figure, the user can remove that person from his contacts list.

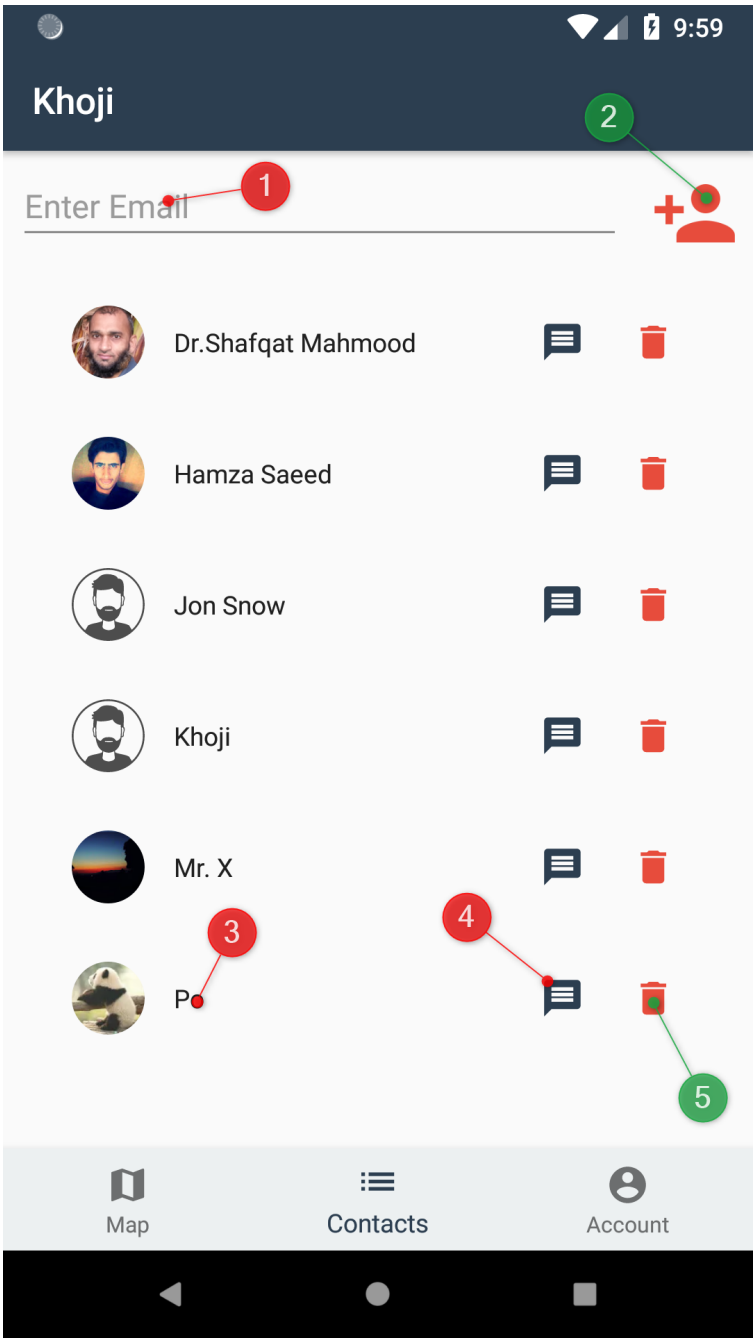

Figure 3.6: UI for Contact Management screen

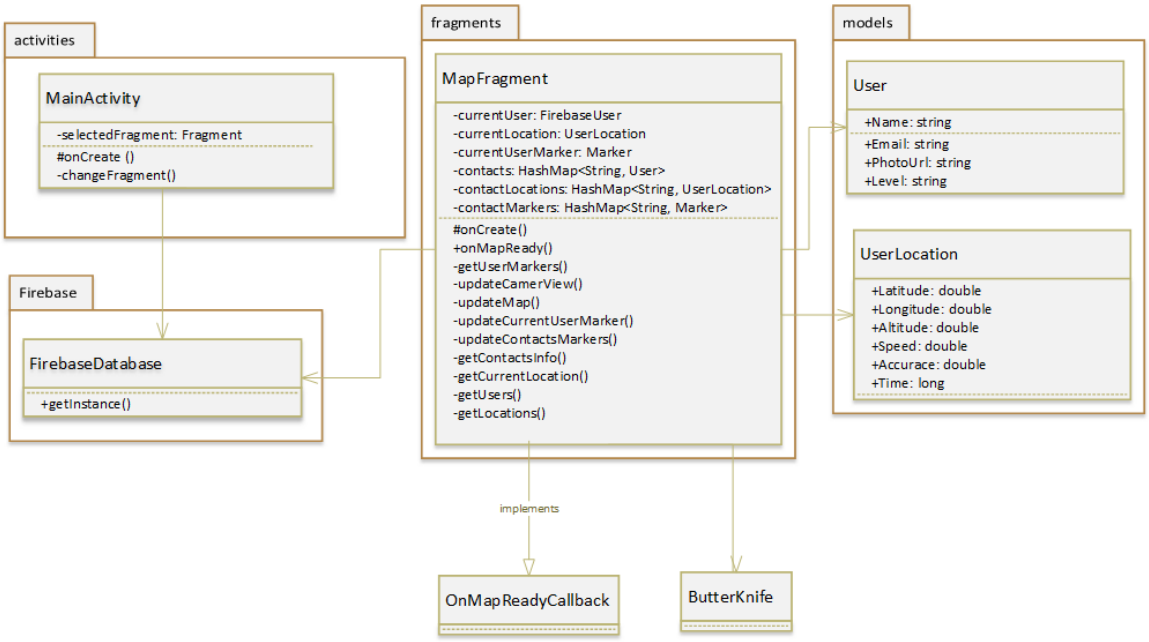

Figure 3.7: UML diagram for Contact's Location Markers

3.5 Contact's Location Markers

Now that we have a way to store the location of every user using our app and we have devised a way for users to add one another as each other's contacts, it is time to fetch the location details of each of a user's contact and show them as markers on a map. We are using a map here because we also need to keep in mind the users that don't have devices capable of augmented reality. If we only used augmented reality to visualize the location of users, then it would not have been usable by those that do not have access to an AR-enabled device, and as a result, it would not have been usable by those that do have an AR supported device because they would not be able to add their friends and family as their contacts.

As shown in Figure 3.7, we load the `MapFragment` into the fragment container located in the `MainActivity`. In MapFragment, we use **Google Maps SDK** to load Google Maps in our view. For this reason, the `MapFramgent` class `implements` `OnMapReadyCallback`, which gets called, as the name suggests, when the map is ready.

