MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Table of Contents

Chapter 1. Introduction

1.1. Project context

We live in a world where we have the possibility to explore what's surrounding us in a way that virtually knows no boundaries. We have the possibility to stay connected almost continuously with each other, we can express ourselves freely in many different ways and we like to stay up-to-date regarding what is happening around. The way we do things are more accelerated then ever due to the continuous information flow that reaches us so we do not like to wait and we want things to happen instantly. This is even more true when our health is not in its best shape.

In the healthcare tings go slower due to the limitations of our bodies, of our nature but when we are in need of treatment or we get hospitalized, things appear to go slower and we get impatient because our lifestyle demands to be everywhere and everything to happen instantly. But staying connected also has positive sides, for example in cases of emergency the victim(s) do not have to wait long for the help to come, because due to the fast information flow the help can react and arrive really fast.

There is always a need for healthcare because we unfortunately tend to have health issues and because of this the healthcare system will always be occupied and many times agglomerated.

Thanks to the fast and constant development of the information systems new doors and possibilities appear from time to time which makes it possible to make our life easier and more comfortable. With the help of these innovations we are able to improve the healthcare in many ways.

One of these ways would be a new application that would be designed to facilitate the communication among doctors and patients and help to organize the work with patients and their data like medical sheets or contact information.

The purpose of this project is to define and construct a system capable of supervising in real time the seriously ill patients inside, near or even outside of a hospital. Starting from this we can deduce that the main users of this system will be doctors and patients.

The system will offer the possibility for different features which might facilitate the work of doctors and nurses, like: tracking their patient's location and status continuously, communicating with the patients through messaging, sending warning messages and initiating emergency calls when certain scenarios happen.

1.2. Motivation

The decision to develop the Patient Tracker system was born because I would like to help the workers and patients of the healthcare system. Nowadays smartphones are essential parts of our lives. Considering this, their portability and the fact that Android is one of the most used operating systems, choosing to develop and Android application was expedient because it is easy to use, user friendly and it can reach a lot of people in a very short time.

1.3. Structure

In the following the structure of the thesis will be presented broken down into chapters and briefly describing their content.

Chapter 1 (Introduction)
This chapter briefly describes the project context, the problem which could be solved by the proposed system and a motivation that confirms that why this project was chosen to be designed.

Chapter 2 (Project objectives)
This chapter contains the detailed presentation of proposed system's the project objectives.

Chapter 3 (Bibliographic research)
This chapter contains the necessary information that is required to develop a tracker application. A study and a comparison will be presented regarding the protocols and technologies that can be used in order to implement such a project. Among these we can differentiate between messaging, GPS and cell network protocols and technologies. This chapter also studies and compares some of the similar systems.

Chapter 4 (Analysis and theoretical foundation)
This chapter contains the principles of operation regarding the system and the different technologies used by the system. This chapter also presents the functional requirements of the system.

Chapter 5 (Detailed design and implementation)
This chapter contains the detailed description of how the system works and how it was designed and implemented. All of the systems components are described and also how they communicate among them.

Chapter 6 (Testing and validation)
This chapter contains the summary of the testing and validation phases of the project.

Chapter 7 (Installation and user manual)
This chapter explains how the system can be obtained, installed and gives instructions about how this system can be used in the right way.

Chapter 8 (Conclusions)
This chapter contains an analysis regarding how the system can be improved.

Chapter 2. Project Objectives

As the healthcare is more agglomerated than ever, sometimes it is hard for the doctors and nurses to keep track of every patient thus extending the time to treat them and ease their pain. In every hospital there are outpatients who can walk outside of the building but has to stay in the hospital for a period of time and every one of them must be supervised by nurses or doctors. The general practitioners also have responsibility for many patients who arrive from longer distances for example other districts of the city or, even from other villages.

With the help of a portable system the care givers and also the patients in a hospital would have an insurance that if anything happens to a patient the responsible person would know exactly where he/she is and can immediately consult with him/her even if they are not in the same area. This system would free some hands and save some time for the caregivers. The system would be deployed as an Android application which offers a possibility to the doctors and patients to talk, change some messages and pieces of important information in a simple way. Patients would also be able to call for help if needed, and -to make sure that the help arrives- send SOS messages with their exact address to the ambulance and also to their doctors who will be able to act fast. The system would give ensurance for the care givers by providing GPS coordinates - and other data if needed - of a patient.

Among the users surely can be expected some elderly people, who might not be able to easily use a mobile application. To help them the application offers a minimalistic, friendly user interface a text-to-speech functionality, which helps by reading the content from the screen.

2.1. Functional requirements

- Real time location tracking (*localization via cell network, GPS*) of patients in order to the doctors and nurses be able to act immediately and accurately if something wrong happens
- To implement the messaging (*communication between the client and service provider using the basic functionality of Android that controls the sending and receiving of messages*), which makes possible to reduce the preparation time for nurses, doctors
- Handle messaging with the help of speech-to-text features
- To implement emergency calls that can be initiated manually and sends automatically the address of patients

2.2. Non-Functional requirements

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. The plan for implementing *non-functional* requirements is detailed in the system architecture, because they are usually architecturally significant requirements.

In a nutshell, non-functional requirements define how a system supposed to be (ex: “the system shall be <requirement>”). The non-functional requirements are listed and explained in details below (Table 2.1).

Table 2.1 Non-functional Requirements

NFR 1 (Availability)	The system always has to be available because in healthcare cannot be permitted that a patient gets out of sight because of some faluires. The system will run on a server continuously and apart from server maintenance or a blackout there are no predictable special cases that can cause a faluire.
NFR 2 (Response Time)	The system has to be able to deliver fast user inputs and respond to them.
NFR 3 (Usability)	The system has to be designed in a way, that any user can easily understand how it should be used. This is supported with a minimalistic and simple user interface and a few instruction messages to help the user.
NFR 4 (Performance)	The system has a short response time because in healthcare is essential

2.3. Necessary resources

- Software:
  - Android L (version 5.0 - Lollipop) or newer version
  - Windows 10
  - Firebase Realtime Database API - to store the necessary information
  - JDK 1.8 + Android SDK - To create the Android application
  - Google API - to implement the GPS tracking feature
- Hardware:
  - Asus Zenfone 2: *intel atom quad core (1,8GHz), 2GB RAM* - Device used to test the application
  - Dell inspiron: *Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz (4 CPUs), ~ 2,3GHz, AMD Radeon HD 8600M Series, 6GB RAM* - Device used to develop and test the application

Chapter 3. Bibliographic Research

3.1. Emergency calls solutions (Principle of operation)

A SIM card is used to just authenticate the identity of a telephone user on a particular network. It also contains the account details which allows the phone user to make a phone call. The phone in itself is capable of making a call, but the SIM card must be able to connect to a particular network after verifying its identity.

The purpose of an emergency call is to call for help immediately but there are several factors on a mobile phone that can delay this extremely important call for help:

- Lack of SIM card[1]:
  Emergency calls do not need any network authentication. This is a well thought feature because in an unfortunate event when someone's life is at risk and every second counts, the time loss due to entering the PIN code and waiting for authentication might be deadly. This is why whenever somebody makes an emergency call, the network connects that person to the emergency desk without authenticating thus bypassing the function of a SIM (Subscriber Identification Module) card.

- Lock screen:
  Usually on every phone a lock screen is set to prevent accessing the device for everyone who is not the owner or not trusted by the owner. This security measure also can delay the emergency call for example when the owner is nervous or is in shock state and can not unlock the phone even for the third try; or even the owner is unconscious and somebody else should call for help not knowing how to unlock the phone. Because of these possibilities it was decided that the emergency call should bypass the lock screen and take the user straight to the dialer in order to be able to call for help as soon as possible.

- Weak network signal:
  It is possible to make emergency calls even if there is no network coverage for a provider (like Orange, T mobile, etc.). This is possible because emergency calls can be made through any available signal of any provider. If a network is too weak then the call will be made through a stronger network which is available at that time. It is important to be noted that if there are no available networks no calls nor emergency calls can be made.

3.2. GPS Tracking technologies (Principle of operation)

The Global Positioning System (GPS) is a worldwide radio-navigation system formed from a total of 24 satellites and their ground stations. The GPS system was initially designed for military purposes.

[1] https://www.quora.com/If-a-cell-phone-doesn-t-have-a-SIM-card-how-can-it-make-emergency-calls

Today mapping solutions are accustomed and important parts of everyday life. Many people use them to locate everything on the planet: to search for a country, a mountain, an ocean or to search for a specific address or just to get driving directions. Nowadays almost every piece of information has a location, and if it has a location than it can be put on a map. The use of this system is free of charge for the civil users.

GPS tracking is a method of telling where something is positioned. It can also be equipped in a car or put in a mobile phone. Since it's portable it is able to track the movement of a person or a car. The GPS tracking system uses the Global Navigation Satellite System (GNSS) network. This network consists of a set of satellites that transmit microwave signals to GPS devices to make available information like location, direction, time and speed so a GPS tracking system can offer real-time and also historic navigation. This system provides special signals for the devices which later process these signals and can calculate velocity, time and even three-dimensional views. There are 24 satellites that travel around the Earth each 12-hours and send signals from space that can be received by the end devices. The control of the Positioning System of different tracking stations that are located across the globe. These stations help in tracking signals from the GPS satellites.

A GPS tracking system can be used for many different purposes. From the commercial perspective, it can be used to track the route of vehicles. Some systems use passive tracking and others send the information to the centralized database. The passive system can monitor location and store this data in certain scenarios. This kind of system is capable of telling for example where a truck has been in the last 12 hours while an active system can do real-time tracking and automatically sends the current location to a center where the responsible people can track a truck in real-time for example. This kind of system is a better option for monitoring people (elderly, children, people with Alzheimer) because it allows a caregiver to see where are these people exactly and whether they are in a safe place or not. Real time tracking is also popular in security solutions because it allows the user to pinpoint the precise location of the device equipped with GPS tracker[3].

3.2.1. Location and Map APIs for Android and their comparison

Today we have many possibilities to choose from when we would like to benefit from the services provided by GPS. Among these let us examine a few applications (and the APIs provided by them) that are well known by society such as: Google Maps, Here, Waze. In the following they will be compared one by one with regard to the project objectives and the best will be chosen to be used in the Patient Tracker application.

Google Maps is an application for smartphones operating under Android OS. can be used. In order to be possible to use the Google Maps within our application we must rely on Google APIs which provides Add-ons for the Android SDK so the set of libraries and services – provided by Google – can be put in use. The Google Maps API can be installed inside the Android SDK Manager. The API automatically handles access to Google Maps servers and response to map gestures[20].

Without a doubt, it is the best overall navigation app for Android. Recently, Google has added the ability to download offline areas (which was there in a less official form

before), which is useful if you are heading to an area with poor connectivity but to save storage space on the mobile devices and ensure the offline maps to be up-to-date, offline maps expire within 15 days. Google Maps should also be preferred when traveling to the countryside. Google updates the information regularly and provides extensive details. The Google Maps API has the capabilities to:

- Get and work with detailed and vast information
- To be used from any location worldwide
- Add markers, polygons, and overlays to a basic map, and
- To change the user's view of a particular map area.
- To convert addresses into coordinates
- To convert coordinates into addresses
- Create info views on the map

Waze[2] is a community-driven application for smartphones or tablets that is capable of turn-by-turn navigation via GPS and sharing user-submitted travel times and route details. It serves the same purpose as Google Maps: to provide detailed information about the current location and to navigate the driver/traveler in the world. It mainly focuses on providing real-time traffic information. As a "wazer" (community member) one can report accidents, traffic jams, and similar incidents on the way. Friends can see each other's Estimated Time of Arrival (ETA) when driving to the same place and it also delivers a unique user experience. Waze may seem to take a more complicated route sometimes, but it knows how to avoid the traffic.

Waze supports Android and iPhone by providing the Waze Transport SDK, which is available for anybody who is a *Waze Transport SDK* partner and would like to develop mobile applications. With this SDK developers can build applications for navigating or getting location data and also many more things such as:

- Getting ETA and routing points (provides capabilities like drive and arrival time calculations, based on Waze's cloud data).
- Search for places in the application
- Set the Waze map to a desired location from the application

Unfortunately the Waze Transportation SDK does not support:

- Server-side access to Waze data, like traffic reports, speed, etc.
- Embedding the Waze map and navigation in your app
- Offline maps
- Detailed maps world wide

Here Maps is an application for Android and iOS that provides mapping and GPS navigation. It is currently available in more than 180 countries and it provides turn-by-turn walking navigation. It also uses an augmented reality technology called *LiveSight* that provides additional information like contact information, open hours and reviews about buildings if the user holds up the phone directs the camera onto the building.

It developed an API (Here Maps API) for Android. Applications built with this API are able to plan routes and interact with extruded 3D buildings. Here also created a

[2]https://books.google.ro/books?id=TehLDQAAQBAJ&pg=PP2&dq=waze+android&hl=en&sa=X&ved=0ahUKEwijjbLvouXaAhWHzKQKHSXfCucQ6wEIQTAE#v=onepage&q=waze%20android&f=false

service for iOS and it provides iPhone users with maps in more than 180 countries, also public transit, walking and driving directions. It is capable of voice guided navigation. Multiple map views are accessible: satellite view, public transportation view and live traffic view. Users are able to or remove new elements with the help of the Here Map Creator. This service is accessible on Android and iOS as well and gives the possibility to the users to add new roads, edit or remove them; add new places, edit or remove them; add new markets/shops, edit or remove them (*grocery store, clothing store, types of restaurants, sports equipment store*). In some cases: photos, working hours and contact details might be attached to these places. The service even permits to edit details such as house numbers, speed limits on roads, number of lanes, type of the road (*open road, tunnel, bridge, pavement, etc.*)[7].

From the users' point of view Here Maps is one of the simplest navigation apps for Android which in some people opinion "could look better". Recently, Here Maps rebranded to – "Here We Go". It has been tailored for offline use. In addition, it lets you download the map of an entire country. Unlike Google Maps, it does not offer an expiration period for the offline maps. It also provides a quick-action button to search for Nearby places. It is one the best alternatives to Google Maps.

Here Maps also launched an SDK to help the work of developers. There are many applications that use Here as backbone. The SDK provides access to multiple sets of data such as average road speeds, traffic build-up. In 2015 Here Mobile SDK was created which contains native Android and iOS APIs which permit many things that expand the list of advantages:

- displaying raster tile map
- search of online points of interest
- geo-coding/decoding and online route calculation
- using turn-by-turn guidance
- having the best method for offline use

By comparing the previously mentioned location APIs and the features and capabilities offered by them, Google Maps (and Google Maps API) had been chosen, because it is an obvious choice in many aspects:

- First of all, Google Maps API was developed by Google just like the Android OS and due to this fact, it can assure compatibility with any Android application that uses it
- It has Add-ons that can be installed directly in Android Studio (*Android SDK Manager*). As mentioned above the API automatically
- Handles access to Google Maps servers and their response to map gestures.
- It is suitable for any location worldwide which means it has an extremely large coverage.
- It can work with offline maps, which expire within 15 days to save memory. Usually this time is short, but it fits our goal perfectly, because it will not be used frequently just occasionally and even then just in some special cases.
- It is not capable of sharing real time traffic information among the members of the community like Waze, but in a hospital we do not need this kind of functionality.
- The UI is user friendly and the Google Maps API as a whole, fits perfectly to our needs.

3.3. Messaging protocols

In this subchapter different messaging protocols will be presented that will make it possible for the components of Patient Tracker project to communicate among each other in an easy and efficient way.

3.3.1. Advanced Message Queuing Protocol

The AMQP is an open standard that defines a protocol for systems to exchange messages. AMQP defines not only the interaction that happens between a consumer / producer and a broker, but also the over-the-wire representation of the messages and commands that are being exchanged. Since it specifies the wire format for messages, AMQP is truly interoperable, nothing is left to the interpretation of a particular vendor or hosting platform. AMQP provides both synchronous and asynchronous methods that can be called by the client. The difference is that synchronous methods return a response while the asynchronous don't return any response.

3.3.2. Hypertext Transfer Protocol

The HTTP[3] is an application protocol for distributed and collaborative systems. It uses a request-response protocol in the client-server computing model. For example, a web browser running on a computer can be the client and a web page running on another computer can be the server. The client sends a HTTP request to the server to which the server responds by providing the necessary resources (HTML files) or by executing the operations asked by the client and sending a response message to the client. The message contains a completion status about the request and it may contain in its message body a result obtained by executing the requested operation by the client or a requested file/information. It is designed in a way to permit a client-server communication even if there are some intermediate elements in the network.

