MINISTRY OF EDUCATION

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

FRITAXI TAXI BOOKING APPLICATION

LICENSE THESIS

Graduate: **Sorina Denisa POPESCU**

Supervisor: **S.L. Dr. Ing. Radu DAN**

2021

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
AUTOMATION AND INFORMATION TECHNOLOGY DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Sorina Denisa POPESCU**

Fritaxi Taxi Booking Application

1. **Project proposal:** *For the final project I propose an application for a taxi request. The users are able to register as customers or drivers and they are provided with the functions to work as a taxi driver or to request the closest taxi available. Several functions are added for a better user experience such as map routing, personal information for accounts and the option to add a destination*

2. **Project contents:** *Table of Contents, Introduction, Bibliographic Research, Analysis and Theoretical Foundations, Development and implementation, Tests and Results, Conclusions, Bibliography, List of tables and figures*

3. **Place of documentation**: Technical University of Cluj-Napoca, Automatics and Information Technology Department

4. **Consultants**:

5. **Date of issue of the proposal:** December 10, 2019

6. **Date of delivery:** February 08, 2021

Graduate: Popescu Sorina Denisa

Supervisor: S. L. Dr. Ing. Radu Dan

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
AUTOMATION AND INFORMATION TECHNOLOGY DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) Popescu Sorina Denisa,
legitimat(ă) cu Carte identitate seria CJ nr. 107819
CNP 2970102125479, autorul lucrării
Fritaxi Taxi Booking Application
elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea Automatica si Informatica Aplicata din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Februarie a anului universitar 2021, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data	Nume, Prenume
12.02.2021	Popescu Sorina Denisa

Semnătura

MINISTRY OF EDUCATION

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

Table of Contents

Chapter 1. Introduction

1.1 Project context

Modern society has always seemed to have a bit of an obsession with technology. But no single device has had as much of an impact on the world than the mobile phone. Back in the early days of mobile phones, their sole use was for calling other people whilst on the move. Soon after, the ability to ‘text’ other mobile phones was introduced.

The world is continuously changing through technology. Nowadays it is considered uncommon for people to not own a smartphone. From young children to elders, everybody is immersed in the limitless possibilities one has in its hands. IT companies have become more and more competitive developing new devices and functions every day. At this point their purpose is clear and they want to make people addicted to their smartphone by making their life more convenient.

Mobile applications have become a part of daily life and people are relying on them more than ever. People can listen to music, watch movies, write notes, search the internet, use the calculator, make appointments and many more.

Fast pace of the evolution of modern technology has increased the speed of progress in mobile programming as well. The software required by mobile devices that was not so long ago needed to write from scratch, has become widely available in the form of mostly free-source blocks of code. This has broadened the options and possibilities of what a programmer can achieve.

Therefore, for my thesis I chose to develop a mobile application for ordering a taxi. With a user friendly interface, people will easily be able to register and request a taxi from their location. The application will search for the closest taxi driver that is available. Moreover, the user is also able to register as a taxi driver and he can pick up the requests of the users registered as clients that are within his area. I named the application Fritaxi, representing a merge of the words ‘friend’ and ‘taxi’, a friend that will come to pick you up as soon as possible.

In the following chapters I am going to present the steps that were required to convert my ideas into a functional android application, Fritaxi. Starting with making a clear list of goals and functions I would have to develop, the intense research that followed, the process of development and finally concluding with the tests runned to ensure a correct flow of the events. Lastly I considered the possible improvements that can be brought to the code, and to the interface, to satisfy the users and bring more interest over its competitors.

1.2 Objectives and specifications

In the present, time has become the most valuable resource we have. Everybody has had, at least once in their lifetime, to arrive at a certain destination within a limited time period. To reach that place, they had to use some sort of transportation method. Whether they took a bus, a train or a taxi, every method required walking to a specific station and thus losing minutes from that precious time. The problem of bringing transport closer to us has been dealt with by software engineers over the years. Mobile phone applications have been developed with the goal of saving as much time as possible. Examples of such applications are train booking, taxi booking and plane booking.

Firstly, I will focus on how to take a taxi. This can be done in three methods. Firstly, one can walk to the station, look for a taxi that is parked there, ask the driver if he is available and only then proceed to the destination. The second method implies that the user has a mobile phone. Then this person can make a phone call to request a taxi. This option is especially slow considering the fact that each taxi company has a different company number. In other words, one can not have access to all the available taxis through a phone call. During the busy hours, when people are leaving or going to work or when the weather conditions are unfavorable, finding a free taxi could be difficult and that person would have to call every company in town and try his luck finding a driver. Considering these two options, it was inevitable that mobile applications for booking a taxi would have been developed. This method gives us the benefit of searching for available taxis through all the companies within the surrounding area with a simple button that represents a request.

For this project I chose to develop an application for requesting a taxi and for registering and working as a taxi driver. The application is made in Android Studio and has a database in Firebase. The Android operating system is available on the most variety of smartphones and it is also a program which allows free coding unlike its biggest rival, the IOS operating system. Meanwhile, Firebase provides easy implementation of a database for mobile and web applications.

The main objectives of my application, named Fritaxi, are:

- Analyzing the need for such a system in the present;
- Studying the structure and the functions given by booking applications, while also looking at the current applications available on smartphones;
- Developing and implementing a mobile phone application which corresponds to the current needs of the user offering him the possibility to request the closest available taxi and also giving the users the option to work as taxi drivers;
- Testing the solution and finding possible improvements for the software.

Chapter 2. Bibliographic Research

2.1 Mobile Phones - a short history

In order to understand the impact of Android and smartphones in our daily lives, we need to think about the beginning of mobile phones. By observing their evolution in time, we can recognize how the android applications simplify our tasks.

Everything started in 1973 when Dr. Martin Cooper, a senior advancement engineer at Motorola, called an adversary media transmission organization and said that he was calling through a cell phone. Regardless of the new achievement, the telephone utilized weighed 1.1kg and gave 30 minutes of talktime following 10 hours of energizing. In 1982, the government commission affirmed the foundation of the cell phone framework and apportioned simple recurrence to be utilized by the organization.

When compared with the subtle and cheap mobile phones available within the markets today, the primary cellular phone might sound too primitive but that's how it all started. Cell phones now and 20 years ago were starkly different in the majority of aspects, right from their uses, price tags, sizes and battery life. While those mobile phones hardly empowered an individual to have interaction during a long voice call, modern cell phones and smartphones not only facilitate long voice and video calls but also exciting features and services like text messaging and voice mails which ensures that the users are always connected to each other.

By the late 1990s, mobile phones started to become increasingly available to the general public. The most well known at the time were Motorola and Nokia. Sony Ericsson, Panasonic, LG and HTC entered the market soon after, followed by Samsung, Apple and Blackberry.

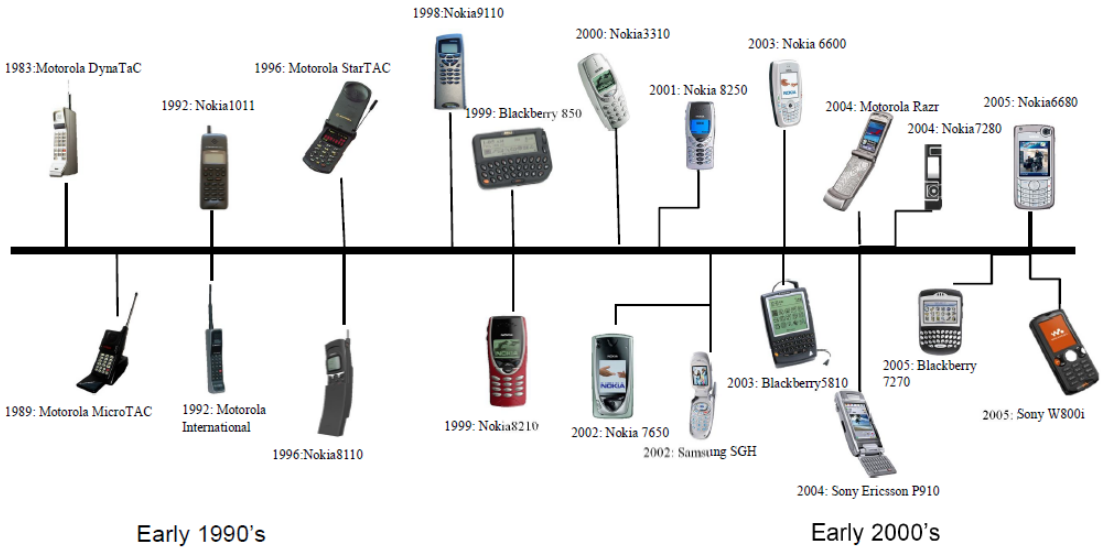

Figure 1. Cell-Phone Evolution

As a new decade approached, mobile phones became a stylish gadget. They began their role as an accessory to match the personality of its user. Models like Nokia N95, Samsung Blackjack and Nokia E90 were released. However, in 2007, Apple unveiled the world's first iPhone, which is a touchscreen smartphone. This phone was the first of its kind and it enabled users to run apps (applications) designed for specific purposes. Apps exist for a wide range of things including movies, books, games, navigation and security. The year 2011 marked the return of touchscreen which dominated the mobile gadget scene with its powerful and sleek look. Samsung introduced the Galaxy SII which became a major competitor to the Apple iPhone. [1]