After Augmented Reality implementation, this was the most cumbersome and difficult implementation in the whole project, as we had to keep track of many things and infuse multiple technologies together.

`MapFragment` class keeps track of the current location of the logged in user and updates his marker on the map, which is shown in green on the map to distinguish him from his contact's markers. `MapFragment` class then queries the Firebase Database to retrieve the contacts of that user. This query will return only the `uid` and the name of their contacts, as we have designed the database schema in Section 2.5 in such a way.

Now that we have the `uid`s of the user's contacts, we store them in a list and query the database for `UserLocation` updates on each and every user in the contact list. We store them in a `HashMap` where the key is the `uid` of the user and the value is the `UserLocation` class populated with the details of that particular user. We also query for the user information for each user in the contact list and store them in another `HashMap` where they key is `uid` of the user and value is the `User` class populated with the details of that user.

Now that we have all the detail that we need to populate the Map view with user markers, we create a `HashMap` to store the Markers as well, where again, the key is `uid` of the user and the value is the `Marker` for that user. We need to keep track of each marker because when the location of that user changes, we need to update the marker as well to reflect that change. We then populate the Map with all the markers, both the current user's which is shown in green color as well as of contact's markers which are shown in red.

After that, we set-up a **Firebase Database Listener** that listen to any changes, if they happen in the database. Whenever the location of the user changes, it will trigger an **event** and we will handle it by fetching the latest location of that user from the database and updating it in our `HashMap`s as well as on Google Map view.

As shown in Figure 3.8, when a user clicks on a marker, marked by **1** in the figure, a small window will show up at the bottom of the screen which will list further information

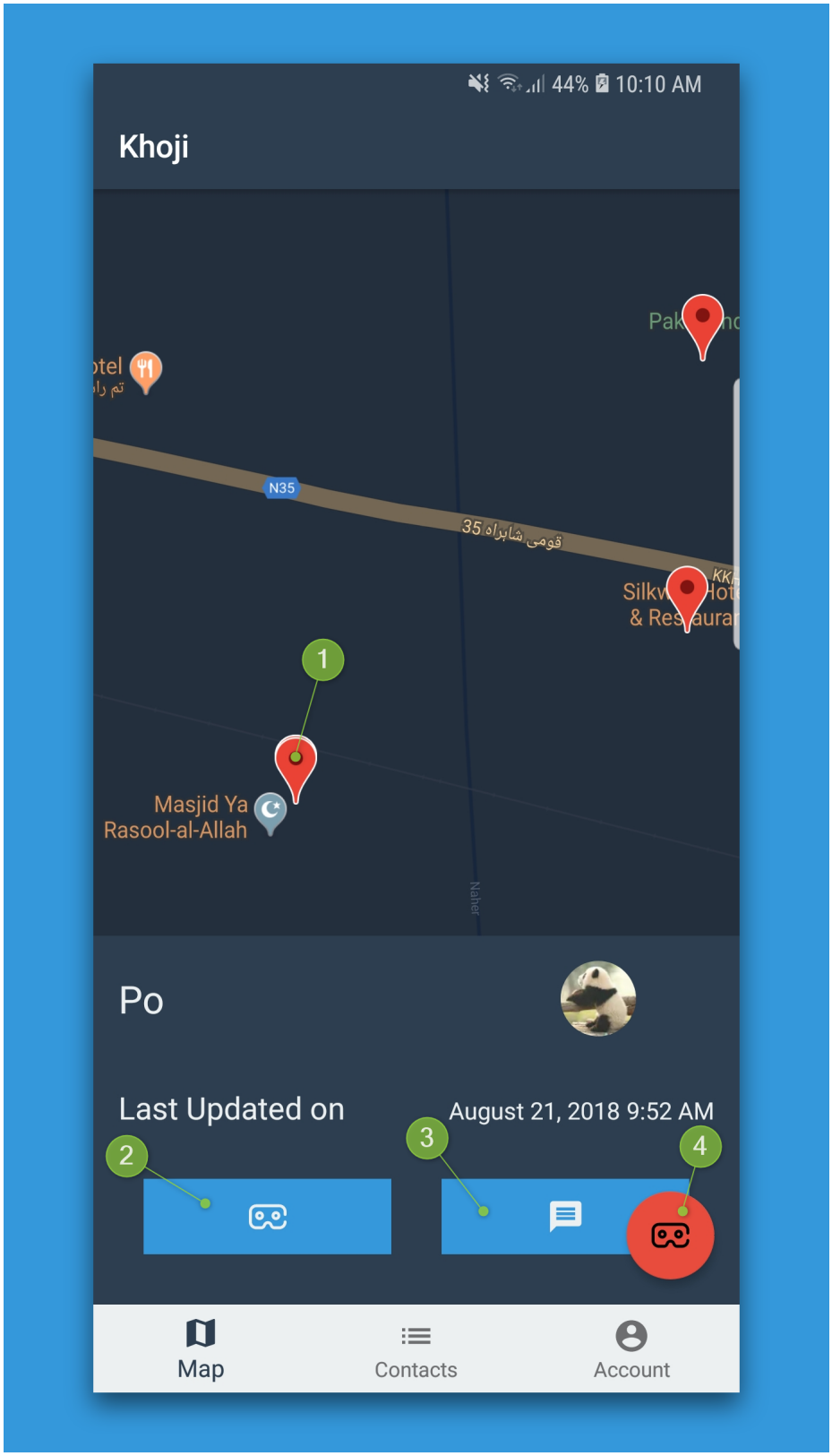

Figure 3.8: UI for Location Markers on the map

about that marker, e.g. name, last update time etc. along with other operations that could be performed for that marker. If the AR is supported on that device, by clicking on the AR button, marked by **2** in the figure, the AR activity will be launched which will only show the AR marker of that particular user in the real-world. If the user presses chat icon, marked by **3** in the figure, then a chat window will appear so he could chat with that user. Finally, by pressing the icon marked by **4** in the figure, the user can launch AR activity which will show all the markers of the users that are in the contact list of that particular user.

3.6 Augmented Reality

In this section, we will talk about the problems that we faced during the implementation of this feature and the different approaches we used to overcome those problems.

3.6.1 Problems

We faced a lot of problems with the implementation of this feature in our application. The `ARCore` library was in early beta release when we started working on this problem. So there was little to no documentation available. We had to look at the source code to figure out how the code is supposed to work.

Another main problem was the stability of the AR objects. During beta release of the library, there were a lot of bugs and the stability was quite weak, but with every update, it improved a little bit. However, along with the improvement of the library, the API of the library also changed, and the way to do things was also affected by this. This resulted in code rewrite every time there was an update as the previous code would not work with the new API.