HTTP resources are identified and located on the network by *Uniform Resource Locators* (URLs), using the *Uniform Resource Identifiers* (URI's) schemes *http* and *https*. HTTP uses new connections for every new request sent to the server, but this was improved by HTTP 1.1 which is able to reuse a connection multiple times thus has less latency.

A *HTTP session* is a sequence of request-response transactions. First of all, the client establishes a *Transmission Control Protocol* (TCP) connection to a port on the server and a HTTP server waits for the client's request message by listening usually on port 80. After this the server responds with a status response and the body of this message contains a requested resource. If the request was successful an OK status is returned, otherwise an error message. It has many methods to indicate towards the server what kind of action is requested. These most important methods are listed in Table 3.1 below:

[3] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Table 3.1 – Http request methods

GET	Retrieves data from the server and it should not have any other effect.
POST	Requests the server to accept the new entity as a new member of the resource identified by the URI. This entity can be a newsgroup, mailing list, a message or even a block of data that can be added to a database/
PUT	Requests to store the sent data under the provided URI. If at the destination already exists such an entity it will be updated, but if the URI points to a location where does not exist a resource, then the server creates that particular resource coded in the URI.
DELETE	Deletes the specified resource
HEAD	Similarly to GET it requests a response but without the body. It might be useful at retrieving meta-data about some entities without having to transport the whole.
TRACE	Echoes the received request so the client can notice if any changes were made by the intermediate servers.
OPTIONS	Returns the HTTP methods supported for the specified URL
PATCH	Applies modifications to a resource specified by the URI

HTTP status codes[4] are very simple and well structured. Every status code is a three-digit number and the first digit defines the type of the status code. There are five categories that are dedicated for different types of status codes. Status codes whose first digit is 1 are *informational* codes. Codes starting with 2 are *successful* codes, starting with 3 are *redirection* codes, starting with 4 are *client error* codes and codes starting with 5 are *server error* codes.

HTTP sends messages in plain text format (ASCII). The client sends the *request* and the server responds with a *response*.

- Request message consists of:
  - A request line (e.g. GET/PUT/DELETE/etc.)
  - Request header fields (e.g. Language: en)
  - Empty line
  - A message body which is optional

- Response message consists of
  - Status line which includes the indispensable status code (*200 OK*, *client's request succeeded*)
  - Response header fields (*context-type: text/html*)
  - Empty line
  - A message body which is optional

[4] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

3.3.3. Signalling System

The **Signalling System No.7**[5] is a set of protocols used for telephony signaling. It is used for more telephone related mechanisms among which (regarding the project) the following can be enumerated: setting up/tearing down telephone calls and Short Message Service. It uses full-duplex links to connect and communicate two nodes (phones) of the system.

3.4. Messaging technologies

During the documentation and research phase of the thesis the following messaging technologies were found, from which later I chose one that fits the most to my project considering all the project objectives and carefully analyzing what these technologies can offer.

RabbitMQ is a message broker[6] software implemented by Erlang that implements the AMQP broker. Erlang is able to run on any operating system. RabbitMQ can easily be extended with the addition of some plugins:

Android system: RabbitMQ can be used since there is a Java Client Library that RabbitMQ implemented and allows Java code to interface with the queue. It allows to use a set of methods for basic (ack, cancel, deliver, consume, get, publish, reject), channel (close, flow, open), exchange (bind, declare, delete, unbind) and queue (bind, declare, delete, purge, unbind) operations[10].

In the first phases of the Patient Tracker's design while looking for a messaging technology RabbitMQ was found but later it seemed that with Firebase better results can be achieved.

Google released **Firebase** in the summer of 2016 with the purpose of giving an infrastructure and support that might be needed on order to build great applications. It is not a replacement to the existing APIs like Android and iOS. It is rather an enhancement, that offers things that might prove useful (database, security, messaging, etc.) for building an application. This helps the developers because they do not need to implement all these components from scratch and saves time. It also offers some technologies that can be used in the application. Each of the technologies can be used separately. The developers are not forced to use them all. Many of these are free to use (Authentication, Cloud Messaging, Notifications, etc.). The other technologies have a free version that has limitations and can be used for testing and for smaller apps. For example, the free tier version of Realtime Database can store 1Gb of Data and have 100 simultaneous connections. Similarly, for Cloud Functions the free tier allows a number of 125,000 invocations per month [15].

Firebase uses **Representational State Transfer** (REST) API[7] in a way that any Firebase Database URL can be a REST endpoint so to send a JSON object all is need to be done is appending the *.json* to the end of the URL and send a HTTPS request from the client. Since REST is an architectural style that defines a set of rules and properties based

[5] https://en.wikipedia.org/wiki/Signalling_System_No._7
[6] www.cloudamqp.com/blog/2015-07-29-rabbitmq-on-android.html
[7] https://firebase.google.com/docs/reference/rest/database/

on HTTP basically the Firebase Real-time Database uses HTTP in the background to work with the data, which is written in JSON objects and structured in a tree of these objects.

For a real-time database it is not enough only to use HTTP because this requires constant requests from the client side to see if there were any changes in the database. This would not be efficient, occupy a large bandwidth and increase latency. In the Firebase Real-time Database **Server-sent Events (SSE)** is also used to synchronize the data with the client whenever a change is being made in the database. SEE is a technology where a client receives updates automatically form a server via HTTP connection. Server-sent events basically is a standard that describes how servers can initiate data transmission towards clients once the initial client connection has been established. They are mainly used to send continuous data streams to a client browser or device.

Firebase offers the possibility to do many different things by using tools. We will discuss the ones that were used for the project in the table below (Table 3.2).

Table 3.2 – Firebase tools used in the project[8]

	Description
Authentication	Provides backend services, easy to use SDKs to authenticate the project's users. It supports authentication by e-mails, phone numbers, passwords and it is able handle Facebook, Google and Twitter integrations.
(Real-time) Database	Stores data -that is coming from authenticated users- in a JSON tree. The data is synchronized and every one of the connected clients will receive the changes in the data as soon as they happen. All of the users share one Realtime Database instance.

These solutions have been chosen because these were the ones that fitted the best to the project's nature. Namely here they are and a few notes why they have been chosen:

Authentication as many other authentication services by knowing the user's identity allows the application to safely and securely store data. Firebase Authentication enables to save user data in the cloud thus providing the same experience regardless of the user's device.

Each user has a unique Identifier, a time stamp that stores the date on which the user account was created, a time stamp with the date of the last sign-in and a very important and also unique User UID. It also worth noticing that the password is nowhere to be found, and not only in the admin's user interface, but also the Firebase SDK does not provide any means to retrieve the user's password programmatically. This is intentional to keep the applications more secure. The password can be retrieved only by the forgot password feature which is accessible only through the user's email account.

The key capabilities are *FirebaseUI Auth* and *Firebase SDK Authentication*.

[8] https://en.wikipedia.org/wiki/Firebase#Firebase_Auth

The first one holds a complete sign-in system and it can be integrated as a whole in applications. The second one is able to handle the basic email, password and phone number authentication. There are some other possibilities like anonymous authentication that lets the user to explore some of the basic features of an application in “guest” mode and later this temporary account can be upgraded to a regular account.

This service has been chosen because it provides a free, easy to use authentication system that is secure, stable, has an already implemented forgot password feature, and it even has Google, Facebook integrations.

Real-time Database is a cloud-hosted, NoSQL-based database. It is able to synchronize connected devices having an event driven database. In its database there is no server-side code instead all of the code is written in the client side. Whenever a data change occurs in the database, events are fired on the client side and later these can be handled.

This database is organizing all the data in a tree structure. The nodes of this tree are JSON objects which can be manipulated by giving their path. For example, let us a user can be accessed by giving its path which is root\users\id. If a new node is added to the database it will get a unique ID and knowing this id the node can be read or modified instantly, without the need to search and match the data in order to locate the node again.

If any modification happens in this structure every user with access to this database and to that specific data will be synchronized with the changes in seconds. It also works with applications with different version number, or among different platforms (web, mobile).

The Real-time Database is integrated with the Authentication service which enables only the authenticated users to read or write data in the database considering that they do not have any restrictions regarding the data in the database.

This database is optimized for offline use too. When a device is offline it is inevitable to lose connection with the real-time database, but with offline sync the developer can get the most out of the application since in offline mode a cache is used with the purpose to store the changes in the data. Later when the internet connection is restored all the changes in the cache are synchronized with the database, so no time or work is lost. The cache has to be enabled programmatically by the developer because not every application requires the offline mode.

The Real-time Database is really helpful because it offers solutions (through the Firebase SDK) to store and read data, which is essential for the project. The application is built on the Real-time database, which forms the backbone of my project and it also resolves many problems for one main feature in the application which namely is the *patient tracking.* Tracking in real time can be a complex task due to the synchronization of the coordinates. The Real-time Database synchronizes solves this problem by instantly feeding the new coordinates to the tracker as soon as they were changed. All these operations has to be executed securely which is ensured by the Firebase Realtime Database Security Rules. These rules are a set of rules which defines how the data is structured and who have access to that data. Its purpose is to make it easier to build a real-time experience. Storing data is a common requirement that appear in apps.

A new version of the Firebase Real-time Database has been identified that has the name *Cloud Firestore*[9]. This is also a cloud-hosted, NoSQL database that is similar to the Firebase Database. It is a Firebase product as well and it is built on its ancestor's (Firebase Realtime Database) success.

Similarly to Real-time Database is capable of keeping all the data synchronized accross many platforms, devices and even application version numbers which is achieved through realtime listeners. It offers offline support just like the Real-time Database that also works on mobile and web as well.

Just like its ancestor Cloud Firestore offers integration with many other tools. Among these we can find all the products from Google Cloud Platform.

Its main capabilities are *felxibility, expressive querying, realtime updates, offline support and also scalability.*

The data in the Cloud Firestore database is stored otherwise. The NoSQL data model stores data in documents which contain fields mapped to values. All these documents are stored in some collections which are basically containers that are used to organize the data. By using this data model Cloud Firestore supports any datastructure that that fits the project's needs.

It is also capable of querying data which is efficient and flexible. It is capable of creating shallow queries to get data at document level without needing to read the whole collection or sub-collections. Sorting and filtering can be applied on the queries in order to narrow down the results.

To keep the applications up-to-date with data Firestore uses real-time listeners just like the Firebase Real-time Database. It uses Firebase Authentication to secure and limit the access to the real-time database.

As a conclusion it can be said that despite of all the new features of the Cloud Firestore[10] and despite the fact that it is newer and maybe more performant compared to its ancestor Firebase Real-time Database. The Real-time Database was chosen instead of Firestore because Firestore is brand new, which also means that it is not entirely tested and cannot be foreseen all the effects that will appear using Firestore. On the Firebase's homepage is even stated that Firestore is still considered as "Beta"[11].

Firebase Real-time Database had been chosen to make sure that there will be no surprises regarding the stability, but the necessary tools and features are doing what is expected.

Cloud Storage[12] offers the possibility to upload and download different kind of files to the cloud from the application.

Among the main capabilities we can mention the *robust operations* and *strong security*. The first assures that in any case the files reach their destination. This is achieved by restarting uploads or downloads if they are stopped and uploading/downloading files even if the network quality is poor. The second one means

[9] https://firebase.google.com/docs/firestore/
[10] https://firebase.google.com/docs/database/rtdb-vs-firestore
[11] https://cloud.google.com/firestore/docs/
[12] https://firebase.google.com/docs/storage/?authuser=0

that the Cloud Storage can provide access based on filenames, sizes or the type of their content.

This service was not used because the users of my project on one hand need a minimalistic and performant application while on the other hand they would not benefit from being able to upload and download files.

3.5. Speech recognition applicability for patient tracking

Android devices provide two main voice-based functionalities: text-to-speech and speech-to-text and both serve the purpose to help the interaction by voice with smart devices.

Speech-to-text: with this feature users of Android devices can dictate into any text box on the device where textual input might be required. E-mails, text messages and even searches all belong in this category. The keyboard has a built-in button with a microphone symbol on it. The user can initiate a speech-to-text operation by pressing this button and the spoken input is automatically transcribed into written text. The user can then decide what should happen with the transcribed text. Text-to-speech also can be used in a way to enable the user to search by voice[13].

The results have improved considerably for using it on small devices due to the use of cloud-based resources for speech recognition and also these devices are held close to the mouth which provides better audio signal.

There are many difficulties with this technology. One of them is that the input is practically unpredictable because users can say anything and an incredibly large vocabulary is needed to cover all the possible inputs.

Text-to-speech: Text-to-speech is used to convert text to speech. Various applications can take advantage of text-to-speech. Many applications take advantage of this functionality like Google Translate, different Fitness Applications (Home Workout, 7 Minute Workout, etc.) and Talk-Back for example uses text-to-speech to help blind and visually impaired users by describing what items are touched. Talk-Back can also be used to read a book in the Google Play Books application. The text-to-speech is also available on Android Kindle as well as on Google Maps for giving step-by-step driving instructions. There is a wide range of third-party applications that make use of text-to-speech[13].

3.6. Competitors analysis

Competitor analysis plays an important role in strategic planning. It helps in gaining awareness of the competitors' past, present and future strategies and provide an informed basis to develop strategies to achieve competitive advantage in the future.

Competitive analysis directs mainly at the competitors who target the same market. The table below shows the major competitors who were all implemented for Android. The source of competitor information is gathered from Google Play [13]. Some of the major competitors were tired and each one of their advantages and disadvantages were observed and analysed. These characteristics have been compared bearing in mind

[13] https://play.google.com/store/apps

this project's main objectives/features which are discussed in the following table(Table 3.3):

Table 3.3 Comparison of the proposed system with similar, already existing systems

Feature	**Remote Patient Monitoring**	**Medocity CHF**	**Caring Buddy**	**Proposed system**
Location Tracking	✔	✘	✔	✔
Messaging	✘	✔	✘	✔
Text-to-speech	✘	✘	✘	✔
Track medications	✘	✔	✘	✘
Call Doctor	✔	✘	✘	✔
Emergency call	✘	✘	✘	✔

Symbol interpretations:
✔ - exists an implementation
✘ - does not exist an implementation

The *'Remote Patient Monitoring'* runs on Android devices and these devices' sensors are used to monitor a patient's activity. The doctors can create geofences to get notified if a patient is leaving the defined area. It is not capable of providing communication functionalities like messaging. It has no text-to-speech feature which helps the elderly to navigate and interact better with the application and finally it does not provide the possibility for emergency calls.

The *'Medocity CHF'* also runs on Android. Among its capabilities one can find a possibility to chat, medication tracking and a collection of medical resources. The application can't track the location of patients nor can read out loud the texts appearing on the screen.

'Caring Buddy' is an Android application that is capable location tracking. There is no possibility for messaging, initiating emergency calls or listening to the written text.

The *Patient Tracker* application is a powerful tool that can offer more than any of its competitors by being able to:

Chapter 4. Analysis and Theoretical Foundation

4.1. Conceptual architecture

The proposed system is implemented using the client-server architecture as you can see in the Figure 4.1 – Simplified Conceptual Architecture below which ilustrates the simplified conceptual architecture of the system and is meant to offer an insight in how the main components of the system are connected.

By being one of the most widely used architectures in software development because of its outstanding reliability, the client-server architecture assumes that there is at least a service provider and a client who needs that particular service. The architecture also assumes that these two main components are part of a distributed system in which the tasks are shared among the service providers, called servers and the service requesters, called clients.

Mostly the clients and servers are communicating through the Internet, just like in our case, because this methodology of communication gives us the benefits of sharing the data fast and also securely.