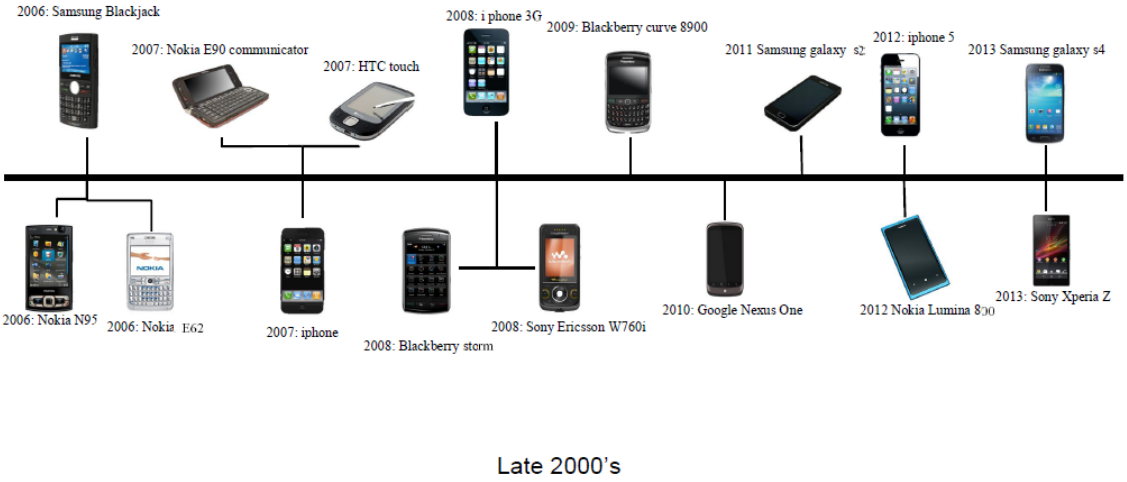

Figure 2. Evolution of smartphones

Mobile phones have evolved considerably in terms of technology. They began as simple devices for making calls to all-in-one handsets that make our daily lives easier. They have not only enhanced communication, but have made our lives easier.

2.2 What is Android?

Android Inc. was established in California, in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White. Rubin portrayed the Android project as having 'colossal potential in creating more astute cell phones that are more mindful of its proprietor's area and inclinations'. [3] tThe first goals of the organization were to make up a high level working framework for advanced cameras. The organization at that time concluded that the marketplace for cameras wasn't enormous enough for its objectives, and after five months it had redirected its endeavors and was pitching Android as a handset working framework that might match Symbian and Microsoft Windows Mobile.

Android is a Linux based operating system designed primarily for touch screen mobile devices such as smartphones and tablet computers. The operating system has developed a lot in the last 15 years starting from black and white phones to recent smartphones or mini computers.

Various drivers and libraries have been altered or presented in android so it can perform all the more successfully and proficiently on cell phones when contrasted with Linux 2.6. A few libraries have their roots in open source projects. Android affiliation chose to assemble their own c library and java runtime motor to avoid the permitting issue. The objective of android is to flaunt their foundation with the restricted assets offered on cell phones. Android can be articulated as the best working climate as it is giving the middleware components,application and incorporating the OS. To summarize, the Android working climate can be best portrayed as an open stage for versatile extension and growth,hardware reference plan for versatile devices,a framework ran by an overhauled Linux 2.6 Kernel, a runtime climate and an application and User Interface(UI) framework.A a-list and a remarkable stage have been given to the designers of android to make their applications for and clients. Moreover, all the most recent media examination designs are upheld by android.

There are currently over 1.500.000 applications available for Android. Android Market is the online application store run by Google, though applications can also be downloaded from third-party sites. Developers write in the Java language. [3]

2.3 Android Versions

The version history of the Android mobile operating system began with the public release of the Android beta on November 5, 2007. The first commercial version, Android 1.0, was released on September 23, 2008. Android is continually developed by Google and the Open Handset Alliance (OHA), and it has seen several updates to its base operating system since the initial release. [5]

In the figure 3 is represented a table with the versions of android, from the first one developed in 2008 to the newest one, Android 11 developed in September 2020. Alongside every version of Android is a code name, which many people use instead of the version number. Each one is named after a dessert or some other form of confection.

As with Android 10 before it, Android 11 includes a number of new user-facing changes and features. Most prominent is a built-in screen recorder, smart home controls in the power menu, revamped media controls, and a dedicated space for messaging notifications.

Name	Version number(s)	Initial stable release date
No official codename	1.0	September 23, 2008
	1.1	February 9, 2009
Cupcake	1.5	April 27, 2009
Donut	1.6	September 15, 2009
Eclair	2.0 – 2.1	October 26, 2009
Froyo	2.2 – 2.2.3	May 20, 2010
Gingerbread	2.3 – 2.3.7	December 6, 2010
Honeycomb	3.0 – 3.2.6	February 22, 2011
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011
Jelly Bean	4.1 – 4.3.1	July 9, 2012
KitKat	4.4 – 4.4.4	October 31, 2013
Lollipop	5.0 – 5.1.1	November 12, 2014
Marshmallow	6.0 – 6.0.1	October 5, 2015
Nougat	7.0 – 7.1.2	August 22, 2016
Oreo	8.0 – 8.1	August 21, 2017
Pie	9	August 6, 2018
Android 10	10	September 3, 2019
Android 11	11	September 8, 2020

Figure 3. Android Versions

Along with the change of names, the android logo has also been adjusted accordingly. In the figure 4, which can be seen below, we can observe how the logo took the form of a small green robot and the sweets from its representation are either a donut, gingerbread,cupcake, eclair or any dessert name that is found in the name of the respective android version.

Starting with version 9.0, the logo has been simplified. The main reason for the change has been the increased popularity of android smartphones in the latest years. Thus the green robot is widely recognized as the symbol for android.

Figure 4. Android version and logo evolution

2.4 Why Android?

Android Studio came out as a replacement to Eclipse. It is an official IDE(Integrated Development Environment) that is used to create applications for the Android Platform. Some of the main reasons it is preferred over other development environments are : [8]

- Android Studio is Free: there are no membership taxes, the user doesn't have to pay for a licence and it is also free deploy the applications on mobile phones and to upload them on Google Play Store;
- Faster Coding: the platform provides instant build and run of the code, great emulators to test the applications, templates and sample applications and Firebase and Cloud Integration services;
- A complete platform: aside from a program specialized in databases such as Firebase, there is no need for other auxiliar technologies to be used in order to develop a complete project;

In conclusion, Android Studio is the coding environment preferred all over the world for the development of mobile applications. Moreovers, it allows the users to update the and bring improvements to the code for an unlimited time period.

2.5 Android Studio Architecture

Any software or system is based on some set of components that work together to accomplish some certain task or to perform some specific function. For a better understanding of the android system architecture, the figure 5 presents the layers of the operating system.