Another main problem is that the ARCore library was not designed with world-scale AR in mind. Now that it has matured enough, it still does not work very well with world-scale AR where we place AR objects in the real-world using longitude and latitude. AR-

Core works best by scanning a surface and create a cloud of points where a user can place their AR objects, and it anchors that object to that point in the cloud. Because of the cloud of points, we can see the object exactly where we placed. ARCore is perfect for this type of AR, but in our application, we are using world-scale AR which means there are no points of cloud present and we place the object using longitude and latitude. This has serious stability issues, as the marker would keep floating in the AR scene or it could be shown in West while it needed to be in the East.

3.6.2 Different Approaches

During the lifetime of this project, we have tried several approaches to solve the problem of implementing Augmented Reality into our application. It has proven to be a lot more difficult task than we imagined it to be. As with any new technology, the lack of proper documentation and the scarcity of tutorials make it much harder to implement it for our application. Here we will briefly describe some of the things that we tried to make it work for us.

3.6.3 Unity and Mapbox

After completing the functionalities as mentioned above of the application, we looked at several online resources to see what would be the best way for us to implement Augmented Reality into our application. After searching long and hard, we decided to use **Unity** and **Mapbox** for this, as they were already working on the problem of world-scale AR and had a library for location-based AR [22].

We were successful in creating a basic prototype that would fetch results from the Firebase Database and show them in AR using Unity Engine. However, there were some problems with device calibration which resulted in the poor mapping of AR markers to the real-world location. The inner compass of the mobile phone was poorly calibrated to work with it, meaning if the user was in East, the AR environment would show him in West.

Another main reason why we decided to abandon that approach is that Unity was not very optimized for Android. It can build a solution for multiple platforms, including Android, iOS, Windows, Oculus, etc. but for Android, we needed to extract every single bit of performance gain that we could get. Before that, we had already implemented the rest of the system architecture in Android, so now, linking a poorly working, resource massive Unity project with our Android app did not seem very efficient. So, we put that aside and tried working with ARCore, using Java.

3.6.4 ARCore with OpenGL

When we started this project, the **ARCore** library was still in early beta. There was little to no documentation available, and we had to look at source code and one or two available examples on the internet to understand how the library works. This was a very tiresome and unrewarding experience for us. Moreover, at that time, we had to work with **OpenGL** to render objects onto the AR scene, which itself is quite a tedious task.

After a lot of hard work, we were able to visualize the markers in AR using simple views that just showed the name of the contact. The same problem persisted here. The markers were not stable. They would keep floating in the AR scene view, always trying to adjust their position.

Although we were very frustrated after this, we still did not give up and started our hunt for some other way this problem could be solved.

3.6.5 ARCore Location

At this time, a new open-source library was released on Github.com, ARCore Location [23], which was trying to solve the same problem that we were having, i.e., placing AR objects within the AR scene with real-world GPS coordinates. We decided to give it a try.

As compared to previous solutions, this was easy for us to implement, because they provided a working example which we can use to understand how this library works. This

library also removed a lot of the boilerplate code that we wrote for our previous version. Under the hood, it was doing the same thing that we were doing before, just that it provided a nice API so that we do not have to write all that boilerplate code ourselves.

We integrated this library into our application. At that time, we were still using OpenGL to render AR objects into the AR scene, which is tedious low-level work with graphics and quite exhausting to modify. This made it harder to experiment with different methodologies and techniques as their was always the danger of breaking it all together.

We noticed a little bit of improvement in the stability of the markers but their positioning was still wrong, and after a few seconds, the markers would start floating again. We decided to keep this solution for now, as it refactored and cleaned our code somewhat.

3.6.6 Current Approach

Now, we are using the same library, `ARCoreLocation` [23], but we have updated it to the latest release. We started with **ARCore** when it was in beta, and now, we are using **ARCore v1.3**. This version is a lot more stable than previous versions. The markers that are shown in the AR scene do not float now. There is a little bit issue with the positioning of the markers sometimes, but mostly it works fine now.

Another significant change in the new version of ARCore was the inclusion of `SceneForm`. By using this, we do not have to work with 3D graphics and **OpenGL** at such a low level. `SceneForm` provides us with a high-level API, realistic physically based renderer for importing, viewing, and building 3D assets and easy integration into ARCore [24].

As you can see in Figure 3.9, the `ArActivity` is started when the user presses the button from the `MapsFragment`. If the device of the user supports Augmented Reality, then the button will be enabled; otherwise, it will be disabled. When the `ArActivity` launches, it first checks if the **ARCore** library is installed on the user's device. If it is not installed, it prompts the user to install the library. After that, it checks to see if the user has given the necessary permissions for the app to use AR features and prompts the user to do