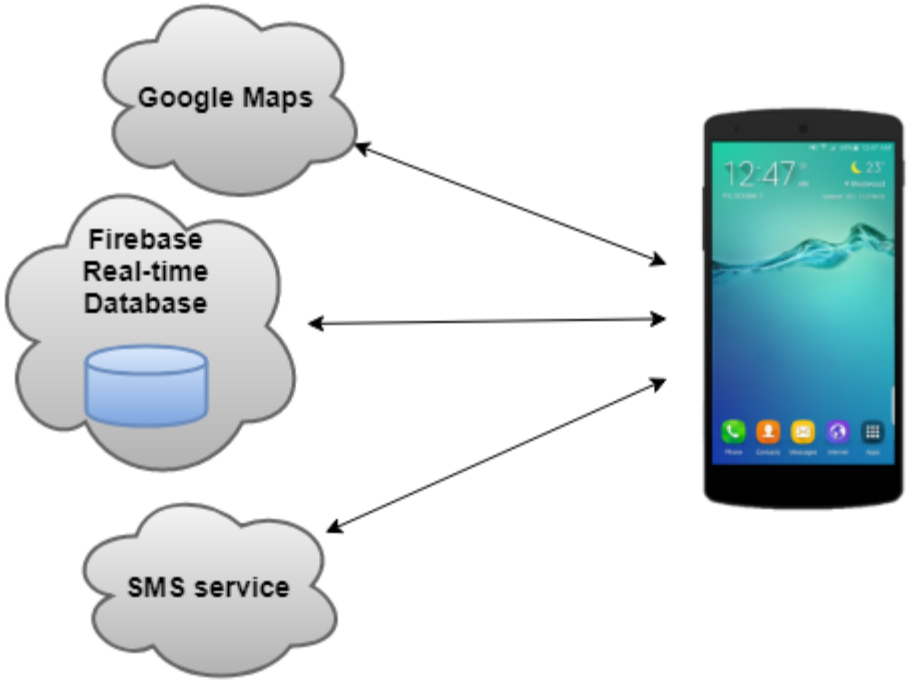

Figure 4.1 – Simplified Conceptual Architecture

As the Figure 4.1 – Simplified Conceptual Architecture above presents in the simplified, high level conceptual architecture of the project, there are two kinds of (distributed) components which are the following:

- Server: the Firebase Real-time Database, the Google Maps and the SMS service represent the servers of the system.
- Client: a mobile device that is used by patients or doctors.

After the client initiated and established a connection with the server(s) it sends its user data and then it requests information about other users (doctors or patients). The client also requests GPS coordinates from Google Maps and it forwards to the Real-time Database.

The Real-time Database stores the information provided by the clients and shares (some part of it) with other clients. Whenever a client changes something in the database, it will send a synchronization request towards all the other clients in order to notify them about the changes and fetch the new data.

4.2. Use cases

There are two types of actors (Figure 4.2 – User types) that can use the system, both having some unique (see Table 4.1 – List of user actions) actions. We will analyze both actors together broken down into features alongside with the associated use cases.

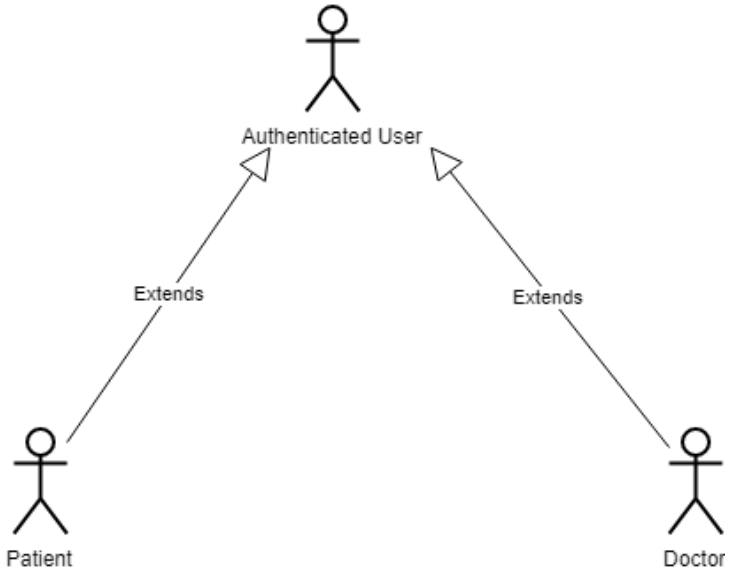

Figure 4.2 – User types

Table 4.1 – List of user actions

User Action number	User Action
UA 1	(Login Page)
UA 1.1	Initiate recover password
UA 1.2	Initiate signup
UA 2	(Signup Page)
UA 2.1	Signup
UA 2.2	Return to login
UA 3	(Recover password Page)
UA 3.1	Recover password
UA 3.2	Return to login
UA 4	(Patient User Main Page)
UA 4.1	Initiate Emergency call
UA 4.2	View/scroll contacts (doctors)
UA 4.3	Send SMS to doctor
UA 4.4	Call doctor
UA 4.5	View/edit profile information
UA 4.6	Listen to diagnosis or treatment
UA 4.7	Signout
UA 5	(Doctor User Main Page)
UA 5.1	View/scroll contacts (patients)
UA 5.2	Send SMS to patient
UA 5.3	View patient's position
UA 5.4	View/edit patient's diagnosis or treatment
UA 5.5	Delete patient
UA 5.6	View/edit profile information
UA 5.7	Signout
UA 6	(View/ Edit Profile Page)
UA 6.1	Edit information
UA 6.2	Add patients (only for doctors)
UA 6.3	Edit CNP (only for patients)
UA 6.4	Save changes
UA 6.5	Signout
UA 6.6	Go back to Main Page

4.2.1. Patient actor

The patient use case diagram is illustrated in Figure 4.3 – Patient Use Case Diagram below. The patient is the user that has installed the Patient Tracker application, but being a patient gives access to the operations that are relevant to a patient. The patient uses the application to message the doctors or to call for help. These are the main functionalities of the application which will be presented later with more details. Since this is a minimalistic application to help the patients -as

the use case shows us- they can do four things which can be divided into two main functionalities and two helper functionalities:

- Main functionalities:
  - Call help: the patient is able to call for help immediately if needed.
  - Send messages to doctors: the patient can send messages to the doctor if he/she has an important question or needs medical attention.
  - Change profile

- Helper functionalities:
  - Speech-to-text: the patient can tell the message and the application writes it in the message box, after which the user just has to hit the send button.
  - Text-to-speech: if the user cannot see properly a text, just by tapping on it the application will read out loud the text.

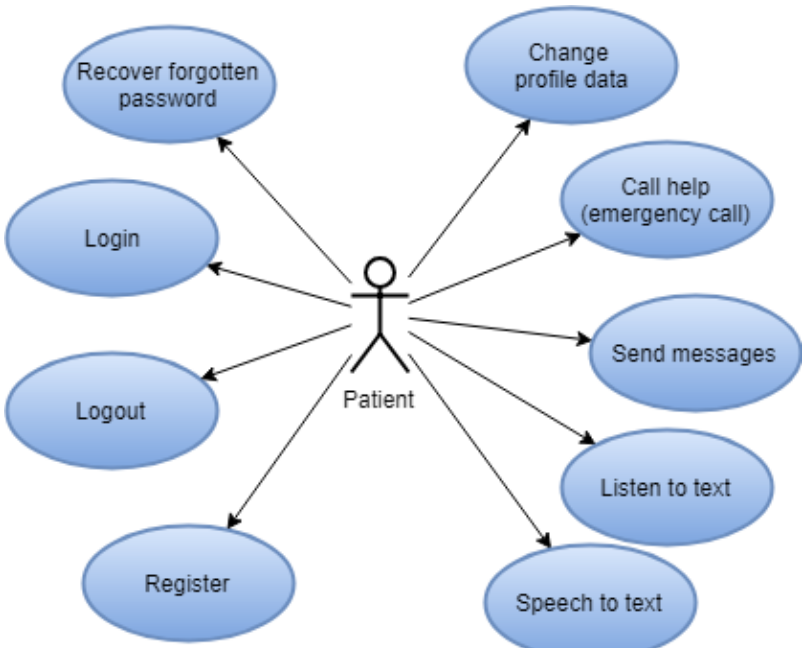

Figure 4.3 – Patient Use Case Diagram

4.2.2. Doctor actor

The doctor use case diagram is illustrated in

Figure 4.4 – Doctor Use Case Diagram below. The doctor is the user that has installed the Patient Tracker application but being a doctor (or nurse in some cases) gives access to the operations that are relevant to a doctor. The doctor uses the application to answer to the messages sent by the patients or to verify that everything is in order with the patient and the patient is in that area where he/she supposed to be. These are the main functionalities of the application which will be presented later with more details.

Since this is a minimalistic application to help the work of the doctors -as the use case shows us- they can do two things with this application:

- Main functionalities

- View and track the position of patients
- View patient profile and edit the diagnosis and treatment attributes
- Sent messages to patients. The doctor can send messages to the patients if he/she answers to an important question or needs to announce important news
- Change profile

- Helper functionalities
  - Speech-to-text: the doctor can tell the message and the application writes it in the message box, after which the user just has to hit the send button.

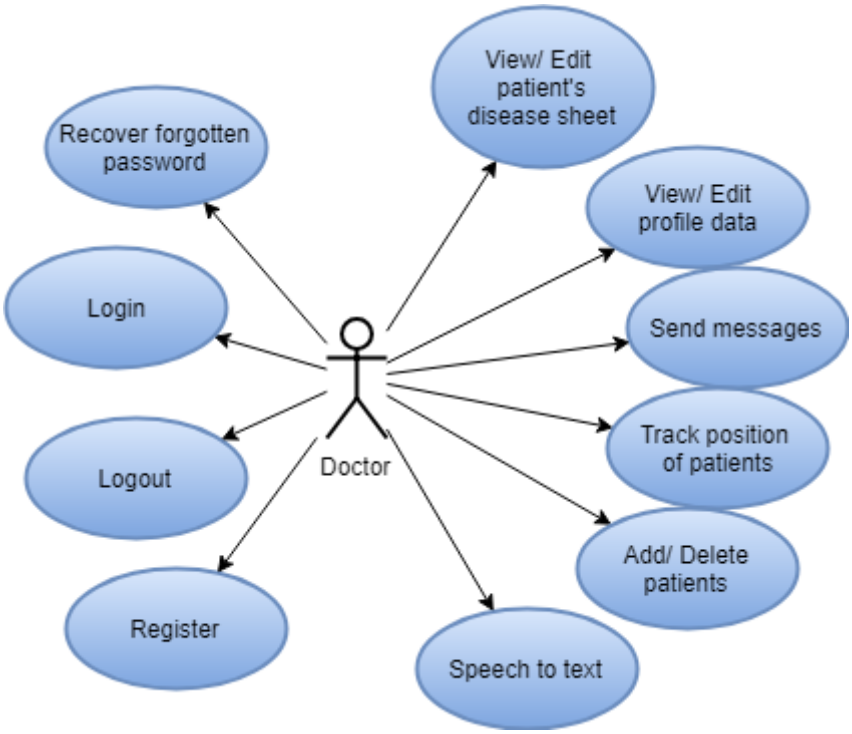

Figure 4.4 – Doctor Use Case Diagram

4.2.3. Emergency call detailed use case

In the following the Use Case for Emergency Calls will be presented, which is a base activity of the system. The figure below (Figure 4.5 - Emergency call flowchart) will present the workflow of the Emergency Call process.

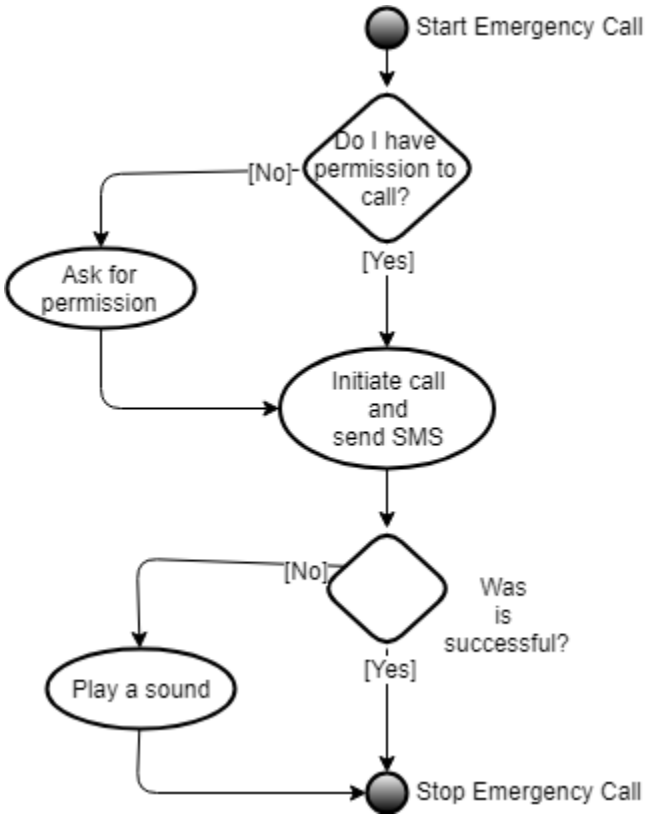

Figure 4.5 - Emergency call flowchart

1. **Basic flow**

Use-Case Start

This use case starts when the actor (patient) wants to initiate an emergency call.

1.1. The actor starts the aplication.
1.2. The system checks the granted permissions.
1.3. The actor initiates an emergency call operation.
1.4. The system calls the predefined number (112).
1.5. The system sends out som SOS messages to make sure there is an answer.

Use-Case End

The actor closes the application.

2. **Alternative Flows**

At any time, System fails:

To support recovery and correct communication, ensure all transaction sensitive states and events can be recovered from any step of the scenario.

2.1. Patient reenters the system.
This flow can occur in one of the following steps: 1.2, 1.4
2.1.1. System detects anomalies preventing to make a call:
2.1.1.1. The system checks the (*phone_call* and *write_sms*) premissions
2.1.1.2. The system starts a sound alarm on the speakers
2.1.2. Session timeout
2.1.2.1. System signals error, rejects entry and asks again for credentials.
2.1.3. The actor selects the Back operation and leaves to the home page.
2.1.3.1. The system remains unchanged

3. Special Requirements

A mobile phone with Android OS 4.0.0 or newer.

4. Preconditions

- The actor is authenticated and authorized for this use case.
- An emergency call scenario is opened through an initiate emergency call operation.
- All necessary permissions are granted

5. Postconditions

- Scenario entities should remain in a consistent state if the emergency call had effect (there was an answer).

4.2.4. Track patient detailed use case

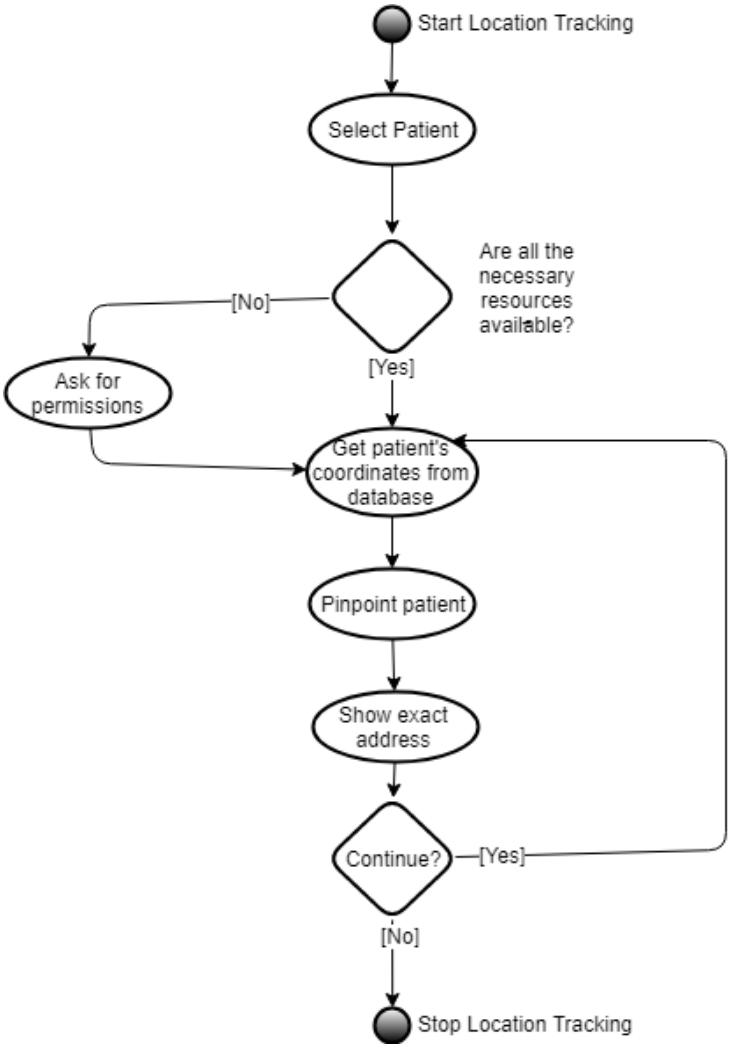

Figure 4.6 – Patient tracking flowchart

1. Basic flow

Use-Case Start

This use case starts when the actor (doctor) wants to verify a patient's location.

1.1. The actor starts the aplication.
1.2. The system system provides the available – and trackable - patients.
1.3. The actor selects an existing patient to track his/her location.
1.4. The system collects the necessary permissions.
1.5. The system collects the necessary resources (coordinates, address).
1.6. The system pinpoints the patient's location.
1.7. The system displays the patient's address.
1.8. The system repeats the previous steps (1.5, 1.7) until the actor does not want to continue.

Use-Case End

The actor ends the pinpoint location operation 1.8.