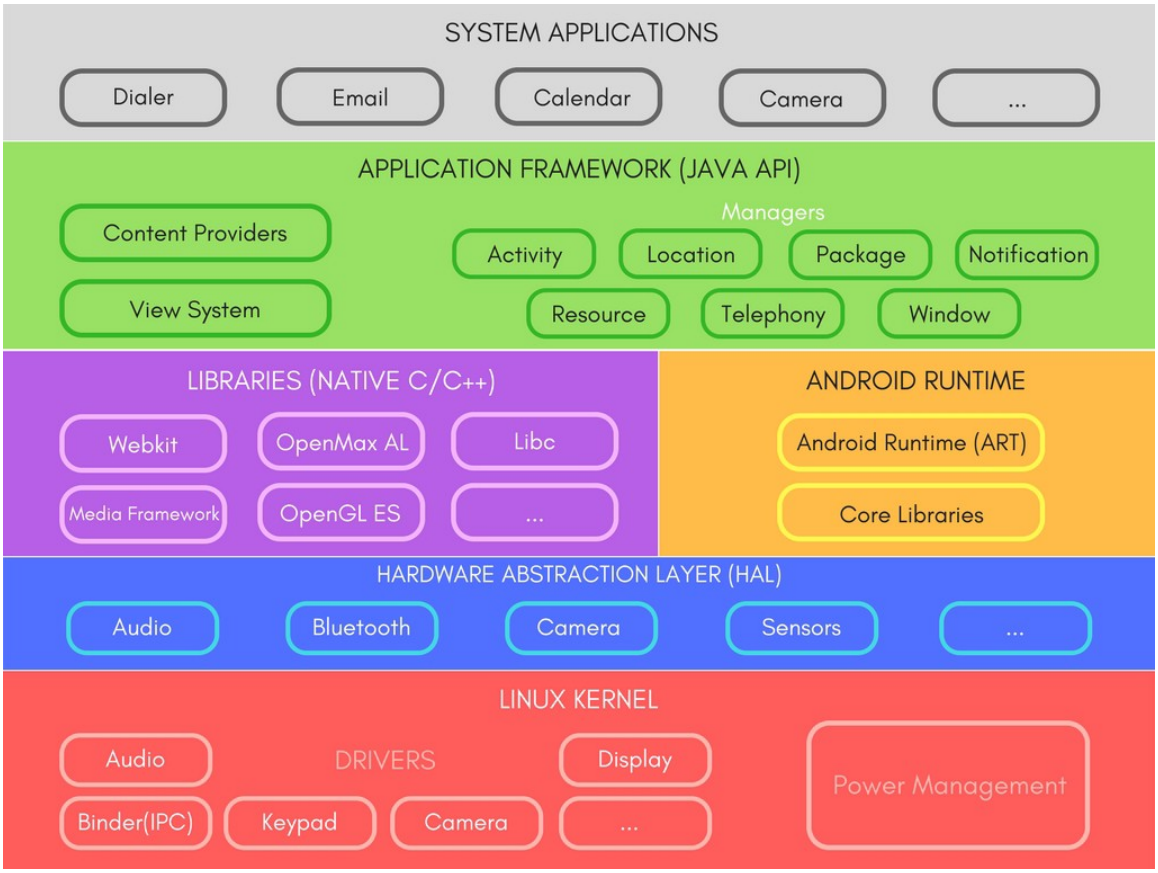

Figure 5. Android System Architecture

Android OS is based on the stack of six main components:

- Application: represents the applications that we all make and use. They reside on the topmost layer of Android Architecture; [9]
- Application Framework: provides many higher-level services to applications in the form of Java classes and also contains other APIs. Application developers are allowed to make use of these services in their applications;[9]

- Libraries: Above Linux kernel, there is a set of libraries including open-source Web browser engine WebKit, well-known library libc, SQLite database, libraries to play and record audio and video, SSL libraries responsible for Internet security etc. All Java based libraries that are needed for building Android apps also reside here;[9]

- Android Runtime: This portion contains one of the most important components Dalvik Virtual Machine, DVM. This is a kind of JVM that is specially optimized and made for Android. It enables every android app to run its own process which enables us to run many applications at same time. It basically converts .java file into .Dex format;[9]

- Hardware Abstraction Layer (HAL): A HAL defines a standard interface for hardware vendors to implement, which enables Android to be agnostic about lower-level driver implementations. Using a HAL allows you to implement functionality without affecting or modifying the higher level system. HAL implementations are packaged into modules and loaded by the Android system at the appropriate time; [9]

- Linux Kernel: the core of any OS. The Linux Kernel sits at the bottom of Android architecture. The Android platform is built on top of the Linux 2.6 Kernel. The Linux Kernel provides support for many features like memory management, security management, process management, and device management etc. It also contains all the drivers that help Android devices to communicate with other hardware devices; [9]

2.6 The Virtual Machines of Android Studio

A virtual machine is based on computer architectures to provide the functionality of a computer. There are 2 main types of Virtual Machine (VM):

- System virtual machines (full virtualization VMs) provide a substitute for a real machine.

- Process virtual machines are designed to execute computer programs in a platform-independent environment.

The architecture of a virtual machine consists of multiple Guest Systems, a hypervisor, other applications on the host operating system, a host operating system and physical hardware.

2.6.1 The Virtual Machine Java (VMJ)

A Java virtual machine (JVM) is an abstract layer between the Java program and the platform the Java code is running on. JVM is platform dependent and different implementations are available for specific platforms.

For example, for a *Hello.java* class and when this class file is runned then javac compiler turns the source code to bytecode and creates *Hello.class* file which means javac compiler does not convert Java code directly to machine code like other compilers do. Bytecode is intermediate code which means humans can not understand this code and this code is not machine/platform dependent. Since bytecode is an intermediate code so it can be given to anyone to run it on any platform. That's why Java applications are called WORA (Write Once and Run Anywhere). [11]

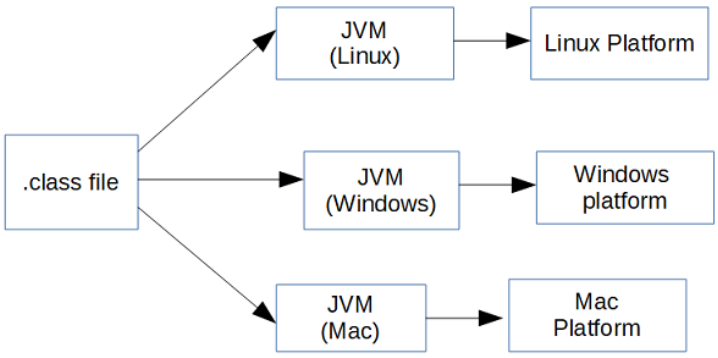

Figure 6. JVM class compiler

The Figure 6 represents how once a *.class* file is ready, then it can be given to any platform and it will convert it to native machine code.

2.6.2 The Virtual Machine Dalvik (DVM)

The Dalvik Virtual Machine (DVM) is a virtual machine which executes the android applications. Since everything in mobiles is very limited whether it would be battery life, processing and memory etc. It had been optimised so that it can fit in with low-powered devices. [11]

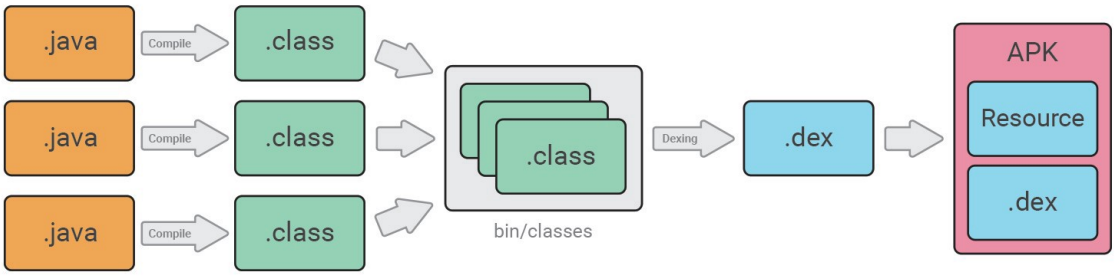

Figure 7. DVM Structure

As it can be seen in Figure 7, everything is the same as JVM except the last two steps. The Dex compiler converts the class files into the .dex file that runs on the Dalvik VM. Multiple class files are converted into one dex file.

2.6.3. JVM vs DVM

One of the main reasons of using DVM in android is because it follows the register based model and it is much faster than stack based model while JVM follows the stack based model which takes a lot of memory and also slower than DVM.