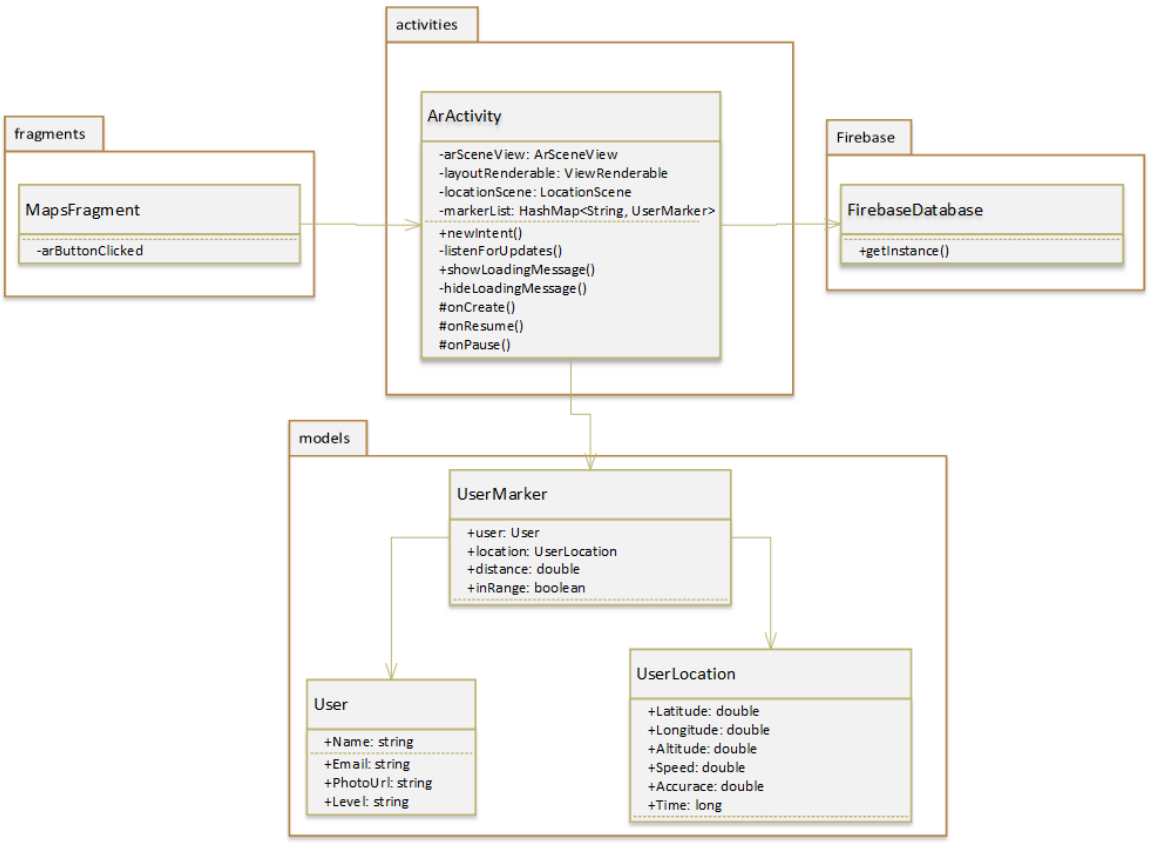

Figure 3.9: UML diagram for AR activity

so if he has not given the permission yet.

Once everything is set-up, the application will then place the markers, loading from the `HashMap<String, UserMaker>` which was passed to this activity from the `MapsFragment` class. This will display the markers in the AR scene at their respective GPS coordinates.

The application will also start listening to the changes in the database. As soon as the location of the contact is updated in the database, it will be fetched from the database, and the respective marker will be updated to reflect the new position of the user.

In Figure 3.10, we can see how the marker will be superimposed on the real-world. Here, it is showing the name, profile picture and the distance of that person from the user at that moment. If the user clicks on the marker, a small window will appear at the bottom, showing further details, like the last time of location update and the option to launch the chat activity so you could start chatting with that person instantly.