2. Alternative Flows

At any time, System fails:
To support recovery and correct location tracking, ensure all transaction sensitive states and events can be recovered from any step of the scenario.

2.1. Doctor reenters the system.
This flow can occur in one of the following steps: 1.2, 1.4, 1.5, 1.6, 1.7
2.1.1. System detects anomalies preventing location tracking:
2.1.1.1. The system checks the (*internet* and *location*) premissions
2.1.1.2. The system starts a sound alarm on the speakers
2.1.1.3. Doctor continues patient tracking
2.1.2. Session timeout
2.1.2.1. System signals error, rejects entry and asks again for credentials.
2.1.3. The system does not get coordinates from the patient:
2.1.3.1. The last know location of the patient is displayed
2.1.3.2.
2.2. The actor selects the Back operation and leaves the pinpoint location page
2.2.1. The system remains unchanged
2.3. The GPS signal is weak
This flow can occur in the step 1.5

3. Special Requirements
A mobile phone with Android OS 4.0.0 or newer.

4. Preconditions

- The actor is authenticated and authorized for this use case.
- A pinpoint location scenario is opened through a pinpoint location operation.
- All necessary permissions are granted
- There is GPS signal in the area
- There is internet connection

5. Postconditions

- Scenario entities should remain in a consistent state if the track location has ended with success.

4.2.5. Call Doctor detailed use case

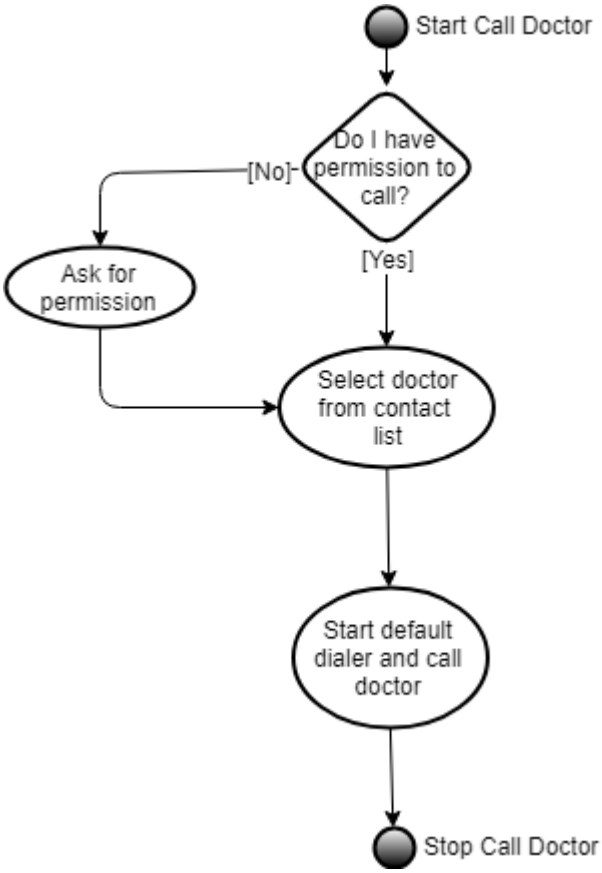

Figure 4.7 – Datailed call doctor flowchart

1. **Basic flow**

   Use-Case Start

   This use case starts when the actor (patient) wants to initiate a call towards a doctor.

   1.1. The actor starts the aplication.
   1.2. The system checks the granted permissions.
   1.3. The actor selects the doctor who will be called.
   1.4. The system calls the doctor using the default dialer on the mobile phone.

   Use-Case End

   The actor ends the call.

2. **Alternative Flows**

   At any time, System fails:
   To support recovery and correct communication, ensure all transaction sensitive states and events can be recovered from any step of the scenario.

2.1. Patient reenters the system.
This flow can occur in one of the following steps: 1.2, 1.4
2.1.1. System detects anomalies preventing to make a call:
2.1.1.1. The system checks the (*phone_call*) premissions
2.1.1.2. The system asks for permission to call
2.1.2. Session timeout
2.1.2.1. System signals error, rejects entry and asks again for credentials.
2.1.3. The actor selects the Back operation and leaves to the home page.
2.1.3.1. The system remains unchanged

3. Special Requirements

A mobile phone with Android OS 4.0.0 or newer.

4. Preconditions

- The actor is authenticated and authorized for this use case.
- A call doctor scenario is opened through selecting a doctor and starting a call operation.
- All necessary permissions are granted

5. Postconditions

Scenario entities should remain in a consistent state if the doctor call had effect or if there was no answer to the call.

4.2.6. Send message detailed use case

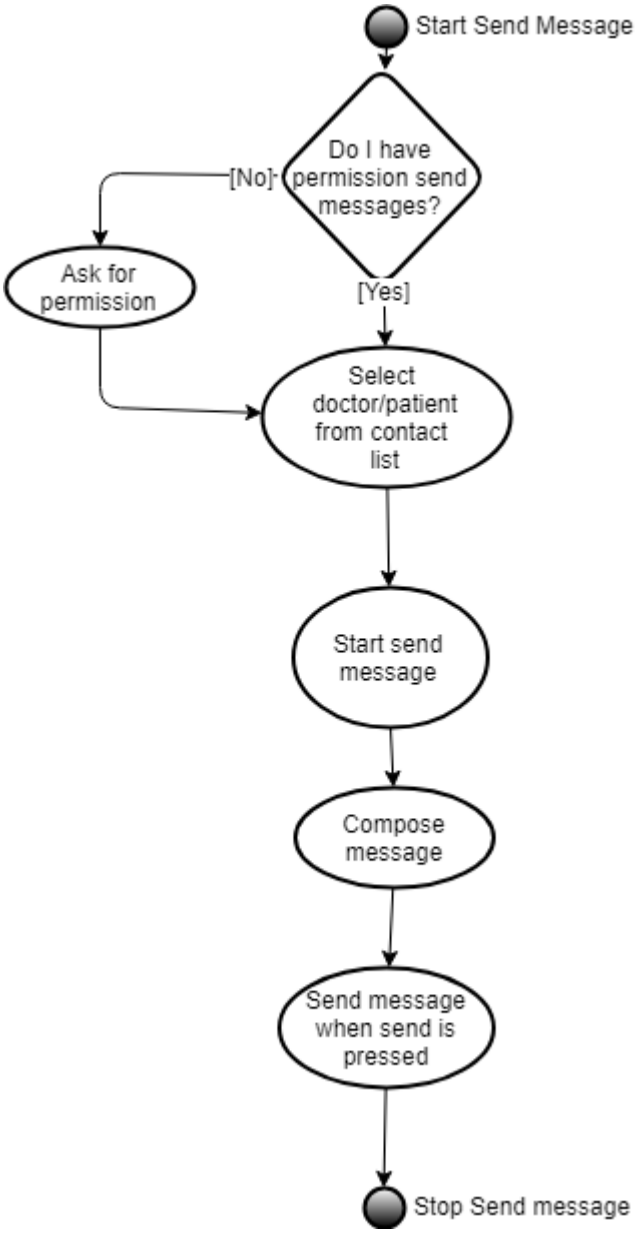

Figure 4.8 – Send message flowchart

1. Basic flow

Use-Case Start

This use case starts when the actor (patient/ doctor) wants to send a message towards a doctor or patient respectively.

1.1. The actor starts the aplication.
1.2. The system checks the granted permissions.
1.3. The actor selects the doctor (or patient) who will be messaged.

1.4. The system initiates the send message operation.
1.5. The actor composes the message in the text editor.
1.6. On send pressed the system sends the message.

Use-Case End

The system terminates the send message operation.

2. Alternative Flows

At any time, System fails:
To support recovery and correct communication, ensure all transaction sensitive states and events can be recovered from any step of the scenario.

2.1. Patient/ doctor reenters the system.
This flow can occur in one of the following steps: 1.2, 1.4
2.1.1. System detects anomalies preventing to send a message:
2.1.1.1. The system checks the (*send_SMS*) premissions
2.1.1.2. The system asks for permission to send message
2.1.2. Session timeout
2.1.2.1. System signals error, rejects entry and asks again for credentials.
2.1.3. The actor selects the Back operation and leaves to the home page.
2.1.3.1. The system remains unchanged

3. Special Requirements

A mobile phone with Android OS 4.0.0 or newer.

4. Preconditions

- The actor is authenticated and authorized for this use case.
- A call doctor scenario is opened through selecting a doctor and starting a call operation.
- All necessary permissions are granted

5. Postconditions

Scenario entities should remain in a consistent state if the send message had effect.

4.2.7. Change user profile detailed use case

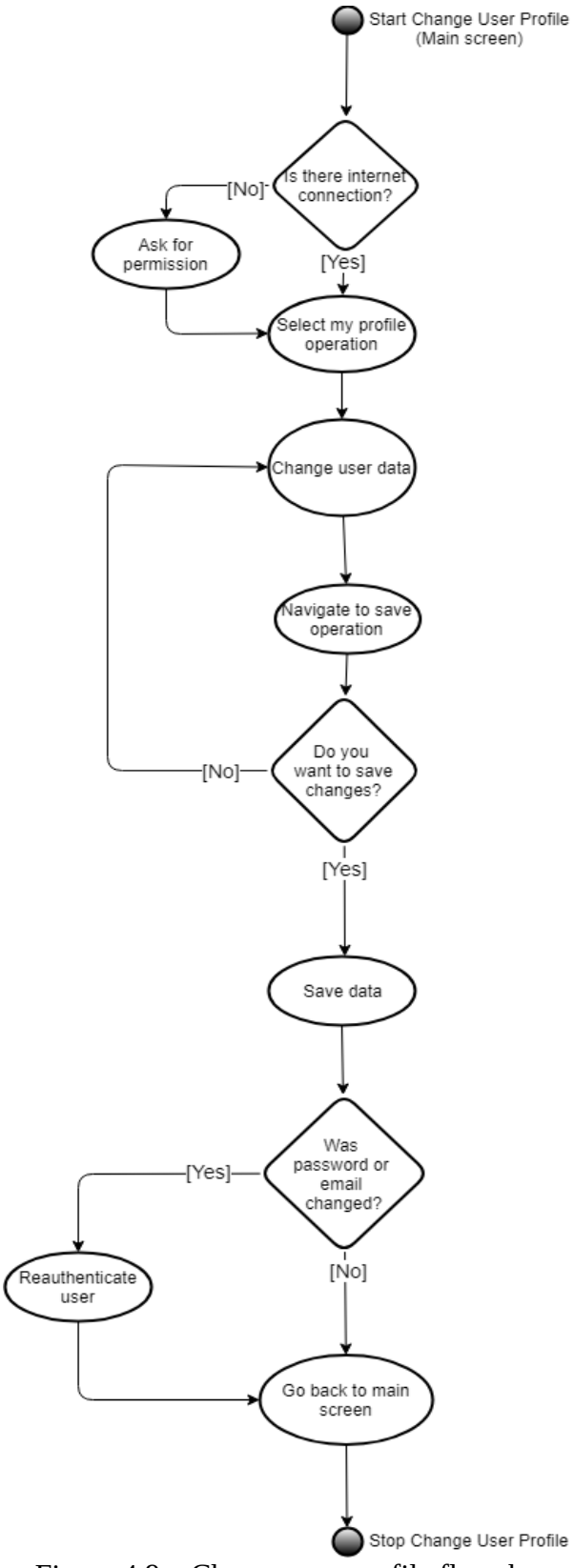

Figure 4.9 – Change user profile flowchart

1. Basic flow

Use-Case Start

This use case starts when the actor (doctor or patient) wants to change his/her profile data

.

1.1. The actor starts the aplication.
1.2. The system checks the internet connecton.
1.3. The actor selects *My Profile* operation.
1.4. The actor modifies user data.
1.5. The actor selects the save operation.
1.6. The system asks wheter the actor really wants to change the profile data.
1.7. The system saves the modified data in user profile.
1.8. The actor is redirected to the main page (activity).

Use-Case End

The actor ends the change profile data operation at 1.8.

2. Alternative Flows

At any time, System fails:
To support recovery and correct location tracking, ensure all transaction sensitive states and events can be recovered from any step of the scenario.

2.1. Actor reenters the system.
This flow can occur in one of the following steps: 1.1, 1.2
2.1.1. System detects anomalies preventing the device to communicate the changes with the server:
2.1.1.1. The system checks the (*internet*) premissions
2.1.1.2. The actor continues the operation.
2.1.2. Session timeout
2.1.2.1. System signals error, rejects entry and asks again for credentials.
2.1.3. The system does not get correct input data from the actor:
2.1.3.1. The system asks the actor to provide the data correctly
2.1.3.2. The actor continues the operation.
2.2. The actor selects the Back operation and leaves the change user profile page
This flow can occur in one of the following steps: 1.4
2.2.1. The system remains unchanged
2.3. The actor selects the Sign-out operation on purpose
This flow can occur in one of the following steps: 1.4
2.3.1. The system verifies that the actor initiated this operation on purpose
2.3.2. The actor is signed out
2.4. The actor selects the Sign-out operation accidentaly
2.4.1. The system verifies that the actor initiated this operation on purpose
2.4.2. The actor returns to change profile operation
2.5. The actor selects the Save operation accidentaly
This flow can occur in one of the following steps: 1.4, 1.5

2.5.1. The system verifies that the actor initiated this operation on purpose
2.5.2. The actor returns to change profile operation

3. Special Requirements

A mobile phone with Android OS 4.0.0 or newer.

4. Preconditions

- The actor is authenticated and authorized for this use case.
- A pinpoint location scenario is opened through a change profile operation.
- All necessary permissions are granted
- There is internet connection

5. Postconditions

- Scenario entities should remain in a consistent state if the track location has ended with success.

4.3. Technological perspective

As mentioned before the *Real-time Database* offered by Firebase is a database that stores data in a tree strucutre in which every node is a JSON object. (Figure 4.10 - Real-time Database structure).

Every node is saved under a unique ID, which are created automatically during a method call which is called *push*. Thes Ids created by using push are 20-character unique Ids. These IDs are used to handle situations when multiple users are accessing the database at the same time. This way they can write data at the same time without conflicts.

Figure 4.10 - Real-time Database structure

A push ID contains in itself 120 bits of information from which a part is a timestamp (48 bits) and a part is random (72 bits). To create the random ids that are visible in the Figure 4.10 - Real-time Database structure the ID formed from 120 bits is encoded into ASCII characters using a base64 alphabet. Hovewer even if the IDs are correct and unique som issues might appear. An issue can be with the order. Since the timestamp is created on the client machine there is a possibility that it's clock is not accurate. This issue is compensated in the moment after establishing a connection with the Firebase server in which te server sends a timestamp to correct the local clock (if that is incorrect). Another issue is the security. Are these push IDs really secure because if someone knows the root of the database and guesses correctly a push ID, the data in that node will be accessible. As Firebase states: „These IDs are very hard to guess, but if you do not trust the IDs generated by push, you can generate IDs using a more secure mechanism"[11].

Google Maps similarly to other applications that use GPS based localization, we can say that it is built on a principle named trilateration, since *the principle of operation of a GPS system is based on trilateration*[14] which is a simple mathematical principle.

There are two categories of trilateration: two dimension and three-dimension Trilateration. To calculate correclty the position the receiver device must know two important things. The first is the current location which is determined by 3 satellites

[14] www.eetimes.com/document.asp?doc_id=1278363

orbiting above the device. The second is the distance between the current location and the Space Vechicles from the Global Navigation Satellite System (GNSS).

The **Short Message Service**(SMS**)** used by the Patient Tracker application is based on the Global System for Mobule communications (GSM) in which messages are sent to a Short Message Service Center (SMSC) which has a store and forward mechanism. This mechanism tries to send the messages to the recipients and if the recepient is currently not available it queues the task for later retry. It also has a froward and forget option that tries to send the message only once. There are no guarantees that a mesage will be delivered but it is not common that some messages get lost.

Chapter 5. Detailed Design and Implementation

The following chapter will present a detailed description of the system considering the solutions used at implementation, the architecture and all the parts involved.

Further the modules will be presented from which the whole project is built up. We will discuss each and one of them separately in detail.

5.1. System architecture

In this subchapter the architecture of the proposed system will be presented in details.