There are some major differences represented in the picture below [12]:

DVM (Dalvik Virtual Machine)	JVM (Java Virtual Machine)
It is Register based which is designed to run on low memory.	It is Stack based.
DVM uses its own byte code and runs ".Dex" file. From Android 2.2 SDK Dalvik has got a Just in Time compiler	JVM uses java byte code and runs ".class" file having JIT (Just In Time).
DVM has been designed so that a device can run multiple instances of the VM efficiently. Applications are given their own instance.	Single instance of JVM is shared with multiple applications.
DVM supports Android operating system only.	JVM supports multiple operating systems.
For DVM very few Re-tools are available.	For JVM many Re-tools are available.
There is constant pool for every application.	It has constant pool for every class.
Here the executable is APK.	Here the executable is JAR.

Figure 8. DVM and JVM comparison

2.7 Android Activities

2.7.1 The concept of activities

The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin within the same place. Instead, the user journey often begins non-deterministically. For example, when someone opens an email app from their home screen, they may see an inventory of emails. In contrast, if they're employing a social media app that then launches your email app, they may go on to the email app's screen for composing an email.

The Activity class is meant to facilitate this paradigm. When one app invokes another, the calling app invokes an activity within the other app, instead of the app as an atomic whole. during this way, the activity is the entry point for an app's interaction with the user.

An activity provides the window during which the app draws its UI. This window typically fills the screen, but could also be smaller than the screen and float on top of other windows. [13] Generally, one activity implements one screen in an app. As an example, one in all an app’s activities may implement a Settings screen, while another activity implements a pick Map screen.

Most applications contain numerous screens, which implies they include various activities. Regularly, one movement in an application is indicated as the primary action, which is the main screen to show up when the client dispatches the application. Every action would then be able to begin another movement to perform various functions. For instance, the fundamental action in a basic email application may give the screen that shows a registration web page. From that point, the principal action may dispatch different activities that give screens to assignments like registering a new user and login to an existing account.

In spite of the fact that exercises cooperate to frame a durable client experience in an application, every movement is simply approximately bound to different activities; there are normally insignificant conditions among the activities in an application. Activities regularly open up activities having a place with other applications. For instance, a program application may dispatch the Share action of an online media application. [13]

In order for the activities to be functioning inside an application, they must be declared in the *Manifest* file of the application and the activities life cycles have to be managed accordingly.

2.7.2 Activities Lifecycle

Every Android application runs on its own processes. Processes are started and stopped to run an application and also can be killed to conserve memory and resources.

Activities, in turn, are run inside the main UI thread of the application's process. In other words, activity is a screen that the user interacts with. Every activity in Android has a lifecycle like created, started, resumed, paused, stopped, or destroyed. These distinctive states are known as *Activity Lifecycle*.

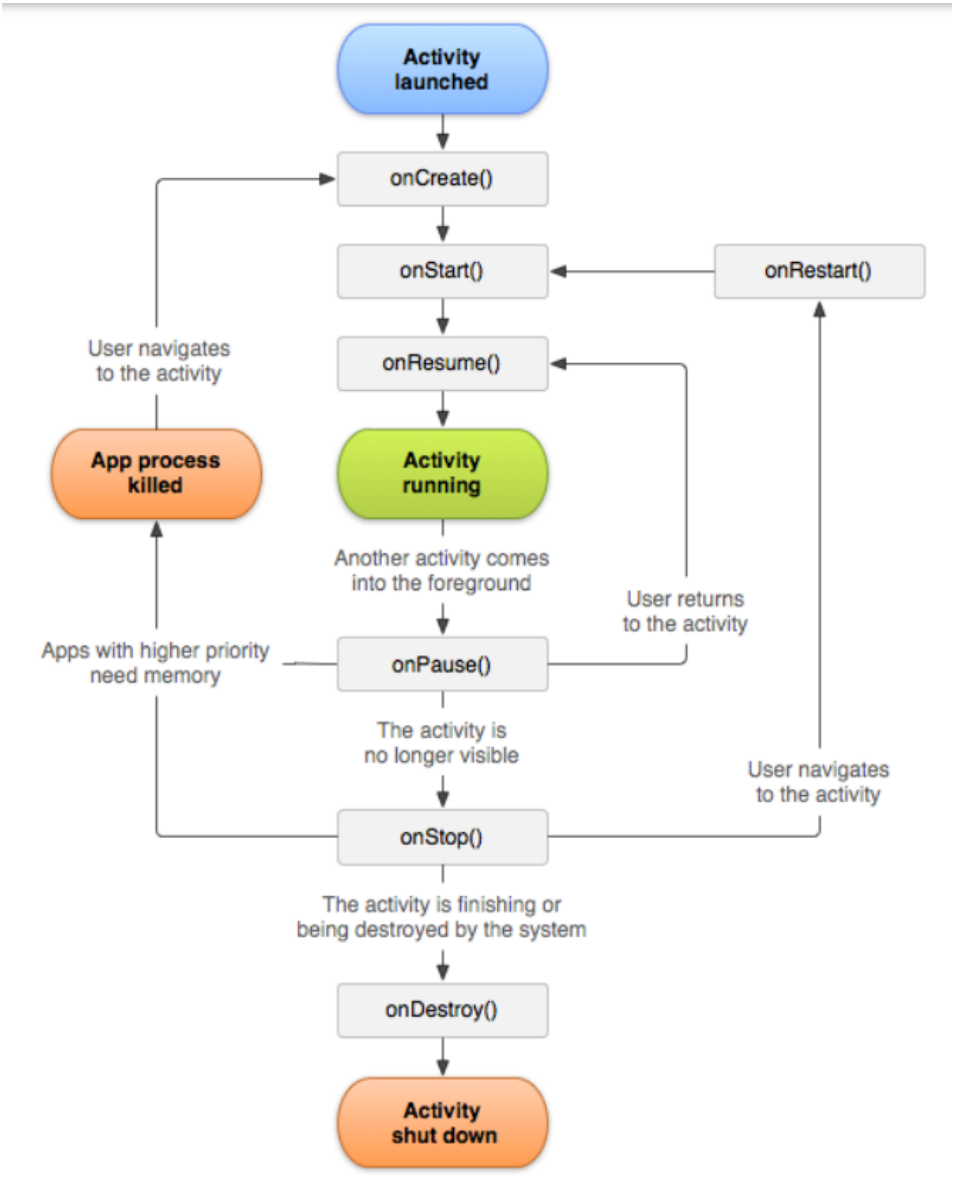

Figure 9. Activity Lifecycle Flow

The above picture shows the complete flow of the Activity lifecycle.

The Activity class provides a set of six callbacks to navigate transitions between stages of the activity lifecycle. Better implementation of the lifecycle callbacks can help to avoid crashing, losing the user progress, and screen orientation problems. Next i will present the main characteristics of each callback:

- **onCreate()**: Must be implemented at the beginning. It is the basic activity creation startup stage. This callback has to be implemented to perform basic application startup logic which will happen only once in the whole lifecycle of the activity (Ex. the opening of an Application);
- **onStart()**: The system invokes this callback after the activity arrives to the "started" state. When an activity gets to the start method, the application will show the main screen (Ex.when the user opens the application he will be led to the homepage screen);
- **onResume()**: the activity stays still, nothing happens, until the user interacts with any activity such as selecting or clicking on any options (buttons) for another activity to start. When an event interrupts the session, the activity goes to the paused state, and the system invokes the **onPause()** callback (Ex. The user, for a couple of minutes, doesn't interact with the application after opening it);
- **onPause()**: When the user is switching from one activity to another, the current one gets paused, therefore the entire process of this activity will either go to the Resume state or it will be killed. In most cases, the on pause() method is called by the system when the user presses the Home button (center button on the device). To broadcast receivers, or any resource that affects the battery life while the activity is paused, the onPase() method can be implemented to release the system resources. (Ex.When the user receives an incoming call, the current application he is using is paused and it is restarted when he clicks on it);
- **onStop()**: This callback is called when the activity is no longer visible to the user. This is so because it has entered the stopped state. The system may also call the onStop() method when the activity has finished running and is about to be terminated; [14]
- **onDestroy()**: the final call that the activity receives. It may also be called when a change in the screen orientation occurs, and then it calls **onCreate()** to recreate the activity. The onDestroy() callback releases all resources that have not yet been released by earlier callbacks;
- **onRestart()**: called when the activity is being restarted, for example when the activity is returning to the starting point. It must be followed by the **onStart()** method;

2.8 Android SDK and ADT instruments

The **Android SDK** (software development kit) is a collection of software development tools and libraries required to develop Android applications. The **Android SDK** comprises all the tools necessary to code programs from scratch and even test them. [15] These tools provide a smooth flow of the development process from developing and debugging, through to packaging. The tools represent:

- Required libraries;
- Debugger;
- An emulator;
- Relevant documentation for the Android application program interfaces (APIs);
- Sample source code;
- Tutorials for the Android OS.