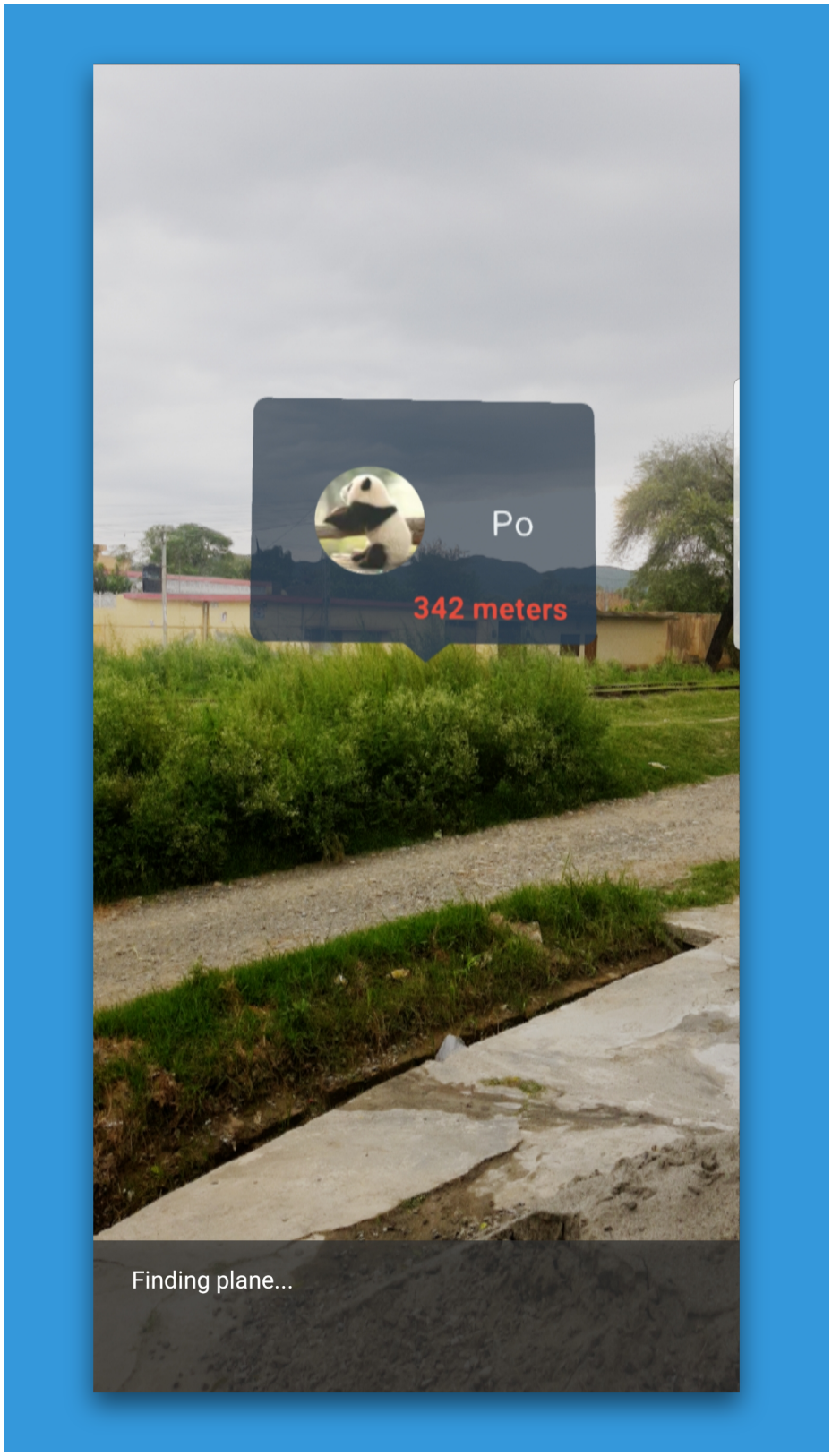

Figure 3.10: User marker shown in AR

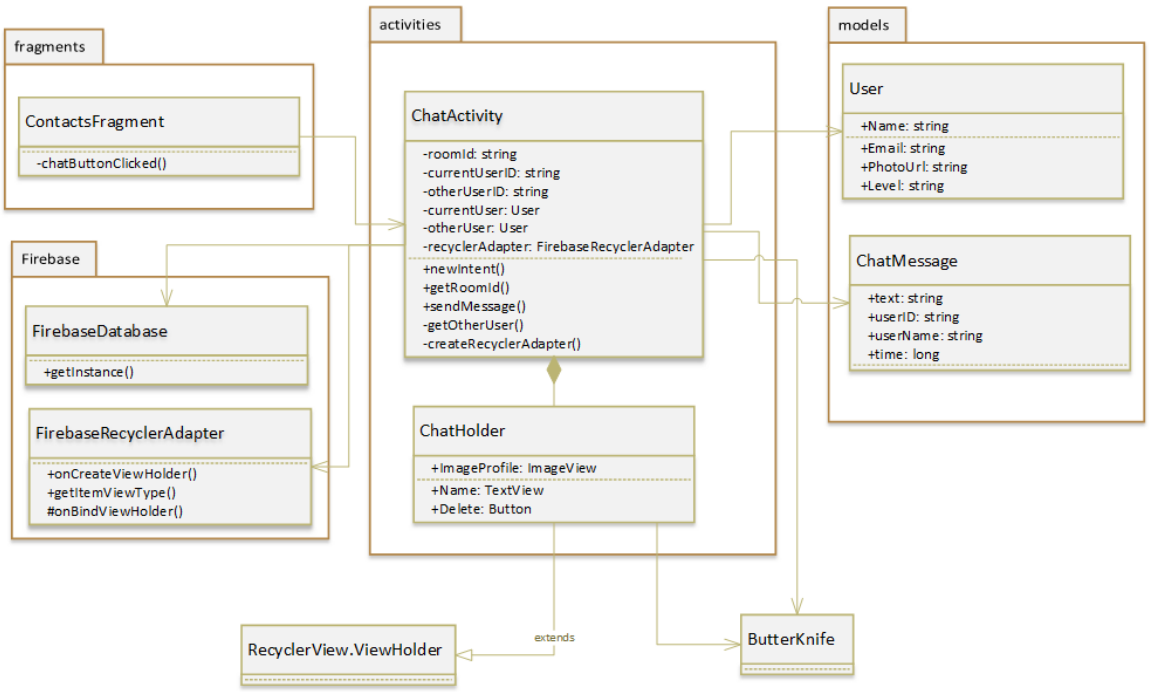

Figure 3.11: UML diagram for Real-time Chat

3.7 Real-time Chat

We also used the Firebase Database for implementing real-time chatting in our application. When the user presses the chat icon from the `ContactsFragment` class, it opens up a new activity where he is provided with chatting functionality with his Contact so he could quickly get in touch with him.

We can see, from Figure 3.11, that the `ChatActivity` class uses two classes from the model package, one `User` class and the other `ChatMessage` class. `User` class is used to retrieve the info of the contact that the currently logged in user wants to talk to. Also, the `ChatMessage` class is used to read and write chat messages into the Firebase database and show them in the chat message `RecyclerView` as well.

We have followed the best practice in designing the database schema for chat rooms so that there is no unnecessary cluttering of data and the data is perfectly denormalized. We generate the chat room id using this simple algorithm:

```
public static String getRoomId(String uid1, String uid2) {
  return ((uid1.compareTo(uid2) > 0) ?
                    uid1 + '_' + uid2 :
                  uid2 + '_' + uid1);
}
```

This algorithm will ensure that there will always be a unique room_id and we can easily query it whenever we want. We are concatenating the `uids` of both the users that are participating in that chat room. We use lexicographical order to sort the `uids` and then concatenate them.

When a user sends a message, we store the details of the message; *message text, message time, sender id, sender name*; in `ChatMessage` class and then write that information in the Firebase Database.

To display the chat messages, we use `FirebaseRecyclerAdapter` which fetches the latest messages from the database and populates the list according to our specified layout. We use a separate layout for different type of messages, that is, for messages sent by the current user and the received messages. We also listen to the database changes, as soon as a new message arrives, we populate the message list with that new message and scroll the list to the latest message position.