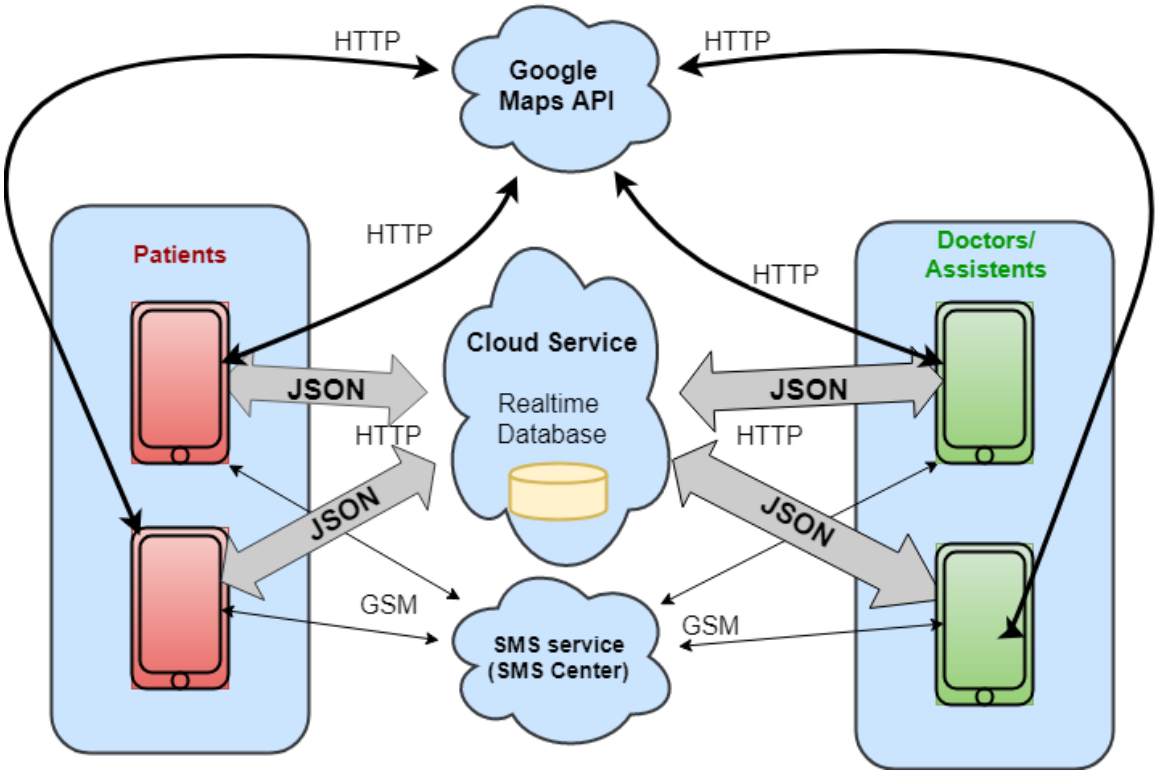

Figure 5.1 – Detailed conceptual architecture

The proposed system is based on a client-server architecture which using the advantages of a distributed system supports the Patient Tracker application which was designed to help the work of doctors in a hospital, general practitioners but also to help in giving more attention to the patients thus facilitating their recovery.

The components of the Patient Tracker system are listed and described in details below, organized in two groups: **clients** and **servers**.

- **Clients** are those mobile devices (smartphones that run on Android OS) which is used by doctors, nurses, general practitioners and their patients. Right at the beginning we should differentiate the clients by their type thus creating two main groups: workers of the healthcare system or **caregivers** and patients of the healthcare system or just simply **patients**. We will discuss in detail the roles, behavior and architecture of these two types of client later.

- **Servers** of the Patient Tracker systems as it can be seen in the
- Figure 5.1 – Detailed conceptual architecture are namely the SMS service, Firebase Real-time Database which is a cloud service and Google Maps. These servers accept requests from clients and most of the times they provide some information that was requested by the client, for example in the case of Firebase Real-time Database or Google Maps. In some other cases the client sends or shares some information with the servers because it is necessary to have some centralized and synchronized data on the servers (or at least on one of the servers) in order to connect the client in question with the other clients. We will discuss in detail the roles of each server, their behavior and architecture later.

As mentioned above the components of the system share information. Sharing information means that the servers offers the client some useful information who will be able to use this, while the client stores some data in the database for later use. While sharing information the servers do not expose information such as their software specification or the state of the hardware on which they run. Instead they provide a transparent interface for the client. This interfacing is necessary for the sake of simplicity, transparency and simplicity and it is achieved with the help of some Application Programming Interfaces (APIs) through which the client and server exchange information. This information between the client and Real-time Database is represented in JSON files.

Because the server(s) provide a lot of useful information to the client, all this with high precision and small latency we can say that the servers have a large amount of computing power. The servers and client exchange data which is in most of the cases is stored on the servers and displayed by the client.

5.1.1. System components

The **Client** application which the main part of the project, obviously is used by clients, more precisely doctors and patients. Since there are two types of users the client application is differentiated accordingly to a "*doctor part*" and a "*patient part*". This design decision was made for the sake of simplicity, because this application is a simple one with not a huge workload, moreover the maintenance of a single application is easier than of two separate applications, also if a software update is coming the compatibility issues will not be a problem. At last but not least the clients will have to download and use one single application, which comes handy especially in cases when the doctors do

not have to worry about the patients downloading and using another (wrong) application or using the right application but with the wrong version.

The **Doctor part** of the client application appears, when someone registers in the application as a doctor. After this he or she will be stored in the database as a doctor and whenever logs in the doctor part of the application will appear for him/her.

As a doctor a user can do the following things with the application:

- Log in
- Use the **forgot my password** possibility
- View the list of his or her patients, which means that the doctor can see as a preview the following information per patient:
    - The patient's full name (first name and last name)
    - The patient's CNP and age
    - The medical signs that were registered and noted by the patient on the patient part of the application.
- View a patient's profile that contains all the necessary information that helps in diagnosing or contacting the patient as fast as possible, which includes:
    - Data related to diagnosis such as: medical signs noted by the patient, age, occupation
    - Data related to contacting the patient like: phone number, address, e-mail address, CNP
- Add some information to the patient's profile information, in order to notify the patient about important news, which can be two specific things:
    - A diagnosis for the patient and a remainder if a diagnosis has a name that is hard to memorize
    - A treatment for the patient to explain is short to him/her what to do for a faster healing or how to take the medications.
- View the patient's location on a map and also see the exact address where the patient is or was when last time used the application.
- Send an SMS message to the patient.
- Add patients by their CNP
- Remove patients from the list
- View and edit the profile data in the user profile menu, where the following attributes can be changed: name, e-mail address, phone number, postal address, password, the doctor's title, specialization or hospital where he or she works.
- Logout

The **Patient part** of the application appears when someone registers in the application as a patient. After this he or she will be stored in the database as a patient and whenever logs in the patient part of the application will appear for him/her. Also, the application will automatically send location data to the database in order to doctors be able to pinpoint a patient's location.

As a patient a user can do the following things with the application:

- Log in
- Use the **forgot my password** possibility
- View the list of his or her doctors or caregivers, which means that the patient can see as a preview the following information per caregiver:
    - The caregiver's full name (first name and last name)
    - The hospital or place where the caregiver works
    - The specialization and title of the caregiver.
- View the diagnosis written by the doctor
- View the treatment methodology mentioned by the doctor
- Send an SMS message to the doctor
- Call the doctor
- Make an emergency call that will notify the ambulance by calling them and by SMS, and it also will notify by SMS the doctors.
- View and edit the profile data in the user profile menu where the following attributes can be changed: CNP, name, e-mail address, phone number, postal address, password, medical signs and occupation.

On the **server** side of the Patient Tracker application some components did not needed to be implemented from scratch (like the real-time database mechanisms or the localization mechanisms) because Google and Firebase provides an already implemented and working solution for both of them, moreover with the different *Application Programming Interface*s (APIs) and *Software Development Kit*s (SDKs) every developer can succeed with ease. Even for using the *Short Message Service* (SMS) the SDK provided by Android can be used.

The application communicates with these servers for the following reasons:

- Store, change, retrieve data, for example when a registration happens in the application the new user and its data will be stored in the Firebase Real-time Database all this is achieved by communicating it through the API provided by Firebase.
- Use a service that can be found on a particular server, for example to draw the map, zoom in on that map and pinpoint a location which can be found on the Google Maps server; or forward a text message towards someone which can be achieved through the Short Message Service Center (SMSC) that navigates the message through the network.

5.2. Application modules

Modules are standalone parts from the application that are complete separately work as little applications. In the build process of the project these modules were implemented first, later tested and finally integrated in the project with the necessary modifications of course, in order to these modules do not have compatibility or communication issues between them.

During the integration process these modules did not keep their separate module form, instead since the were small enough, their classes were moved to the main project (or module).

5.2.1. Patient Tracker

The patient tracker module is one of the biggest modules which has one single purpose, which is letting know the doctors where their patients are. This goal is achieved by the cooperation of more components, namely: doctor and patient parts of the application, the real-time database and localization via passive location providers or GPS and network providers (cell tower and WIFI).

Let us observe below in Figure 5.2 – Patient Tracking module, conceptual diagram the conceptual architecture of this module:

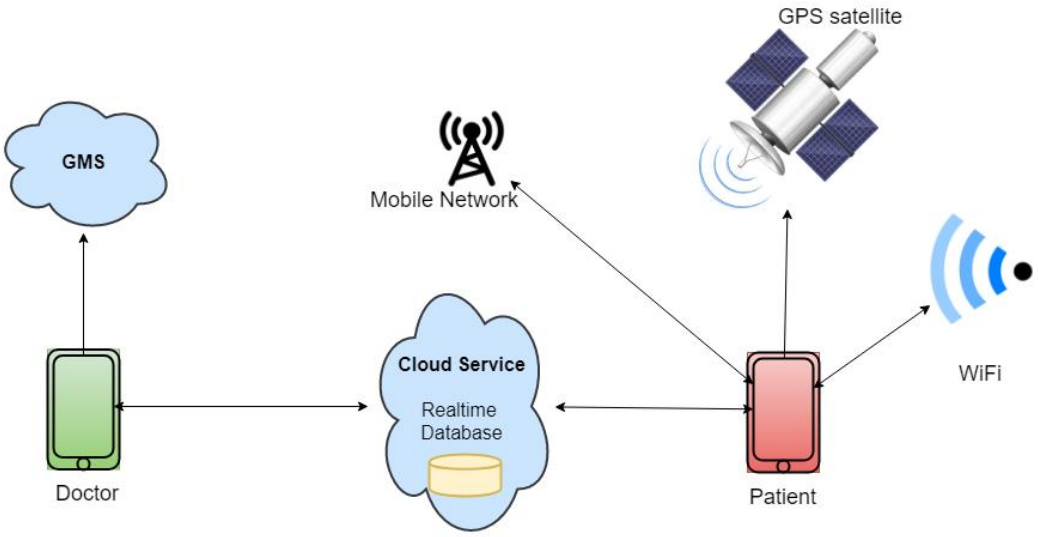

Figure 5.2 – Patient Tracking module, conceptual diagram

The clients are connected via the Cloud Service, which assumes there is internet connection. For this module to work it is not enough to have internet connection but it also requires the location to be enabled and be in the ON state. However this precondition is only true for the *Patient* because the patient will use localization via GPS or network, to determine its own current location.

As soon as a patient enters the application and provides the necessary permissions (*mobile data* and *GPS*), the application will request in the background a location from all the providers available, which includes GPS, cell towers and WiFi. If there is a result and that result differs from the last known location the application uploads the new location data of the patient to the Real-time Database. Since using GPS and mobile network at the same time is resource demanding because it highly increases the battery usage the application will not request new location data all the time, but periodically which can be set to half a minute, a minute or any desired time interval.

When a doctor logs in, the application in the backgorund will automatically listen to changes in the real-time database, more specifically to key – value pairs that store the coordinates of the patients. If the doctor is looking at a specific patient's position as soon as the coordinates were modified the marker (that shows the patient's exact position on the map) will be moved to the new position, which normally should be a minor visible change on the map as long as the patient does not travel at very high speeds.

5.2.2. Emergency Call

The Emergency Call module is one of the most important modules in the projects since it is designed to call for help in situations where even a life might depend on it. This functionality is accessible only for patients because in these kinds of environments where this project will be used they are more vulnerable and it is justifiable for them to have a measure to call for help.

```
DoubleClickListener doubleClickListener = new DoubleClickListener() {
    @Override
    public void onDoubleClick(View view) {
        dialContactPhone(presenter.getEmergencyPhoneNr());
    }
};

@Override
public boolean dialContactPhone(String phoneNumber) {
    Intent callIntent = new Intent(Intent.ACTION_CALL);
    callIntent.setData(Uri.fromParts( scheme: "tel", phoneNumber,  fragment: null));
    if(ActivityCompat.checkSelfPermission( context: this, Manifest.permission.CALL_PHONE)
            != PackageManager.PERMISSION_GRANTED){
        return false;
    }
    startActivity(callIntent);
    return true;
}
```

Figure 5.3 – Code snippet for initiating an emergency call

The patient does not have to search for a phone numbers or write it by hand instead he/she only has to press a big red button to call immediately for help. As we can see in the Figure 5.3 – Code snippet for initiating an emergency callonly a few lines of code is needed to start the call. This call will open the default dialer of the smartphone and call the given number right away.

To avoid calling 112 when the button is accidentally pressed a listener was implemented that will only enable to initiate an emergency call if the button was pressed two times consecutively in a certain period of time, for example in half of a second.

If there is no call permission granted by the user unfortunately the application will not be able to make an emergency call, instead it will ask the user to grant this permission.

5.2.3. Messaging

The messaging module is a helper module in the project because it offers the users the possibility to contact each other via SMS. This functionality is available both doctors and patients which means doctors can text their patients and vice versa. This might be helpful especially for doctors because if they have something important to announce they can just look up a patient from the application and text him right away without the need to search among phone numbers, notes or datasheets. This also applies to the patients even if they don’t have to pay attention to many doctors.

The messaging module helps this communication with hints like in Figure 5.4 – Send SMS doctor and patient view and writing an automatic introduction before the message and a note after the message based on the user profiles found in the database such as: "*Hello Pop Andrei! I am Dr. Iancu from the hospital*" after which comes the message composed by the user and finally "*Sent from PatTrack application.*"

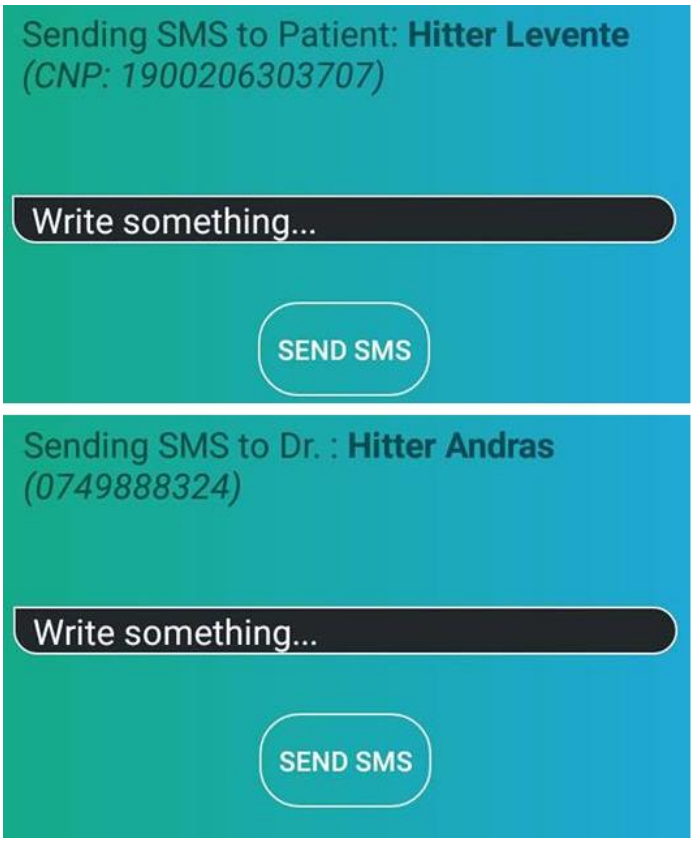

Figure 5.4 – Send SMS doctor and patient view

5.2.4. Typing by voice

This module is a helper module for the previous Messaging. Basically, it offers a possibility to write the text message by voice input. It might prove useful when both hands are full or when a patient is bedded and cannot handle easily such a task.