The Android Development Tools (**ADT**) is a plugin which adds powerful extensions to the Eclipse integrated development environment. It allows the user to create and debug their Android applications easier and faster. [16]

ADT gives the user access to other Android development tools from inside the Eclipse IDE. For example, ADT allows the user to access the many capabilities of the DDMS tool: take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse. It also provides a New Project Wizard, which helps the user quickly create and set up all of the basic files they'll need for a new Android application. Lastly, ADT provides an Android code editor that helps at writing valid XML for Android manifest and resource files.

2.9 Devices with Android Operating System

The devices running the Android operating system have appeared in all shapes and sizes. The most well-known and widespread such devices are smartphones. In addition, there are other models on the market, such as: tablets, ereader devices, devices for accessing files with MP4 extension (MP4 players), netbook laptops, smartwatch, on-board computers for cars, game consoles and others.

Based on studies, research and observations conducted in the field, it is expected that a significant increase in the number and diversity of devices running the Android operating system is expected, to exceed the number of devices with Windows operating system, which are currently the more widespread, but also greater than the number of iOS devices, their main competitor.

2.10 State of Art

As of today, there are several applications available for android smartphones. In this part, I will present the applications that represent a source of inspiration in developing my own taxi application.

First of all, the most important component of the application I developed is represented by the connection to GPS tracking service and the google map function. These components will be explained in detail in the next chapter. Next i will present details regarding the most popular application as of today with the components mentioned before.

Google Maps and **Waze** are both excellent GPS apps. They are also both developed by Google. **Google Maps** is kind of the measuring stick for navigation apps. It has tons of locations, reviews, directions, and street-level photography of most locations. Additionally, it has accessibility features and the user can download maps for offline use. **Waze** is a little more simple. It's great for directions, especially on road trips or daily commutes and allows users to mark roads in case of irregular traffic conditions. [18]

Polaris Navigation tries to be the all-in-one navigation app and in most cases it succeeds. Its biggest feature is that it has access to Google Maps, OpenStreetMap, MapQuest maps, and Cycle Route Maps. [18]

Zift/Net Nanny, which is considered to be the best parental control app, has excellent web-filtering technology and a modern, intuitive design. Net Nanny can track the user's child's location, display their location history, and set time allowances and schedules equally well on both platforms. The iOS version lets the user block several dozen apps on your kid's phone; the Android one lets you block them all. [19]

Life 360 is quite a popular family tracking app complete with a range of amazing features. Its real usability is that users can add circles and groups for their family and friends to detect their locations along with a log of their past locations. [19]

An important category of applications based on GPS tracking is represented by taxi booking apps. The competition is high in this category and each developer adds functions meant to attract more users. The taxi booking applications main functions are the registration of the users, the option to order a taxi and finishing the rides. Thus the users that are registered as consumers are saving time that would normally be lost searching for an available driver or calling companies. Moreover, it also sends the actual location of the customer to the driver and he will know exactly where the pick up location is. The key behind this kind of technology is the GPS Service found on every existing smartphone.

In order to develop my application, i tried several taxi booking applications that can be found in Google Play, the store for android mobile applications. In the following rows, I will present some of the most common and used applications.

Available in other cities in the country, not only in Bucharest, **Clever Taxi** allows the user to interact directly with taxi drivers and even choose a favorite driver to prefer. If the taxi driver is registered in the application, they will be able to pay for the trip, including with the card. When the user first registers in the application, he will be able to choose the taxi companies he wants to collaborate with. [18]

Created by a team of Romanian developers, **Star Taxi** in an application available for Android, but also iPhone. Among the advantages of the application the users have the possibility to follow the taxi orders in real time and check the selected driver based on the comments left by other customers who used his services. In addition, they can exchange text messages with the taxi driver, for example to give him additional instructions on finding the location or waiting time. [18]

Taxify(Bolt) application is available for iOS and Android and works in Bucharest, Timisoara and Cluj Napoca, but also in other European and world cities. Taxify is very simple to use: request a ride from your location, follow the route on the map while traveling to the chosen destination and pay at the end right from the application, using your bank card details. In other countries, Taxify offers a combination of ride-sharing (Uber) and the services of licensed taxi companies.

Launched by an American company in 2010, the **Uber** service has been available in Romania since 2015, being currently considered one of the main threats to traditional taxi companies. With rates comparable to a regular taxi, Uber operates exclusively with individual drivers and their personal cars. [17]

Like Uber and Taxify, **Black Cab** is a ride-sharing taxi service that works with drivers and private vehicles. The main difference is the premium range: Black Cab cars are top of the range and very well maintained, and the driver's behavior must be impeccable. [17]

Figure 10. Mobile Taxi Application

Chapter 3. Analysis and Theoretical Foundations

3.1 General Concepts

In the last couple of years, the interest towards mobile devices, especially for smartphones, has increased rapidly. Thus more companies have centered their focus on developing applications for smartphones that attract the users making their tasks easier. The users have access to the internet through the broadband cellular networks (2G, 3G, 4G, 5G, HSPA+). The year 2019 marks a step forward towards technology as the 5G was being developed and as of 2020 the newer smartphones included it. In other words, the users have access to better and faster internet connectivity. Considering the previous facts, the number of applications available today, whether they are free or not, is not peculiar.

One of the most popular categories of mobile applications is represented by the apps based on GPS Tracking. From the first cell phone that was called a smartphone to the latest models, all of them include the GPS Service. This means that the user is able to see his position on the world map with his phone. The user can download an application called Google Map, available on all operating systems, and by enabling the GPS Service, their current position is marked on the world map. An important mention is the fact the map doesn't only track the user position, but it also displays the surroundings, including parks, important buildings, entertainment venues and so on. This feature allows the users to use the application for navigation. Considering the previous facts, the developer's interest toward GPS based applications has increased and today the users have access to a multitude of applications such as car navigation, family members tracking and taxi booking.

Before developing Fritaxi, I focused on the GPS applications that are already available for Android. By understanding their functions, I started to get a clearer image of how the user interface should look like. In the previous chapter I presented the most important applications in developing my own taxi app.

In this project, I will present a software application for booking a taxi without having the constraints of a country or city. Fritaxi will help the people who are in a hurry and need a taxi and the people who are trying to become taxi drivers but find difficulty in getting hired by a company. For the registration the user will need a smartphone and internet connection. He will have the option between registering as a customer or a driver. The application will proceed to ask for permissions in order to access his current position and display it on the map. In the case of the customer, he will have a button for requesting a taxi. Once a request is made, the application will search for the closest available driver. Another option is introducing a destination before requesting a taxi which will be sent to the driver immediately. In the case of the users registered as drivers, they will be provided with a switch representing their status as working or not working. Thus once a driver wants to take a break he will only need to switch off instead of logging out. The taxi requests from the surrounding area of the user will be picked up

automatically and an option for ending the current ride will pop up. For a more user friendly experience, both users will have the option to add his name and phone number, information which will be transmitted to the customer, or driver, through the tax request.

Upon testing a couple of applications, it is remarkable that each one has at least one unique function that makes it succeed its competitors. In the case of Fritaxi, as a distinctive characteristic, I used markers for the positions of the users and also for the distances between a customer and a driver and for the ride that was initialized with a destination. For even better user experiences, several messages have been added through the application such as 'The driver has arrived'.

3.2 Technologies Used

For the implementation of my project, Fritaxi, I used a couple of software applications connected together to form a complete mobile application available for the smartphones with android operating system. The users will need an internet connection that is given through the GPRS or Wi-fi services. The internet connection makes possible the communication between the android system and the database. The association between these concepts is more visible in Figure 11.

The first programs to be mentioned are Android Studio, where the code is implemented using Java and OOP concepts, and Firebase which contains the database with the information about users and requests is stored. Another word, often mentioned in this thesis, is GPS services. In the following subchapters I will focus on explaining these softwares.