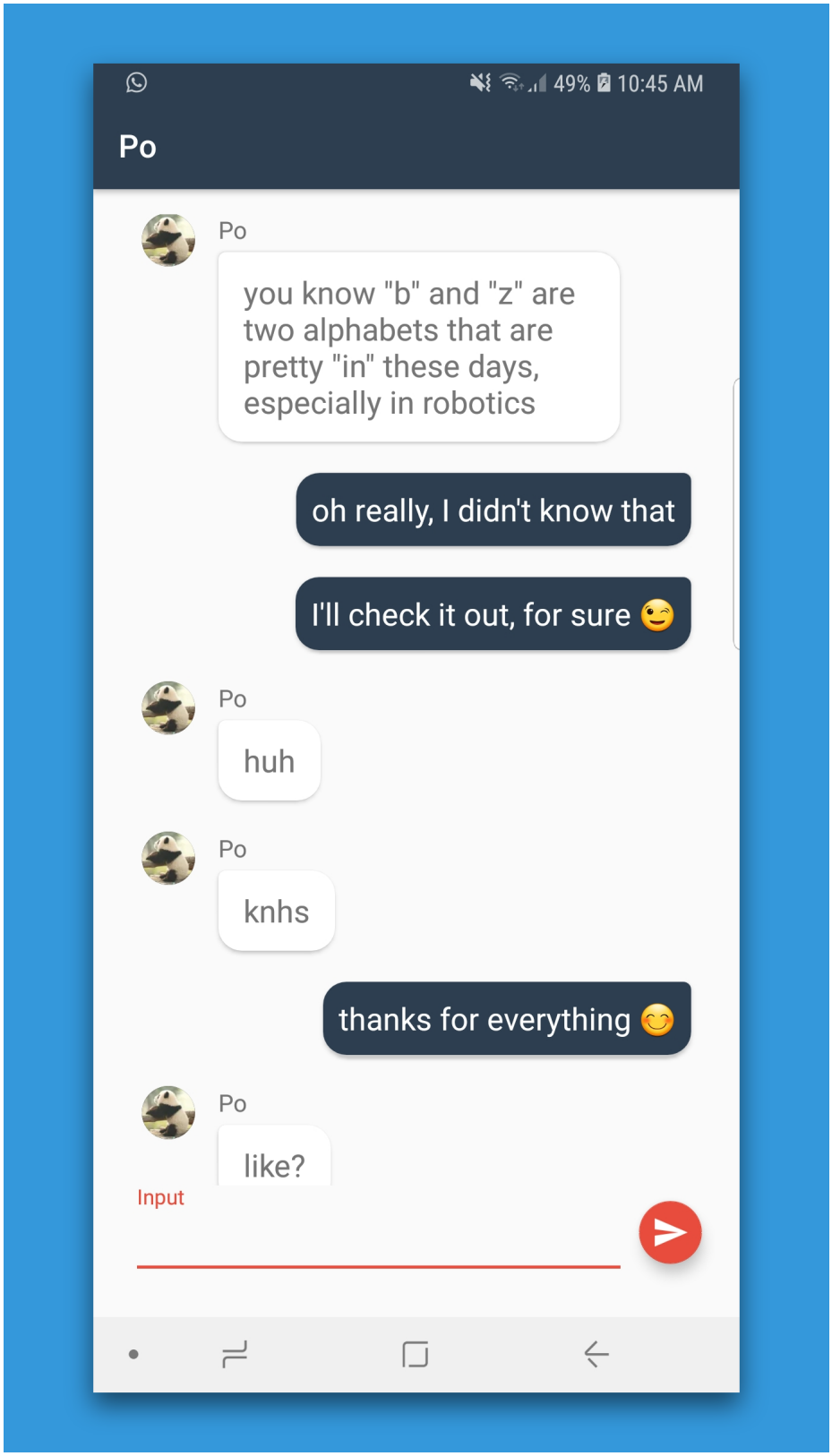

Figure 3.12: UI for real-time chatting in Khoji

3.8 Chat Bot Integration

We have also included a chatbot in our application to aid the user in effectively using this app and to make his experience more enjoyable. The chatbot will be automatically added to the contacts of all the users as soon as they sign-up. The users could ask a simple question to the bot, and it will respond accordingly, doing his best to satisfy the needs of the users. The bot should be able to understand almost all of the queries of the user as it will use NLP (Natural Language Processing) to comprehend the user's queries.

Several options provided the services we required for our application, and among them, we chose **DialogFlow** [25] for this application. Formerly, this was known as **Api.ai**, but after its acquisition by Google, they changed its name to DialogFlow. We chose DialogFlow because of their fantastic documentation, easy customization and quick integration with several platforms. The main reason we went with DialogFlow is their pricing model [26]. They provide free services in their Standard Edition plan, which has more than enough criteria for our app usage.

As shown in Figure 3.13, we used Fuel to use the DialogFlow services. `Fuel` is a lightweight networking library that is used to send and retrieve requests across a network. We use `FuelManager` to call the REST services provided by DialogFlow to interact with our bot. The bot is trained on the DialogFlow platform using queries and answers specific to our application scenario. We then use FuelManager to send the query of the user from our app to the DialogFlow service, our bot receives the query and responds accordingly by sending the answer back to our application.

When the ChatActivity is started, we first see if the user has initiated the chat with our Bot or not. If he is chatting with some other contact, nothing is changed, and it works just as before, but if he has initiated the chat with our Bot, we first set up the Bot API details, that is the credentials and path to our bot so that Fuel could access it. Now, when the user sends the message, we first send a request to the bot with the user's query and also, store it

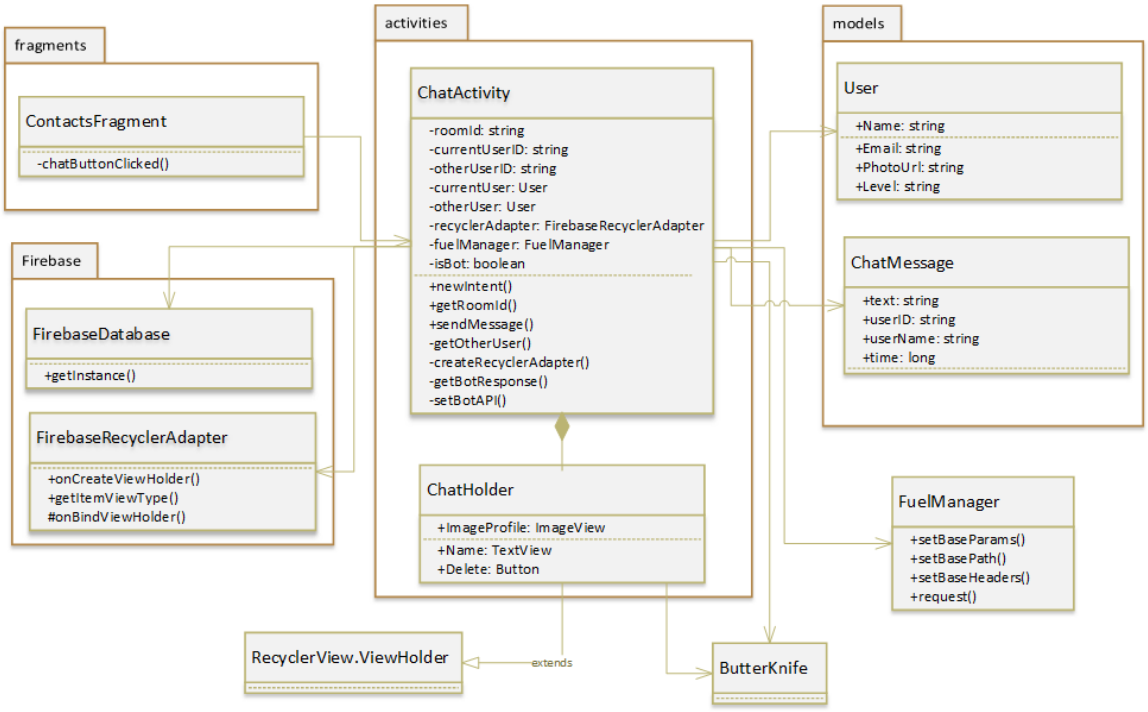

Figure 3.13: UML diagram for Chat Bot Integration

in the Firebase Database. We listen for the response from the Bot and upon receiving the response, we save it in the database and show the message to the user in the UI. In this way, he can interact with the Bot to resolve his queries.

By using this architecture, we can update and improve our bot without affecting the user's experience and without him, having to update the app again and again to get the latest version of the bot. We update and train the bot in the DialogFlow platform, and it gets updated for all the clients as they can call the updated bot using the same API as they were using before.