5.3. Design patterns used in the project

5.3.1. Model View Presenter pattern

As the name suggest this design pattern is based on three components which are the following:

- Model
- Presenter
- View

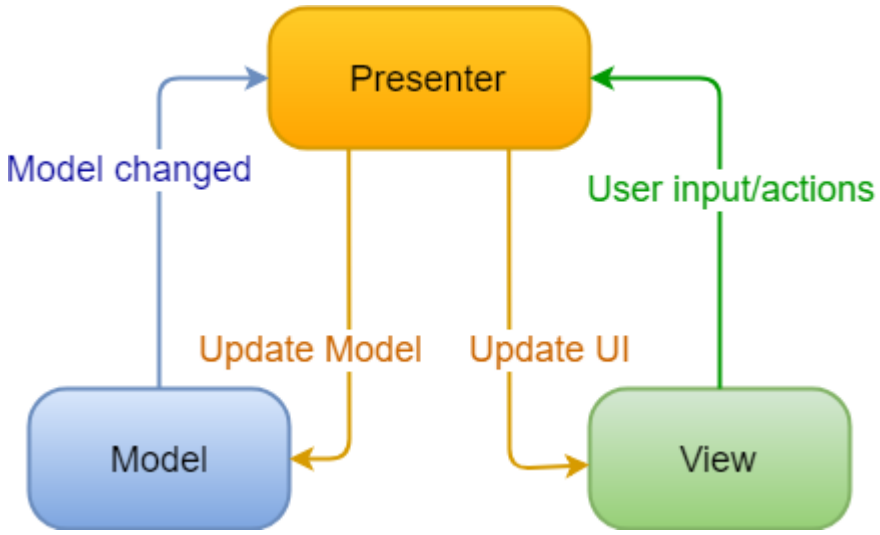

Figure 5.5 Model-View-Presenter design pattern

As the Figure 5.5 Model-View-Presenter design pattern presents each and one of the three components have a well defined role. This modularization first of all help to organize the code especially for large projects, makes the project more understandable and organized and it makes the project easily expansible and testable.

The **Model** is responsible to manage the data which in our case is in the Firebase Database and the the *Presenter* communicates with via the API provided by Firebase. It stores all the data which will be used by the application[1].

The **Presenter** forms a middle layer between *model* and *view*. All the presentation logic belongs here, which includes querying the data from model or react to user inputs by updating the view with the required data. It decides what will happen if there was an interaction with the view. The view registered the user action but what to do with it is implemented in the Presenter. After deciding the appropriate action to a user input returns the formatted data to the *View*[1].

The **View** has one single purpose which is displaying the data in a way that was specified by the *presenter* which means that the only thing it will do is to call a presenter method whenever it registers a user action. Usually the *Activity* classes represent the view from this design pattern[1].

5.3.2. Factory method pattern

Using the **Factory method pattern**, the project will be able to create objects without being specified in advance what is the class of the object that will be created[12]. This mean that if it can be known about an object that which class it will belong to, by calling the factory method the problem is solved, because this method will decide and create the appropriate class for the object in question.

In this project the factory method is used at registration since it cannot be know in advance who will register next: a doctor or a patient? By analyzing the user inputs the factory method will decide what class is going to be created for the registering user which can be either *DoctorUser* or *PatientUser*.

5.3.3. Adapter pattern

This design pattern is a structural design pattern and it joins two unrelated interfaces[12].

In my project there are two adapter classes: *DoctorUserListAdapter* and *PatientUserListAdapter*. These adapters have the purpose of making it possible to use customized and complex design elements in a normal *ListView* that usually is used for displaying lists in Android. These design elements contain the list of patients for doctors and list of doctors for patients respectively.

5.4. Architecture of the Android application

In this subchapter is described the architecture of the mobile application that was developed in Android Studio *Integrated Development Environment* (IDE) using mainly the Java programming language. This environment uses the SDK provided by Android and it also offers the possibility of integrating and using APIs like the ones provided by Firebase and Google Maps.

The *Graphical User Interface* (GUI) was built in Android Studio's layout editor which uses XML language. The layout editor has a powerful visual design editor that based on the position and size of the movable UI elements generates XML code thus helping the developer. In this project the layout files were written by hand with the help of IDE's auto complete features and they were verified with visual feedback provided by the visual design editor.

The project (as we can see in the Figure 5.6 – Project's modules) is constructed from two modules which are the following:

- Arch: is the skeleton of the project. It has abstract classes and interfaces and it defines the whole projects architecture. The whole project follows its pattern which is presented in more detail above (Model View Presenter pattern).
- App: is the working part of the application, the "meat on the skeleton". This module contains all the Activities, helper classes and user interface elements.

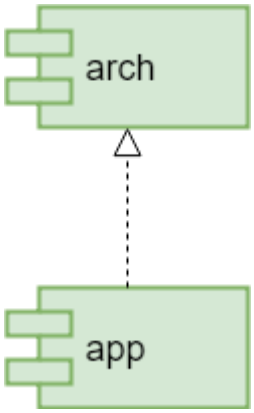

Figure 5.6 – Project's modules

5.4.1. Arch module

This module contains the following classes which are illustrated in the Figure 5.7 – Arch module, class diagrambelow:

- **BaseContract**: is an interface in order to make the the differentiation between *presenter* and *view* accordingly to the MVP design pattern and precisely defin the actions that each one of them can individually do. It has two inner classes:
  - *Interface View:*
  - *Interface Presenter*
- **BaseActivity**: has two generic types (*View* and *Presenter*) and couples them by creating a *BaseViewModel.*
- **BasePresenter**: has a single generic type: *View*. It contains different methods for attaching, detaching, getting the view and verifying if there is a view attached.
- **BaseViewModel**: is a ViewModel that contains two generic types: a *presenter* and a *view.* The *view* extends *BaseContract.View*, while the presenter *BaseContract.Presenter* (which has a *view* and this view is a part of the presenter).
  Basically it creates an architecture where we can attach and detach *views* (activities or fragments) to the presenter. With this arcitecure we can save all the important data (temporarily) in the presenter which will not be destroyed when the View or Activity gets destroyed. Instead we can attach a new one to the presenter and load the necessary data in it.
- **BaseFragment**: is is similar to the *BaseActivity* but instead of *Activity* it extends the *Fragment* interface which has some different characteristics.

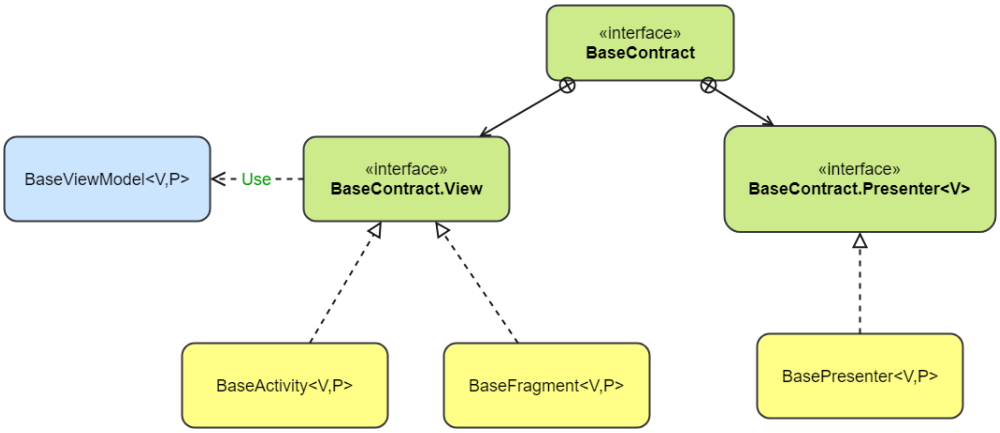

Figure 5.7 – Arch module, class diagram

5.4.2. App module

This module contains the following groups of classes which are illustreted in the below:

- **Activities**: they represent the *View* from Model View Presenter pattern and it usually contains a reference to the *presenter* provided by dependency injectors. In this project there is an interface which defines the relation between the *presenter* and *view*. This means that *view* and the *presenter* are in a ViewModel. In their relation a *presenter has-a* view. The activities are: LoginActiviy, SignupActivity, ResetPasswordActivity, DoctorMainActivity, PatientMainActivity, DoctorProfileActivity, PatientProfileActivity, MapsActivity and SmsActivity, PatientDatasheedActivity.
- **Contracts**: they are interfaces which define what the activities(vies) and presenters will do, and connects these two components. The contracts are: LoginContract, SignupContract, ResetPasswordContract, DoctorMainContract, PatientMainContract, DoctorProfileContract, PatientProfileContract, PatientDatasheetContract.
- **Presenters**: they represent the *Presenter* from the MVP pattern. These are: LoginPresenter, SignupPresenter, ResetPasswordPresenter, DoctorMainPresenter, PatientMainPresenter, DoctorProfilePresenter, PatientProfilePresenter, PatientDatasheetPresenter.
- **Validators**: are used to validate user input. In order to avoid rewriting the same code many times an interface was used (Figure 5.8 – Validator class diagram) and in every class that implements this (Validator) interface changed the condition against which the user input vas validated. Some of these are: PasswordValidator, NameValidator, EmailValidator, PhoneNrValidator.
- **Models**: these represent the real world doctors and patients ans objects. There is a general *User* class which holds the common attributes of the doctors and patients. *PatientUser* and *DoctorUser* are extending the User class and each adds its „special" attributes that are different for each.
- **Layout files**: they part of the MVP pattern, because they display all the data and through them collects the system the user inputs.

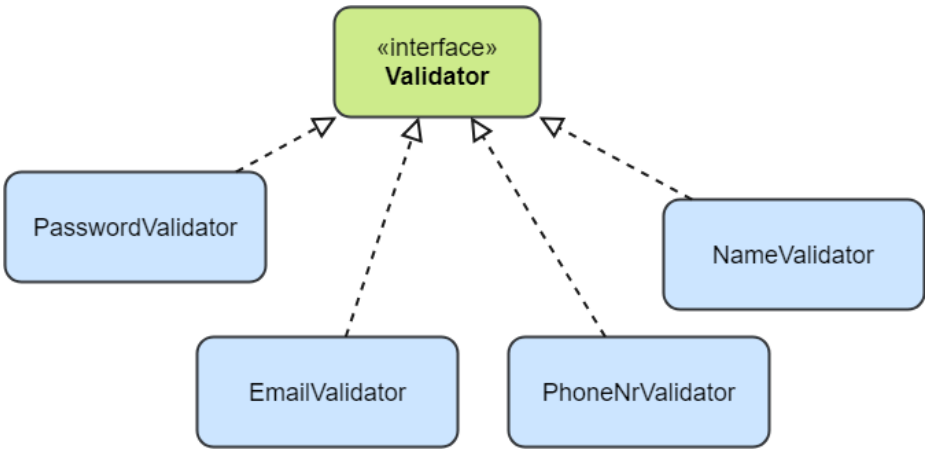

Figure 5.8 – Validator class diagram

As mentioned before the application is build on a skeleton defined in the *arch* module. This means that every class from the *app* module extends or implements the corresponding interface or abstract class from the *arch* module(see Figure 5.11 – Application package diagram). Since there are many relations an illustration will follow that will present the relation of classes in the Login part of the application (Figure 5.9 – Partial class diagram).

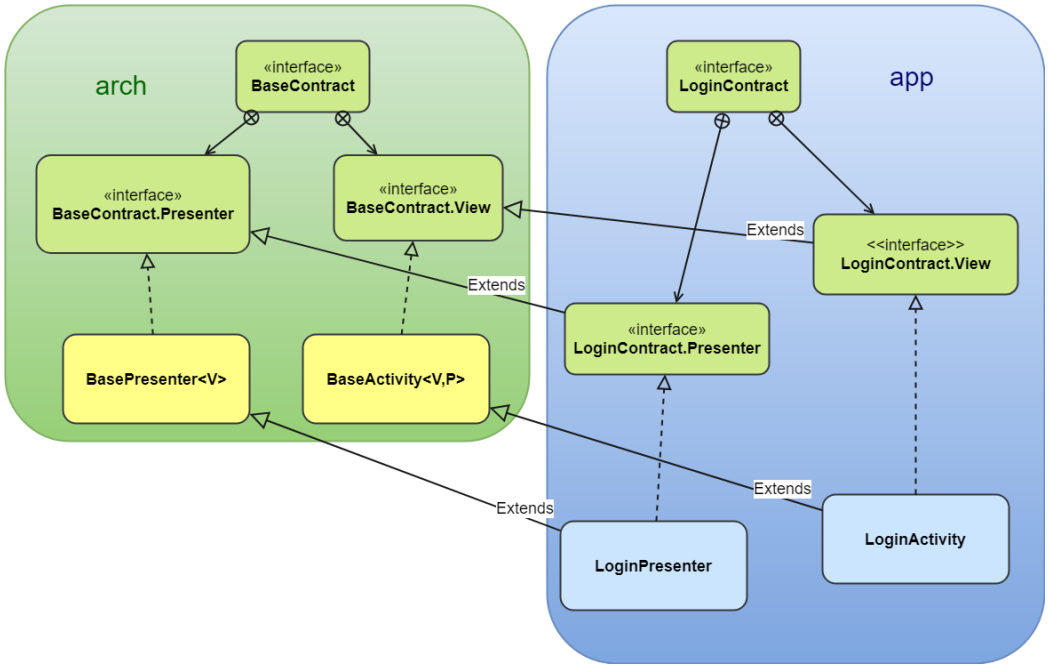

Figure 5.9 – Partial class diagram (Login)

When as shown in the figure when the Login part of the project is created a *presenter* (LoginPresenter) is created, initiated and it also gets a *view* (LoginActivity) attached. After all this the LoginPresenter and LoginActivity can communicate and use each other's methods. An example can be seen in the Figure 5.12 – Presenter and Activity cooperation.

The whole project is built on this model thus the diagram would be the exact same for the other activities except for the SmsActivity and the MapsActivity because the get the required data through the intents (Figure 5.10 – put and get information from intent) when they are created. They get the information either from *DoctorMainActivity* or *PatientMainActivity* the MVP pattern still stand because the SMS and Maps activities function as views with the difference that they get the information from *DoctorMainPresenter* or *PatientMainPresenter*.

```
@Override
public boolean showPatientLocation(int patientPos) {
    PatientUser patient = presenter.getPatientList().get(patientPos);
    String doctorId = presenter.getDoctorUser().getId();
    Intent intent = new Intent( packageContext: DoctorMainActivity.this, MapsActivity.class);
    intent.putExtra( name: "id", patient.getId()).putExtra( name: "doctorId",doctorId);
    startActivity(intent);
    return true;
}
```

-inside DoctorMainActivity we put the necessary information

```
this.patientId = getIntent().getExtras().getString( key: "id");
this.doctorId = getIntent().getExtras().getString( key: "doctorId");
```

-inside MapsActivity we get the information

Figure 5.10 – put and get information from intent

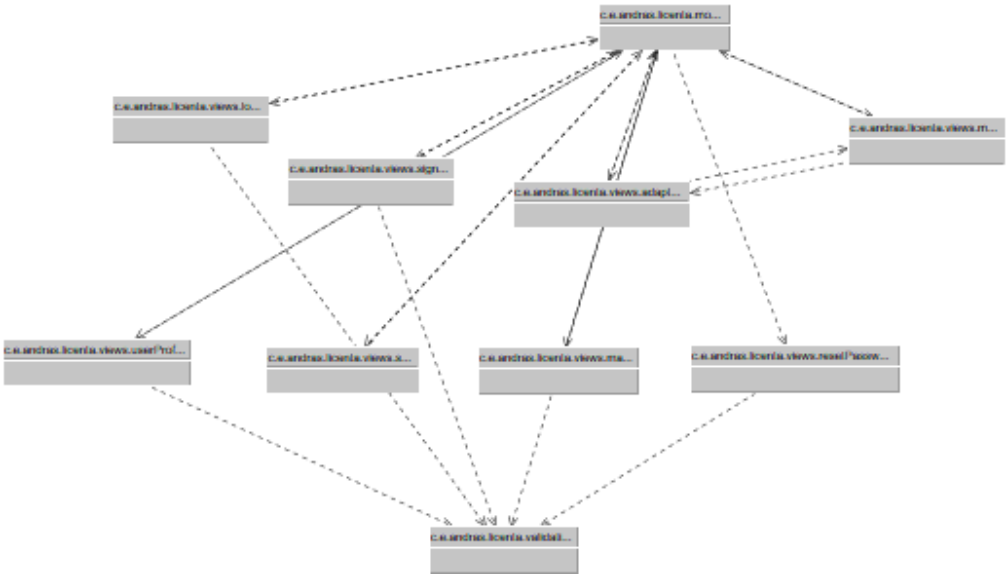

Figure 5.11 – Application package diagram

```
public View.OnClickListener onLoginClick = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //showRequestInProgress();
        presenter.onLoginClick(editTextEmail.getText().toString(), editTextPassword.getText().toString());
    }
};

public View.OnClickListener onSignupClick = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        presenter.onSignupClick();
    }
};

public View.OnClickListener onResetPassClick = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        presenter.onForgottenPassClick();
    }
};
```

In LoginActivity

```
@Override
public void onLoginClick(String email, String password) {
    if (!isViewAttached())
        return;

    if (validateEmail(email) & validatePassword(password)) {
        getView().correctLoginFormat(email,password);
    }
}

@Override
public void onSignupClick() {
    if(!isViewAttached())
        return;
    getView().signup();
}

@Override
public void onForgottenPassClick() {
    if(!isViewAttached())
        return;
    getView().forgottenPassword();
}

@Override
public boolean validateEmail(String email) {
    if (!isViewAttached())
        return false;

    EmailValidator emailValidator = new EmailValidator();
    Validator.ERROR_TYPE emailErrorType = emailValidator.validate(email);
    if (emailErrorType != CORRECT) {
        getView().setEmailError(true);
        return false;
    } else {
        getView().setEmailError(false);
```

In LoginPresenter

Figure 5.12 – Presenter and Activity cooperation

Chapter 6. Testing and Validation

This chapter will describe the testing methods used during the project development and the obtained results as well.