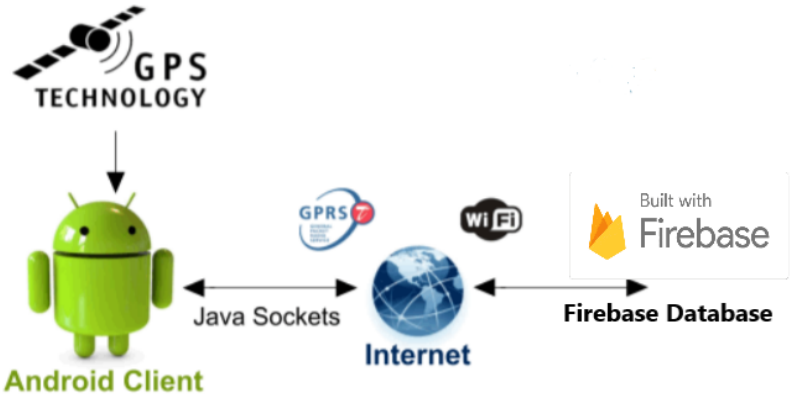

Figure 11. Implementation Scheme

3.2.1 Object-Oriented Programming

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of this or self). In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types. [20]

There are four basic principles of OOP:

- **Inheritance**: the process of creating new classes, called derived classes, from base classes where an "is-a" relationship exists. The derived class extends from the base class in order to inherit its properties;

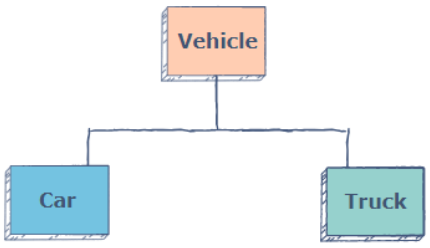

Figure 12. Example of inheritance

- **Polymorphism**: the ability of an object to take on many forms. For example, square, rectangle, circle and triangle classes can have the same parent class Shapes, but each class can have its own method named drawShape();

Figure 13. Example of polymorphism using the class drawShape

- **Abstraction**: showing only the essential features of an object to the user and hiding the other details to reduce complexity.

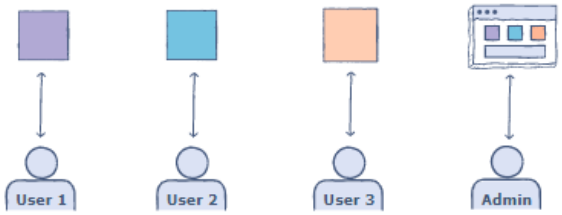

Figure 14. Example of Abstraction

- **Encapsulation**: binding the data and the methods to manipulate that data together in a single unit (class). This is normally done to hide the state and representation of an object from outside.

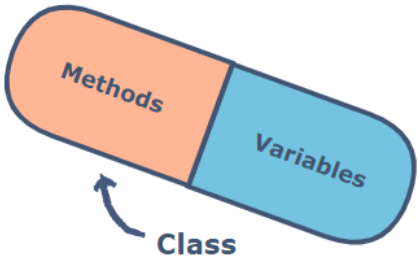

Figure 15. Example of Encapsulation

The design of Object-Oriented Programming is based on entities such as classes and objects. Classes are software models that describe general properties of entities that the software system works with. Objects are instances of the class of objects in which they fall, a class being, in fact, a model, a sketch according to which the objects are created. [20] [21]

3.2.2 Java Programming

The Java programming language was developed by Sun Microsystems in the early 1990s. Although it is primarily used for Internet-based applications, Java is a simple, efficient, general-purpose language. Java was originally designed for embedded network applications running on multiple platforms. It is a portable, object-oriented, interpreted language.

A program, written in the Java programming language, is compiled and runs on a Java virtual machine (JVM), which makes platform applications possible for Java applications. This virtual machine interprets the byte code (.class files) obtained from compiling source files (.java files). [22] [23]

3.2.3 The Client-Server Model

Most modern network programming is based on a client/server model. The purpose of the client/server application is to store large quantities of data on an expensive, high-powered server. Therefore most of the program logic and the user interface is handled by client software running on personal computers. In most cases, a server primarily sends data, while a client primarily receives it. In rare cases, one program is used to send or to receive exclusively. A more reliable distinction is that a client initiates a conversation, while a server waits for clients to start conversations with it. Figure 16 illustrates both possibilities.

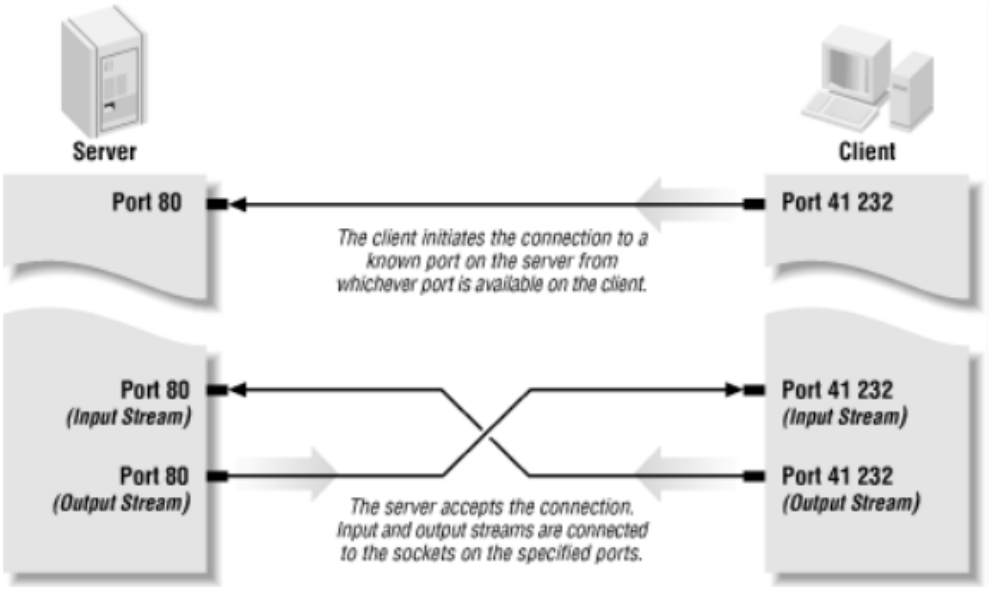

Figure 16. Client Server System

Some servers process and analyze the data before sending the results to the client. Such servers are often referred to as “application servers” to distinguish them from the more common file servers and database servers. A file or database server will retrieve information and send it to a client, but it won’t process that information. In contrast, an application server might look at an order entry database and give the clients reports about monthly sales trends. An application server is not a server that serves files that happen to be applications. [24]

3.2.4 The GPS Technology

3.2.4.A Global Navigation Satellite Systems

The meaning of GNSS (Global Navigation Satellite System) is the technical interoperability and compatibility between various satellite navigation systems such as modernized GPS, Galileo, reconstructed GLONASS to be used by civilian users without considering the nationalities of each system in order to promote the safety and convenience of life.

The concept of reference system for navigation is essential since all the applications of GNSS are related to the coordinate system used. The main application of GNSS is focused on the potential of to determine the position in the Global reference system any where any time on the Globe in a simple, fast and cost-effective manner. [27] [28]

The GNSS consist of three main satellite technologies: GPS, Glonass and Galileo. Each of them consists mainly of three segments:

- **The space segment**: represents a constellation of satellites, generally 24 orbital satellites (eight each, organized in three circular orbits, later structured in six orbits, each with four satellites). The orbits are centered inside the Earth, but do not rotate with the movements of the planet, but remain fixed so as to respect a correct alignment, i.e. all four satellites are visible for a certain point for several hours every day. Each satellite in the constellation emits radio frequency (RF) signals, modulated by codes and navigation messages;

- **The control segment**: consists of a central control station, an alternative control station, four dedicated ground antennas and five dedicated monitoring stations on Earth, which manage the proper functioning of the satellites, by monitoring them and updating the messages. navigation, calculating their spatio-temporal positions;

- **The user segment**: includes the GPS radio navigation receivers of the user community such as numerous military users, both civilian, commercial and scientific users. The receivers specialize in receiving decoding and processing codes and navigation messages, converting spacecraft (VS) signals into position, speed and time estimates. These receivers consist of several components, among the most important being: the power supply system, the antenna with the signal amplifier, the microprocessor, the high frequency oscillator, the control unit and the memory for data storage. Depending on the qualities of the receiver and the antenna, the accuracy of the positioning accuracy or the navigation elements is determined. [26]

3.2.4.B What is GPS?