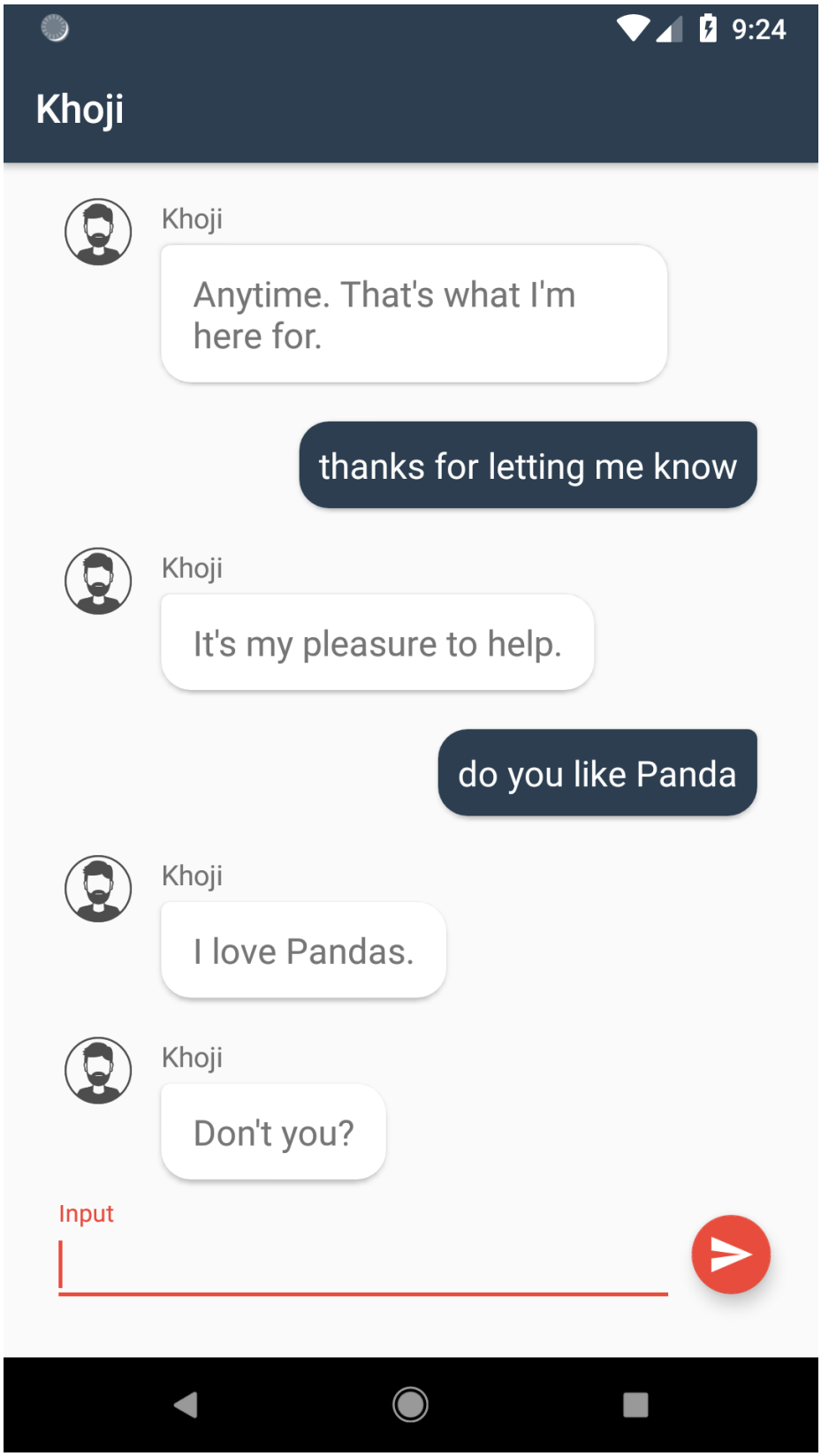

Figure 3.14: UI for Chat bot in Khoji

Chapter 4

Evaluation of Objectives

In this chapter, we will talk about the end result of all the effort that was put into this project and whether it was worth it or not at the end. We will evaluate as to what extent our implementation of the software solves the problem that it was intended to solve.

The core objectives set for this project, as defined in the section 1.4, were:

- Design and implement a system that tracks a user location and stores it in a remote database in real-time.
- Implement some social features so that a user can add other users into his contact list and keep track of their location at all times.
- Add real-time chatting functionality in the application for easier communication between users when navigating.
- Use Augmented Reality for the visualisation of the user marker in the real-world at any specific interval.
- Add in-app chat-bot that provides guidance and assistance to the user in using the app effectively.
- The devised system should be economical, for the end-user as well as for the developers.

We now analyse the extent that to which this project was successful in achieving these objectives.

4.1 Location Tracking

The first objective was to track each user and update his location details in a remote database in real-time. We have successfully implemented this feature in our application. As soon as the user logs in to the app, he is prompted to turn on his location services if they are disabled, and his location updates are immediately updated in the Firebase Database. If there is no network connection at that time, the location updates are stored locally, and they are updated to the database as soon as the device is back online.

4.2 Contacts

We have successfully completed this objective as well. The user can add other users as his contacts and is able to listen to the location updates of his contacts. We visualise the location of contacts using markers on a map. As soon as the location of the contact is updated in the database, the newly updated record is fetched from the database, and the marker of the contact shown on the map is also updated to reflect that change in real-time.

4.3 Real-time Chat

This objective has also been completed successfully. Two users can initiate a conversation with one another whenever they want. They will be able to message one another in real-time. If there is no network availability at the moment, we have enabled offline persistence in our app as well, so this will store the messages locally, and as soon as the device is back online, it will push the updates to the database.

4.4 Augmented Reality

We have also completed this objective partially. The user markers are shown in the AR scene at their specified GPS coordinates. The markers are stable, i.e. they do not float around in the AR scene as they used to before. Their position also gets accurate the longer they open that activity. By moving around after opening the AR scene, it updates its sense of the surrounding environment and adjusts its position accordingly.

4.5 Chat Bot Integration

We have successfully integrated a chatbot into our application using the REST API services offered by the DialogFlow platform. Users can query the bot by sending him messages through the app, and the bot will respond accordingly to their queries and try to help out the users in accomplishing their desired tasks.

4.6 Economic

Our devised solution is economical for both, end-users as well as the developers. For end-users, it is completely free. They do not have to pay a single dime in order to use the app. For the developers, it is also completely free, but there are some limitations to that, as discusses earlier. Under a certain usage limit, it is completely free, but after the app usage crosses that limit and uses additional resources, a small fee is charged. For the normal use case, the app will not go over the usage limits. So, unless the app becomes a nationwide phenomenon, it will remain free for the developers as well.

4.7 Summary

We have completed all the objectives that were stated at the start of the project. There is still space for some improvement, especially in the implementation of Augmented Reality feature, but with the passage of time, as the library matures and devices get powerful enough, the stability and the features will be improved as well.

Chapter 5

Conclusion and Future Work

In this chapter, we will provide the conclusion of our project along with some future enhancements and features that could be added to further improve the effectiveness, stability and robustness of the application.

5.1 Conclusion

In this project, our goal was to design an economic software system that could be used to track the location of a user and then use augmented reality to visualize his location in AR scene by using markers. We used Google Firebase as the back-end for our application which acts as an Authentication provider as well as a remote database. The location details of the user are extracted from his device and stored in the database in real-time. They are then fetched from the database and shown to the user's contacts, either in Google Maps or AR scene view if the device of the end-user supports it.

We also implemented a social system such that the users could add one another as their contacts. The user could be able to track the location of their contacts at any point in time. The position of the user is then visualized using markers on both, Google Maps as well as in AR scene where the AR marker is placed at the GPS coordinates of the user.