After every development phase it is required a testing phase during which the developer tries to find unwanted behavior in the software. Besides testing the new module, the compatibility and cooperation is also tested between the "old" components and the new one. All this happens before accepting an update in the software.

The project was tested manually following the test cases which are based on the system's functionalities, but a tool named *Test Lab* offered by Firebase was tried as an experiment.

Test Lab might be a useful tool during the development phase because with this tool the developer can test the product on a wide range of devices which would be impossible otherwise, because even the developer cannot possess nor tens or hundreds of devices to test the application, nor the necessary time to do the testing. This tool has been tried but the main testing phases were done manually on 2-3 devices.

6.1. Test cases

In this subchapter the main test cases of the Patient Tracker system will be enumerated.

6.1.1. Logging the user into the system

Table 6.1 – Test case – user login

	Phase	Result
1	Opent the Patient Tracker application	The application displays the LoginActivity.
2	Enter credentials in the login fields (email and password)	The text entered by the user will appear in the login fields
3	Validate the user input	The user will see error messages regarding the format of the input (valid/invalid email)
4	Confirm credentials (press Login)	The user gets a response from the server according to the correctnes of the credentials

6.1.2. Register the user into the system

Table 6.2 Test case – user register

	Phase	Result
1	Opent the Patient Tracker application	The application displays the LoginActivity.
2	Press *Register Now* and select the checkbox (only for doctors)	The application displays the SignupActivity and changes the icon according to the checkbox's state.
3	Enter the required information in the text fields.	The text entered by the user will appear in the fields.
4	Validate the user input	The user will see error messages regarding the format of the input (valid/invalid email, password, phone number, etc.)
5	Confirm signup (press Register)	The user gets a response from the server according to the correctnes of the credentials

6.1.3. Add new patient

Table 6.3 – Test case – add new patient

	Phase	Result
1	Opent the Patient Tracker application (as a docotr)	The application displays the DoctorMainActivity.
2	Press *My profile*	The application displays the *DoctorProfileActivity* and the profile data.
3	Change *Add Patient* switch position.	The system enables user input at add patient field
4	Enter CNP	The text entered by the user will appear in the fields and the user will see an error message if the CNP has a wrong format.
5	Confirm add(press ADD)	The user gets a response from the application according success of the operation
6	See results	The new patient appears on the main page

6.1.4. Send message

Table 6.4 Test case – send message

	Phase	Result
1	Opent the Patient Tracker application	The application displays the DoctorMainActivity or PatientMainActivity.
2	Select someone from the user list and press the message icon on the left.	The application displays the *SmsActivity* receiver's name and phone number.
3	Enter a message	The system displays the entered text
4	Sending the message	The system sends the message.
5	Receiving the message	The receiver user gets the message completed automatically with an introduction of the sender.

6.1.5. Emergency call

Table 6.5 - Test case – Emergency call

	Phase	Result
1	Opent the Patient Tracker application (as a patient)	The application displays the PatientMainActivity.
2	Double tap on the Emergency Call	The application calls 112.
3		The system sends messages to 112 and to the doctor with the patient current location.

Chapter 7. Installation and User manual

In this chapter will outline the main steps that are required to install and run the Patient Tracker application. Later is described a user guide that will walk the user through the user step by step with illustrations.

The application have some system and software requirements that the users' device has to meet in order to be able to use the app.

7.1. System requirements

The patient tracker application can be installed on any smart device with Android OS, at least 10 MB of free storage and 50 MB RAM memory. Another requirement is to have a stable internet connection, because otherwise the application cannot communicate with the servers (Google Maps and Firebase) and the user will not be able to login, register, or have a look at the list of contacts.

The application works with GPS coordinates, which requires the location to be turned on or it will not be able to provide patient location data towards the doctors. The application also needs a working SIM card in order to be able to call or send SMS.

7.2. Software requirements

The Patient Tracker application requires an Android OS which has an API Level equal or higher than 23 (Android M - Marshmallow), which is the recommended API Level for the application.

7.3. Installation

Thanks to the always developing Google Play there are more ways to install the Patient Tracker application on the user devices.

The first methodology is to add the device to the list of devices in Google Play and search for the application from a desktop computer. When the application is found one can give the command to install the application to the mobile device as soon as it is connected to the internet. This methodology is advised if the user is in the possession of multiple mobile devices and wants the application to be installed on all of them. This way it avoids the inconvenience to install the application separately on each one of them.

The second methodology is to enter Google Play Store (which is preinstalled on all of the Android devices) from the mobile device by tapping on the icon and wait for the application to start up.

In the main menu at the top of the page the user can see a hint that says "Google Play"(Figure 7.1 – Google Play Search). By tapping on that text, the text will transform into "Search Google Play" which indicates that the user is in search mode and the

application waits for the user to input something to search for. There the user will enter the application name which is “Patient Tracker”.

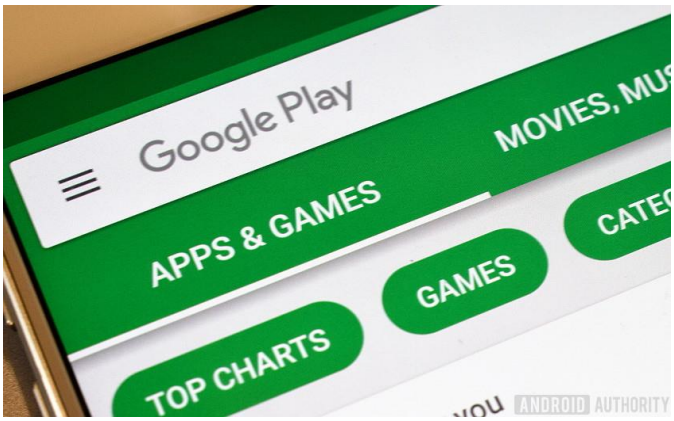

Figure 7.1 – Google Play Search

According to the quality of the Internet connection, Google Play will display the found matches for „Patient Tracker”. Every one of the results can be inspected by tapping on the which redirects the user to the application’s page to show all the information about that particular application.

By pressing the button „Install” the download will start and the user will se a progress dialog which provides visual feedback about he staus of the installation in percents. When the installation completes a notification will apear that the application is installed and is ready to use. A bubble window might appear at the bottom of the screen that says „Application is ready” and gives two possibilities for a few seconds: „Cancel” and „Open”.

There is a third method to install the application, which is installation by the .apk file. This type of installation requires the user to download the *apk* file on the device or first to the computer and after that copy to the device. When all this is done the user have to search for the saved apk file on the mobile device. When the file is found the user must start the install manualy by taping on it and answering to the questions that appear or pres the file for longer and select from the menu the install option and follow the instructions.

7.4. Using the application

7.4.1. Opening the application

After the installation process the user opens the Patient Tracker application by tapping on its icon. The icon is easy to recognize, it simbolizes a doctor and a patient standig next to each other. The figures are white on a blue backround.

7.4.2. The first use of the application

The user will see a short animation as the icon and the login fields appear and move to their final place. Right after the animation the user is asked to give the application the permission to access the device's location as it is presented in Figure 7.2 – First login. It is recommended to respond with „Allow" .

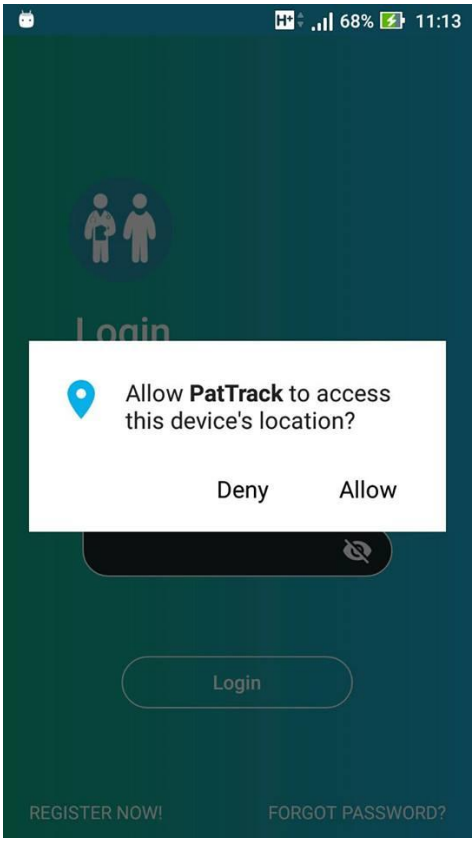

Figure 7.2 – First login

7.5. Login, Signup and Recover Password

By pressing the "Register Now!" text the user will be redirected to register page which will also greet the user with an animation. On the signup page the user is asked to enter the necessary information to create a new account.

The first information is given by checking or unchecking a checkbox which says "Doctor?". If the user is a doctor he or she must check this checkbox. If the icon is changed to a doctor then the box was checked successfully and the user will be registered as a doctor otherwise will be registered as a patient.

Doctors must provide information about their: name, e-mail, phone number, job title, specialization, hospital where the work and a password along with a confirmation password to make sure there was no mistake when entering the password.

Patients must provide information similarly about their: name, e-mail, phone number, occupation, CNP, medical signs (what they observed on themselves, where it hurts?) and a password along with a confirmation password to make sure there was no mistake when entering the password.

For every information there is a validation and a real-time feedback to let the user know if the entered information is acceptable or not like in Figure 7.4 – Incorrect register input and Figure 7.3 – Correct register input.

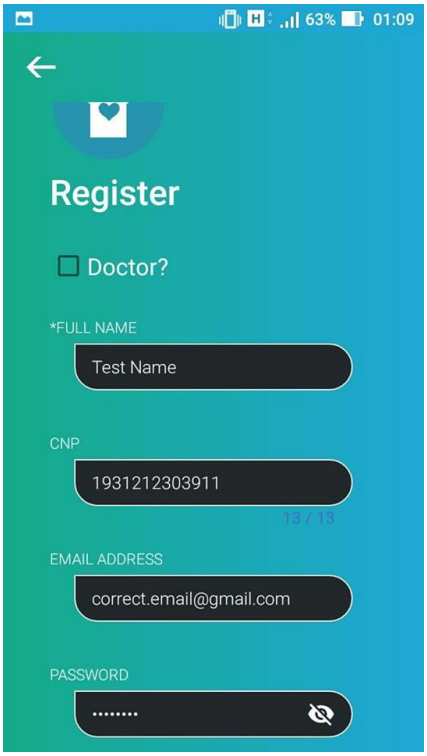

Figure 7.3 – Correct register input

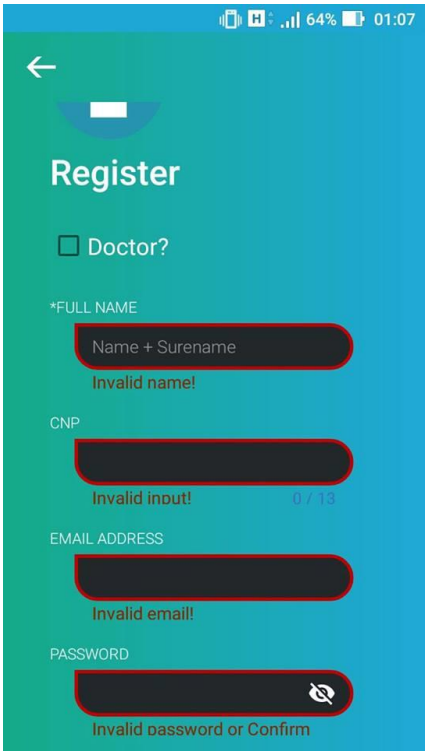

Figure 7.4 – Incorrect register input

After the signup process the user is redirected to the main page that corresponds to the user type.

If the user already has an account it can login on the login page by entering the credentials (email and password) and pressing the login button as presented in the Figure 7.5 - LoginPage and Figure 7.6 – Logged in below.

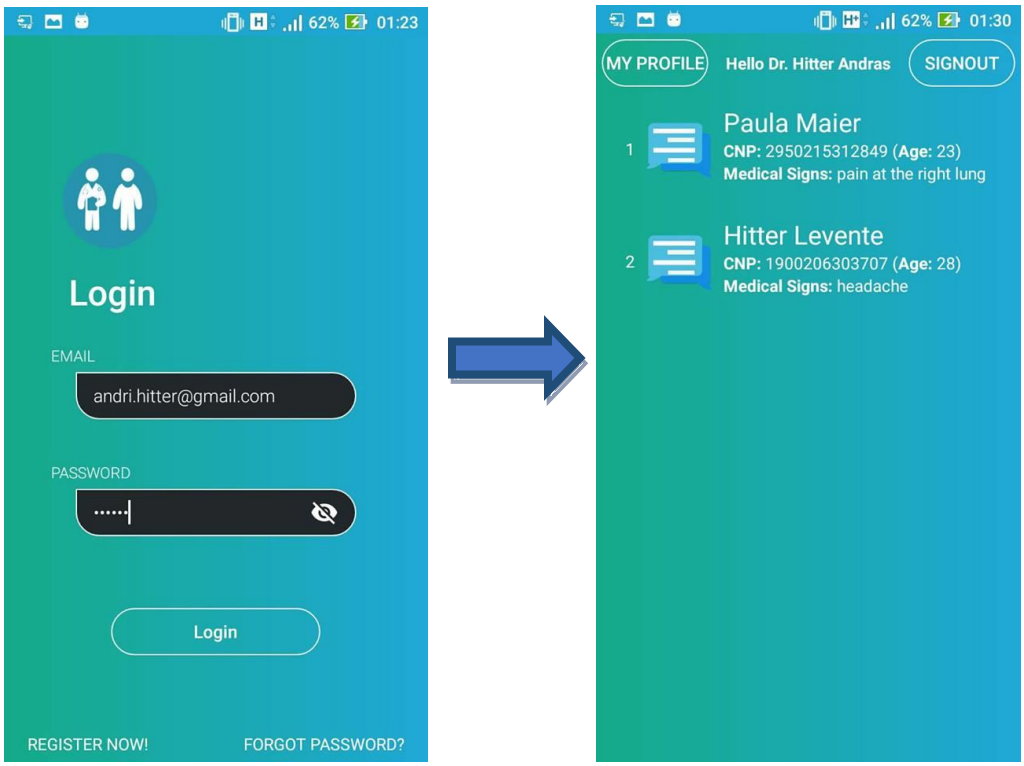

Figure 7.5 - LoginPage and Figure 7.6 – Logged in

In the unfortunate scenario in which the user forot the password he/she can tap on the „Forgot Password?" text in the righ bottom corner of the screen and will be redirected to a page where the password can be recovered by providing the e-mail address. Here as we can see in Figure 7.8 Forgot Password1 and Figure 7.7 – Forgot Password2 the system asks for an email address and it will send to that address a link. By accessing that link the patient will be redirected to an external page (which practically is a page provided by the Firebase server) where the user can provide a new password.

The application has a circular structure which means that the user can navigate back to the login page from anywhere and from the login page can navigate to anywhere but in some cases with the help of intermediate pages.