The Global Positioning System (GPS) is a satellite-based navigation system made up of at least 24 satellites. GPS works in any weather conditions, anywhere in the world, 24 hours a day, with no subscription fees or setup charges. The U.S. department of Defense (USDOD) originally put the satellites into orbit for military use, but they were made available for civilian use in the 1980s.

GPS satellites circle the Earth twice a day in a precise orbit. Each satellite transmits a unique signal and orbital parameters that allow GPS devices to decode and compute the precise location of the satellite. GPS receivers use this information and trilateration to calculate a user's exact location. Essentially, the GPS receiver measures the distance to each satellite by the amount of time it takes to receive a transmitted signal. With distance measurements from a few more satellites, the receiver can determine a user's position and display it electronically to measure your running route, map a golf course, find a way home or adventure anywhere.

The GPS provides particularly coded satellite signals that can be processed in a GPS receiver, allowing the receiver to estimate position, velocity and time (Hofmann-Wellenhofvet al., 2001). There are four GPS satellite signals that are used to compute positions in three dimensions and the time offset in the receiver clock. The segments specific for the GPS are: the **space segment**, consisting of the GPS satellites which represent a couple of space vehicles (SVs) that send radio signals from space, the **control segment**, which includes a system of tracking stations located around the world, and the **user segment**, formed by the GPS receivers and the user community. GPS receivers convert space vehicle (SV) signals into position, velocity, and time estimates.

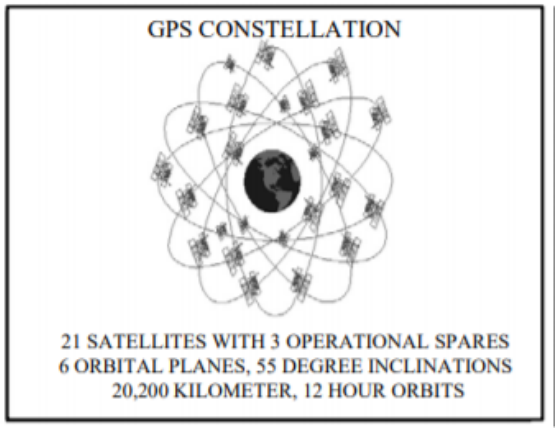

Figure 17. GPS Constellation

The satellites are dispersed in six orbital planes on almost circular orbits with an altitude of about 20,200 km above the surface of the Earth, inclined by 55 degrees with respect to the equator and with orbital periods of approximately 11 hours 58 minutes (half a sidereal day). Figure 18 shows the main GPS segments. [25][28]

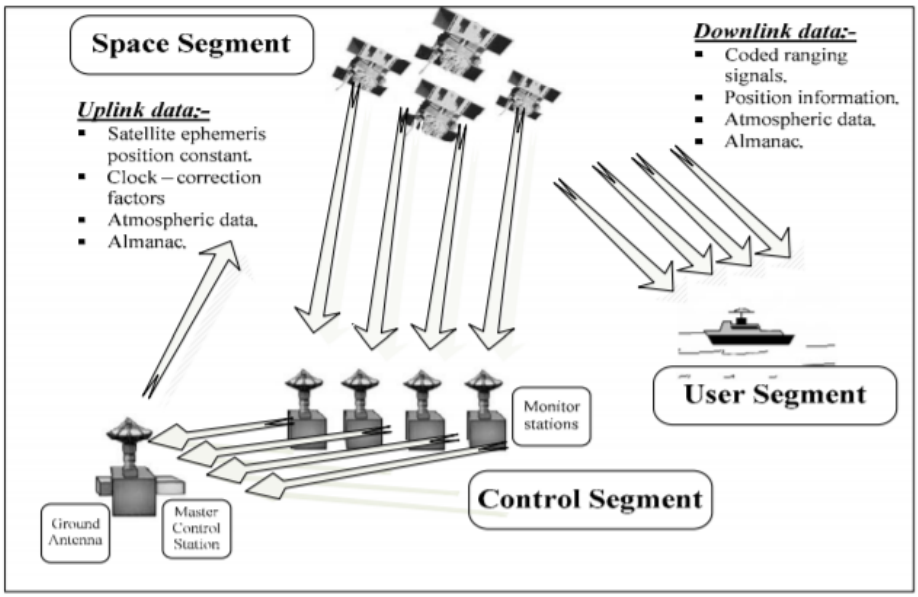

Figure 18. GPS Segments (Aerospace Corporation, 2003)

3.2.4.C Determining the position with the triangulation method

The basic principle inherent in GPS is to determine with the best possible accuracy a point in space, as defined by three coordinates, the geographical latitude and longitude, as well as elevation above sea level. This is done by means of triangulation, that is measurement of triangles. In practice, this involves determining the distances to at least three GPS satellites from the user's GPS receiver. The positions of the satellites in space are known all the time by means of various observational methods and orbital computational methods. These positions are calculated using the concept of triangulation, using the known position of satellites overhead to determine the position of a GPS receiver/antenna pair on Earth (Figure 19).[29]

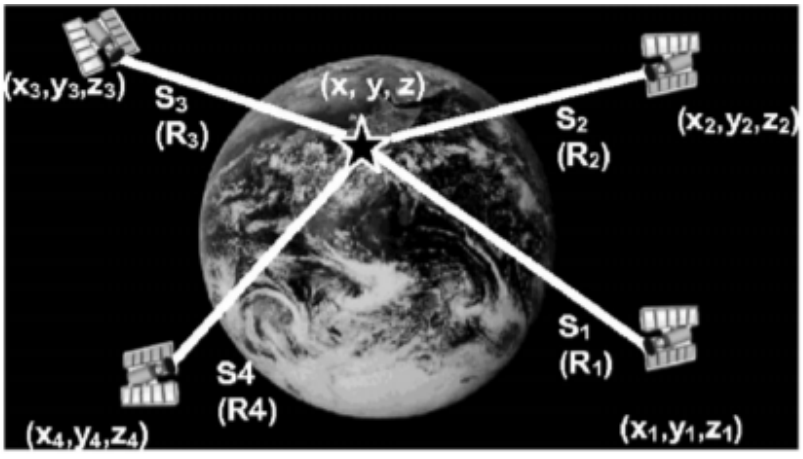

Figure 19. Triangulation Method

Each satellite continuously transmits the current time kept by atomic clocks, as well as information about its current position x_i, y_i, z_i, in its orbital path. The distance, or slant range S_i, of the ith satellite to the unknown position on Earth *x, y ,z* is determined from the travel time of the transmitted electromagnetic signals moving at a rate of 290,000 km/s. This position *x, y ,z* is defined in terms of the World Geodetic System 1984 WGS84 coordinate system, which provides, in Cartesian coordinates, the position on the surface of an ellipsoid representative of the Earth, as described in Leica 1999a. This can then be projected onto a local coordinate system predefined for every region of the United States. For a constellation of N_{sat} satellites, a series of slant ranges can be defined as:

$$S_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \qquad i = 1, 2, \ldots, N_{sat} \tag{1}$$

By inspection, a minimum of three satellites would be required to determine a position estimate, however, this requires very accurate timing mechanisms and nanosecond-level accuracy too expensive for civilian GPS receivers. Instead, less accurate quartz crystal clocks are used to determine pseudo ranges based on uncorrected time values. To account for clock inaccuracies, a time bias b is then introduced into *Eq. 1*, where slant range is replaced more appropriately by pseudorange, R_i , resulting in expanding the number of required satellites to a minimum of four to solve for all the system unknowns.

$$R_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - b \qquad i = 1, 2, \ldots, N_{sat} \tag{2}$$

It should also be noted that although each satellite has its own atomic clock, the time bias for all is the same as it is defined with respect to the same receiver clock. Typically, as more than four satellites are available, an overdetermined set of equations can be generated using *Eq. 2* to obtain an even more accurate position for the receiver. [27]

3.2.4.D The GPS messages format

GPS data is displayed in different message formats over a serial interface. There are standard and non-standard (proprietary) message formats. Nearly all GPS receivers output NMEA data. The NMEA standard is formatted in lines of data called sentences. Each sentence contains various bits of data organized in comma delimited format (i.e. data separated by commas). Here's example NMEA sentences from a GPS receiver with satellite lock (4+ satellites, accurate position): [29][30]

```
$GPRMC,235316.000,A,4003.9040,N,10512.5792,W,0.09,144.75,141112,,*19
$GPGGA,235317.000,4003.9039,N,10512.5793,W,1,08,1.6,1577.9,M,-20.7,M,,0000*5F
$GPGSA,A,3,22,18,21,06,03,09,24,15,,,,,2.5,1.6,1.9*3E
```

Figure 20. GPS message example

The GPGGA sentence represented in Figure 3.2.2.4.1 contains the following:

- Time: 235317.000 is 23:53 and 17.000 seconds in Greenwich mean time
- Longitude: 4003.9040,N is latitude in degrees.decimal minutes, north
- Latitude: 10512.5792,W is longitude in degrees.decimal minutes, west
- Number of satellites seen: 08
- Altitude: 1577 meters

3.2.4.E GPS System Accuracy

Today's GPS receivers are extremely accurate, thanks to their parallel multi-channel design. Our receivers are quick to lock onto satellites when first turned on. They maintain a tracking lock in dense tree-cover or in urban settings with tall buildings. Certain atmospheric factors and other error sources can affect the accuracy of GPS receivers. Garmin GPS receivers are typically accurate to within 10 meters. Accuracy is even better on the water.