We also implemented real-time chat in this application so users could easily commu-

nicate with one another during the navigation without having to leave the app. We also integrated a chatbot into our app so users could ask him their queries and have a rewarding and enjoyable experience. We also managed to design this whole system in such a way that it is not a financial burden, for both the end-users as well as the developers.

5.2 Future Work

Certain elements in this project leave scope for further development. With almost any project that includes a software component, a list of future enhancements could be endless. In this chapter, we will highlight the general areas where extra work would benefit the system by increasing its usability, reliability, and effectiveness.

5.2.1 AR Stability Improvements

Although the stability of the markers shown in the AR scene has improved tenfold, there is always the need for further improvement. With the passage of time, as the library improves, the stability of the markers will also improve along the way. So, the software should be updated to use the latest ARCore version as soon as it is released.

5.2.2 VPS

GPS is not a very reliable source of information, especially in urban areas. Its result is usually off by a couple of meters, and sometimes more, which is not a big deal most of the time, but when it comes to visualizing the location of an object in AR, it matters a lot if the marker shows its position to be other than where it is. This low accuracy affects the user experience a lot. Google is developing a new kind of technology, Visual Positioning System (VPS) [27], that uses the live video feed along with GPS to better understand where you are in the world. This technology could also be used for indoor navigation as well. By using this technology, the effectiveness, reliability, and stability of our application will be

increased greatly, as we will be able to locate users inside buildings, on different floors, with better location awareness and stability of the markers.

5.2.3 User Groups

For now, in our application, a user can not group his contacts, meaning one group for family, another for friends, another for Naaraan Tour, etc. By introducing grouping functionality into this app, it will be a lot easier for the users to keep track of a huge number of users easily. They could, at one time, only decide to show the markers for a particular group, to see what they are up to and where without cluttering their space with those that are not required at the moment.

Once the grouping functionality is implemented, it will be easier to implement group chat messaging as well. They would be able to create chat groups for specific people in their contact list just like they can do in apps like WhatsApp and Messenger. For example, if they are going on a hiking trip and they create a group with all the participants of the trip, they would only choose to show the markers for that particular group for the time being, and they could also message one another for secure communication.

5.2.4 Guide in AR

To increase the user experience, a playful guide could also be added into the app. Once the user opens the AR activity, the guide would pop up in the AR scene, most probably in the form of an animal or bird or some other character, and it would guide him to his location, or it could just roam in the AR scene mindlessly. It could also respond to various queries from the user and try to company them as they navigate to their contacts.

5.2.5 Altitude Usage

We were planning to use the altitude of the user in our app as well. This will greatly increase the effectiveness of the app as the users will be able to see where a particular user is in the building and on which floor. This is a type of functionality that traditional 2D based systems cannot convey effectively and is better suited for an AR app. For now, the library does not support this functionality, but there is work being done on it so there is a strong chance that it might be available as a feature in the near future.

5.2.6 Summary

These were some of the things that came to our mind during the development of this project. We were planning to implement a few of them, but because of some unforeseen problems and the shortage of time, we were not able to do so. We are hoping that in the future, we, or someone else will take up the mantle and implement some of these features into this app which will make the usability and effectiveness of the app improve tenfold.

References

[1] R. Silva, J. C. Oliveira, and G. A. Giraldi, "Introduction to augmented reality," *National laboratory for scientific computation, Av. Getulio Vargas*, 2003.

[2] D. Schmalstieg and T. Hollerer, *Augmented Reality: Principles and Practice*, ser. Usability. Pearson Education, 2016. [Online]. Available: https://books.google.com.pk/books?id=qPU2DAAAQBAJ

[3] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997.

[4] N. Hachach-Haram. (2017) How augmented reality could change the future of surgery. [Online]. Available: https://www.ted.com/talks/nadine_hachach_haram_how_augmented_reality_could_change_the_future_of_surgery

[5] MapboxAR. (2017) Creating a world scale ar experience using geospatial data. [Online]. Available: https://youtu.be/vRmTn25xm7Q?t=12m8s

[6] G. Maps. (2018) Google maps ar. [Online]. Available: https://youtu.be/ogfYd705cRs?t=1h26m6s

[7] A. Craig, *Understanding Augmented Reality: Concepts and Applications*. Elsevier Science, 2013. [Online]. Available: https://books.google.com.pk/books?id=7_O5LaIC0SwC

[8] D. Moon. (2018) Mobile ar is evolving faster than you think. [Online]. Available: https://venturebeat.com/2018/05/19/mobile-ar-is-evolving-faster-than-you-think/

[9] Apple. (2018) Arkit developers. [Online]. Available: https://developer.apple.com/arkit/

[10] ARCore. (2018) Arcore developers. [Online]. Available: https://developers.google.com/ar/discover/

[11] A. Developers. (2018) Arcore supported devices. [Online]. Available: https://developers.google.com/ar/discover/supported-devices

[12] W. contributors. (2018) Firebase wikipedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Firebase&oldid=852183550

[13] Firebase. (2018) Firebase services. [Online]. Available: https://firebase.google.com/products/

[14] F. Developers. (2018) Firebase pricing. [Online]. Available: https://firebase.google.com/pricing/

[15] W. contributors. (2018) Model-view-presenter. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93presenter&oldid=849618475

[16] FirebaseUI. (2018) Firebase ui android. [Online]. Available: https://github.com/firebase/FirebaseUI-Android

[17] W. contributors. (2018) Nosql. [Online]. Available: https://en.wikipedia.org/w/index.php?title=NoSQL&oldid=851300456

[18] Firebase. (2018) Firebase database usage limits. [Online]. Available: https://firebase.google.com/docs/database/usage/limits

[19] Google. (2018) Background location limits. [Online]. Available: https://developer.android.com/about/versions/oreo/background-location-limits

[20] Android. (2018) Android permissions overview. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview

[21] Google. (2018) Fragments developer documentation. [Online]. Available: https://developer.android.com/guide/components/fragments

[22] Mapbox. (2018) Mapbox unity sdk. [Online]. Available: https://github.com/mapbox/mapbox-unity-sdk/

[23] appoly. (2018) Arcore location. [Online]. Available: https://github.com/appoly/ARCore-Location

[24] ARCore. (2018) Arcore sceneform. [Online]. Available: https://developers.google.com/ar/develop/java/sceneform/

[25] DialogFlow. (2018) Dialogflow site. [Online]. Available: https://dialogflow.com/

[26] D. Flow. (2018) Dialogflow pricing. [Online]. Available: https://dialogflow.com/pricing

[27] G. Developers. (2018) Google vps. [Online]. Available: https://youtu.be/ogfYd705cRs?t=1h27m1s