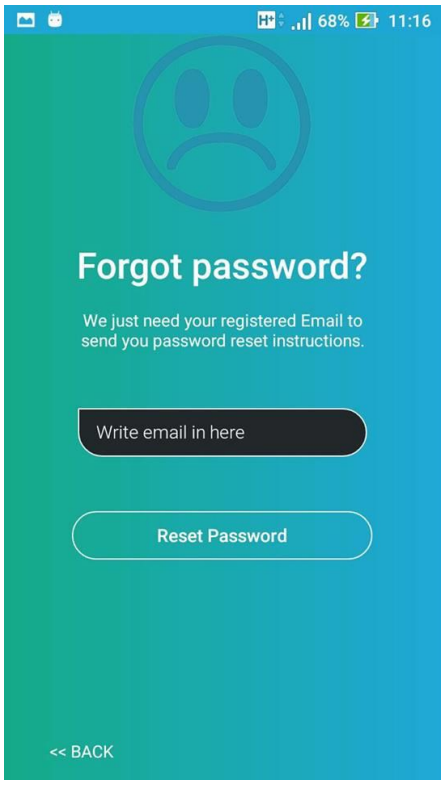

Figure 7.8 Forgot Password1

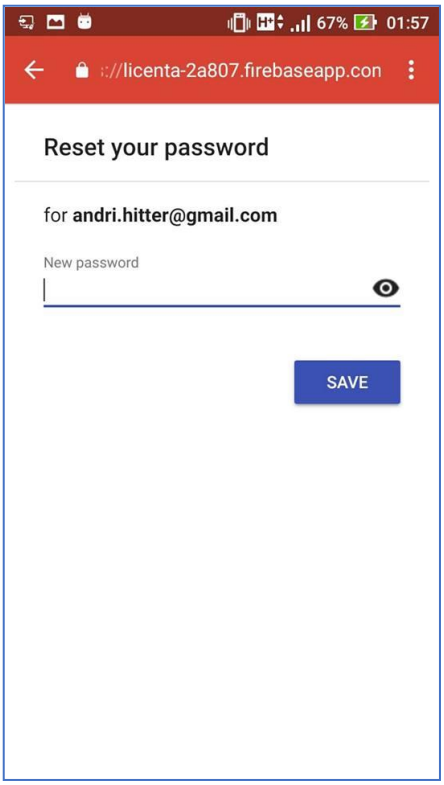

Figure 7.7 – Forgot Password2

When the user successfully logs in there are a set of actions that he/she can do based on which user type he/she belongs to:

- Doctor:
  - by pressing the chat button he can write an SMS
  - by swiping left on a patient and tapping on the GPS icon he can see on a map -with address- where is that particular patient
  - by swiping left on a patient and tapping on the profile icon he can see the patient profile and edit the diagnosis and treatment fields
  - by swiping left on a patient and tapping consecutively twice on a patient the doctor can remove the patient from the list
  - by tapping on logout the doctor can log out and goes to the login screen
  - by tapping on my profile the doctor can see his own profile informations and edit them. Here he can also add new patients to his list.
  - By tapping on save in MyProfile the doctor saves the changes
  - By tapping on logout in MyProfile the doctor signs out

Patient: (

- Figure 7.9 Patient user guide)
  - by pressing the chat button he can write an SMS to a doctor
  - by swiping left on a doctor and tapping on the CALL icon he can see dial up the doctor’s number immediately

- by tapping consecutively twice on the EMERGENCY CALL the 112 will be called, and the location of the patient will be sent to the doctors and 112
- by tapping on logout the patient can log out and goes to the login screen
- by tapping on my profile the doctor can see his own profile informations and edit them. Here he can also change the CNP
- By tapping on save in MyProfile the patient saves the changes
- By tuapping on logout in MyProfile the patient signs out

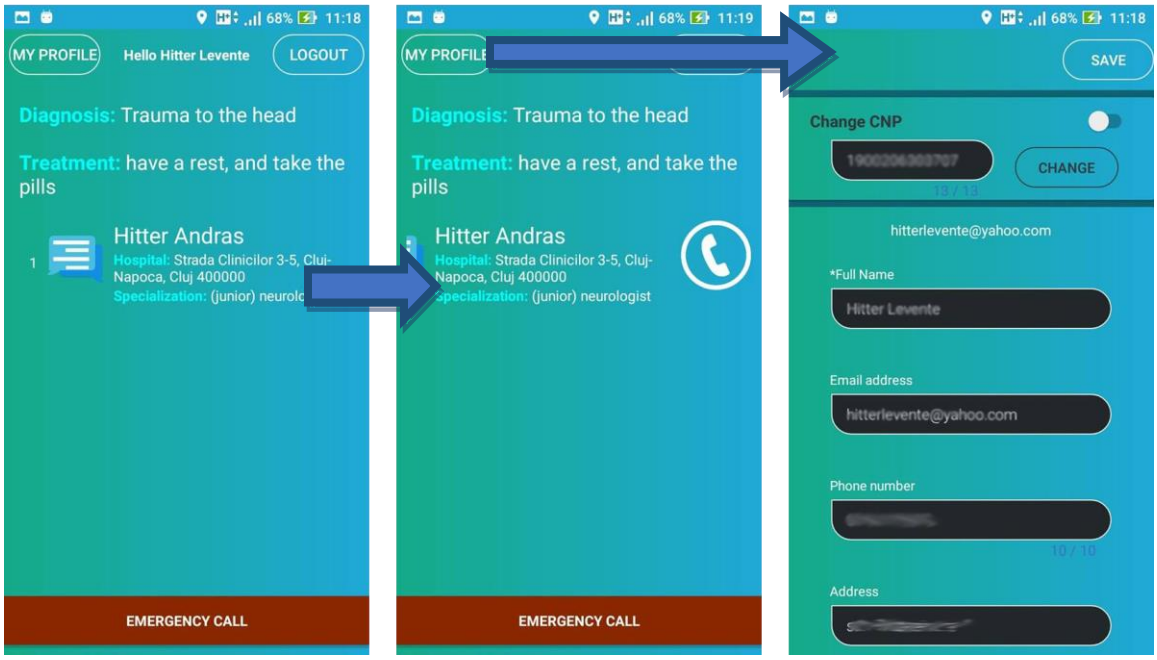

Figure 7.9 Patient user guide

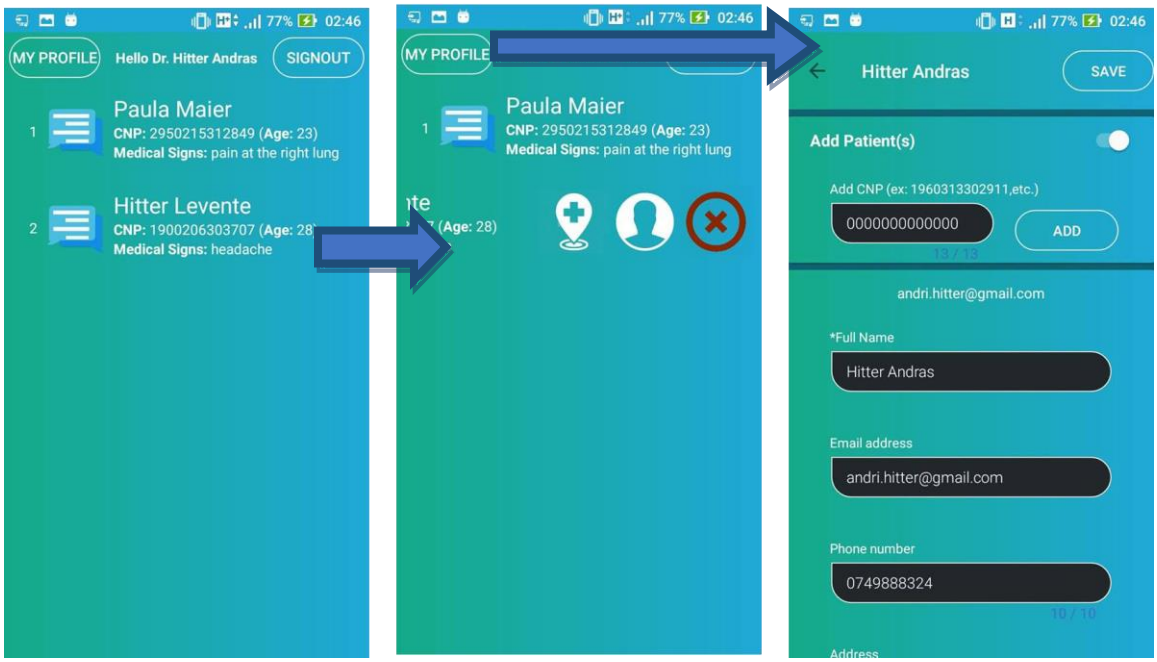

Figure 7.10 Doctor user guide1

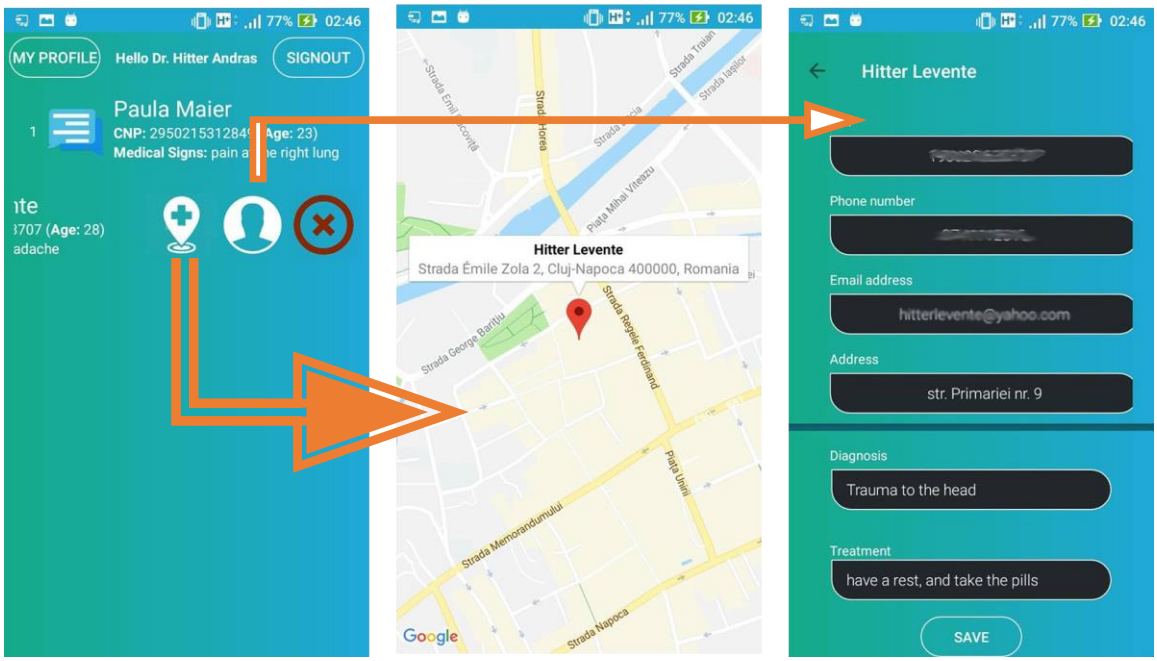

Figure 7.11 Doctor user guide 2

Chapter 8. Conclusions

In this chapter will be discussed the achievements of this project and also some ideas for developing the project even further.

8.1. Contributions and achievements

This project has created an opportunity for the technical development of the software developer.

Along the way he has gone through the following stages:

- He gained useful and solid knowledge about software development in Android such as: the concepts with which it operates, good practices related to mobile software development.
- He has actively participated in the analysis and design phases, which has formed an overall view of the project and also gained a perspective on possible implementations
- He implemented the system in an iterative manner which enabled to go through all the phases of software development for each iteration
- Has searched for different technical problems
- Found optimal solution in order to resolve every iterative task.

The project fulfilled all the proposed objectives from Functional requirements so the final project allows and helps its user in:

- Creating an account with a valid e-mail address
- Logging in the application with the registered e-mail address and the given password
- Seeing a list of patients/ doctors based on the user type.
- Viewing important information about the contacts that are in the contact list
- Sending well-composed messages to the contacts
- Calling doctors with a single press of a button
- Calling for help with a single press of a button and sending location data in message format at the same time
- Adding patients to the contact list
- Removing patients from the contact list
- Editing data that is important for patients (diagnosis, treatment)
- Tracking patients on the map in real time
- Recovering forgotten password
- Change profile data

The project helped in acquiring knowledge about different frameworks, messaging and tracking technologies, protocols, architectural patterns and many ways to solve a large variety of problems.

8.2. Further development

As any other system could be improved this system has room too for further improvements. To improve the user-experience the idea of some improvements came up that could be introduced in the newer versions of this project:

- Developing an internal chat module that could improve the response time in messaging and the user experience by knowing if the other part is available or not
- Reducing the GPS and battery usage as much as possible
- Implementing real-time tutorials that would show the steps for the user and would ask him to follow the instructions to do a task
- Background location tracking, would not only improve the accuracy of the tracker but it would reduce power consumption since the screen shouldn't be switched on all the time to track location
- Implement dropdown lists where the user can quickly choose from, for example: specializations, occupations, etc.

Bibliography

[1] Agarwal, Nitin. “Android MVP for Beginners – AndroidPub.” *AndroidPub*, AndroidPub, 18 Apr. 2017, https://android.jlelse.eu/android-mvp-for-beginners-25889c500443

[2] Ahmed Nabeel. “Message Broker vs. MOM (Message-Oriented Middleware).” *Stack Overflow*, Aug. 2016, Available: https://stackoverflow.com/questions/13202200/message-broker-vs-mom-message-oriented-middleware.

[3] Bertanga, Patrick. “How Does a GPS Tracking System Work?” *EETimes*, 2010, www.eetimes.com/document.asp?doc_id=1278363 .

[4] Das, Ankush. “Best Navigation Apps For Android.” *Ubergizmo*, Ubergizmo, May 2016, www.ubergizmo.com/articles/best-navigation-apps-android/

[5] Dossot, David. *RabbitMQ Essentials: Hop Straight into Developing Your Own Messaging Applications by Learning How to Utlize RabbitMQ*. Packt Publishing, 2014.

[6] “Firebase Cloud Messaging | Firebase.” Edited by Anonymus Editor, *Google*, Google, July 2018, https://firebase.google.com/docs/cloud-messaging/

[7] G, Aubrey. “HERE vs Google Maps Detailed Comparison as of 2018.” *Slant*, 2018, www.slant.co/versus/3750/6934/~here_vs_google-maps .

[8] Hashimi, Sayed Y., et al. *Pro Android 2*. Apress, 2010.

[9] “Hypertext Transfer Protocol.” Edited by Anonymus Editor, *Wikipedia*, Wikimedia Foundation, 6 July 2018, https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol .

[10] Johannson, Lovisa. “Get Started with RabbitMQ on Android (Android Studio).” *CloudAMQP*, 2015. Available: www.cloudamqp.com/blog/2015-07-29-rabbitmq-on-android.html

[11] Lehenbauer, Michael. “The 2^120 Ways to Ensure Unique Identifiers.” *The Firebase Blog*, 11 Feb. 2015, https://firebase.googleblog.com/2015/02/the-2120-ways-to-ensure-unique_68.html

[12] Kumar, Pankaj. “Java Design Patterns - Example Tutorial.” *JournalDev*, 2 May 2018, www.journaldev.com/1827/java-design-patterns-example-tutorial .

[13] Leiva, Antonio. “MVP for Android: How to Organize the Presentation Layer.” *Antonio Leiva*, 4 July 2018, https://antonioleiva.com/mvp-android/

[14] McTear, Michael, and Zoraida Callejas. *Voice Application Development for Android: a Practical Guide to Develop Advanced and Exciting Voice Application for Android Using Open Source Software*. Packt Pub., 2013.

[15] Moroney, Laurence. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. Apress, 2017.

[16] Muhammad Kamran Afridi, "Assistance System Service Development using a Message-Oriented Middleware" master's thesis, 2012, http://midas1.e-technik.tu-ilmenau.de

[17] “Non-Functional Requirement.” Edited by Walter Görlitz, *Wikipedia*, Wikimedia

[18] “SMS.” Edited by Anonymus Editor, *Wikipedia*, Wikimedia Foundation, 5 July 2018, https://en.wikipedia.org/wiki/SMS

[19] Sommerville, Ian. *Software Engineering*. Pearson, 2016.

[20] Svennerberg, Gabriel. *Beginning Google Maps API 3*. Apress, 2010.

[21] W., Raj Amal. *Learning Android Google Maps*. Packt Publishing, 2015.

Appendix 1 - List of figures and tables

Appendix 2 – Glossary

Term	Description
PT	Patient Tracker
GPS	Global Positioning System
API	Application Programming Interface
JSON	javaScript Object Notation
SDK	Software Development Kit
RAM	Random Access Memory
PIN	Personal Identification Number
SIM	Subscriber Identity Module
GNSS	Global Navigation Satellite System
OS	Operatin System
ETA	Estimated Time of Arrival
AMQP	Advanced Message Queuing Protocol
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
TCP	Transmission Control Protocol
SMS	Short Message Service
REST	Representational State Transfer
SSE	Server-sent Events
UID	User ID
CNP	Cod Numeric Personal (Personal Identification Number)
MVP	Model – View - Persenter
XML	Extensible Markup Language
GUI	Graphical User Interface
IDE	Integrated Development Environment