Some Garmin GPS receiver accuracy is improved with WAAS (Wide Area Augmentation System). This capability can improve accuracy to better than 3 meters, by providing corrections to the atmosphere. No additional equipment or fees are required to take advantage of WAAS satellites. Users can also get better accuracy with Differential GPS (DGPS), which corrects GPS distances to within an average of 1 to 3 meters. [29][30]

3.2.4.F GPS Signal Error Sources

Factors that can affect GPS signal and accuracy include the following:

- Ionosphere and troposphere delays: Satellite signals slow as they pass through the atmosphere. The GPS system uses a built-in model to partially correct for this type of error;
- Signal multipath: The GPS signal may reflect off objects such as tall buildings or large rock surfaces before it reaches the receiver, which will increase the travel time of the signal and cause errors;

- Receiver clock errors: A receiver's built-in clock may have slight timing errors because it is less accurate than the atomic clocks on GPS satellites;
- Orbital errors: The satellite's reported location may not be accurate;

- Number of satellites visible: The more satellites a GPS receiver can "see," the better the accuracy. When a signal is blocked, you may get position errors or possibly no position reading at all. GPS units typically will not work underwater or underground, but new high-sensitivity receivers are able to track some signals when inside buildings or under tree-cover;
- Satellite geometry/shading: Satellite signals are more effective when satellites are located at wide angles relative to each other, rather than in a line or tight grouping;
- Selective availability: The U.S. Department of Defense once applied Selective Availability (SA) to satellites, making signals less accurate in order to keep 'enemies' from using highly accurate GPS signals. The government turned off SA in May of 2000, which improved the accuracy of civilian GPS receivers.[29][30]

3.3 The GPRS and Wi-fi Services

GPRS (General Packet Radio Service) and Wi-Fi technologies are the ones that ensure the connection between a digital device, such as the smartphones, and the computer network (Internet).

3.3.1 GPRS

General Packet Radio System is also known as GPRS is a third-generation step toward internet access. GPRS is also known as GSM-IP that is a Global-System Mobile Communications Internet Protocol as it keeps the users of this system online, allows to make voice calls, and access internet on-the-go. Even Time-Division Multiple Access (TDMA) users benefit from this system as it provides packet radio access.

GPRS also permits the network operators to execute an Internet Protocol (IP) based core architecture for integrated voice and data applications that will continue to be used and expanded for 3G services.

GPRS supersedes the wired connections, as this system has simplified access to the packet data networks like the internet. The packet radio principle is employed by GPRS to transport user data packets in a structured way between GSM mobile stations and external packet data networks. These packets can be directly routed to the packet switched networks from the GPRS mobile stations.

In the Figure 21 is represented a schematic of the transmission of data packages through GPRS. [31][32][33]

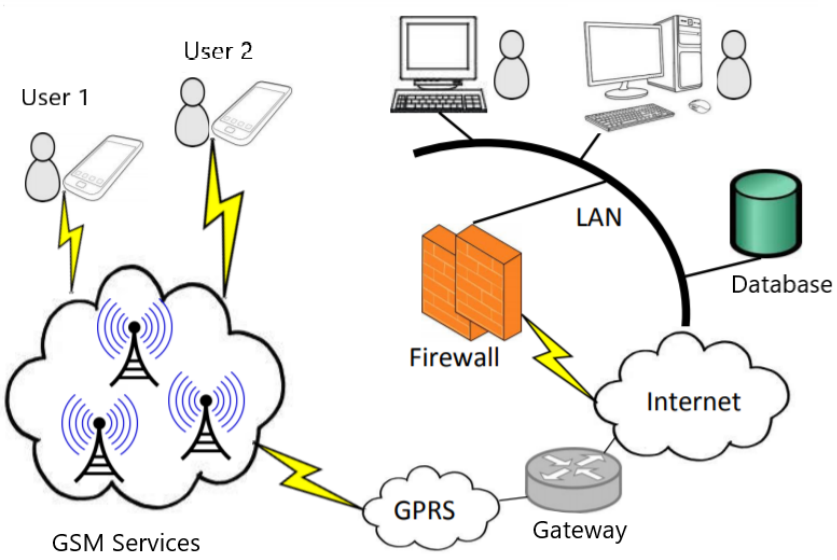

Figure 21. Communication Service through GPRS

3.3.2 Wi-Fi

Wi-Fi is a family of wireless network protocols, based on the IEEE 802.11 family of standards, which are commonly used for local area networking of devices and Internet access. Wi-Fi is a trademark of the non-profit Wi-Fi Alliance, which restricts the use of the term Wi-Fi Certified to products that successfully complete interoperability certification testing. As of 2017, the Wi-Fi Alliance consisted of more than 800 companies from around the world. As of 2019, over 3.05 billion Wi-Fi enabled devices are shipped globally each year. Devices that can use Wi-Fi technologies include personal computer desktops and laptops, smartphones and tablets, smart TVs, printers, smart speakers, cars, and drones.

Wi-Fi networks have no physical wired connection between sender and receiver. Instead, they function by using radio frequency (RF) technology, a frequency within the electromagnetic spectrum associated with radio wave propagation. When an RF current is supplied to an antenna, an electromagnetic field is created that then is able to propagate through space.

The cornerstone of any wireless network is an access point (AP). The primary job of an access point is to broadcast a wireless signal that computers can detect and use to establish a connection to the network. In order to connect to an access point and join a wireless network, computers and devices must be equipped with wireless network adapters. [34][35]

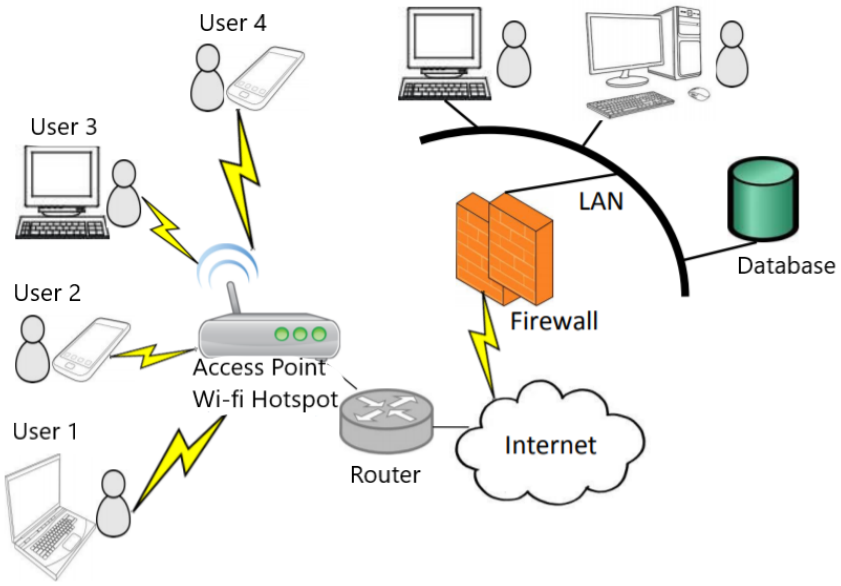

Figure 22. Data communication through Wi-Fi

3.4 The Firebase Database

Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

The purpose of the Realtime Database feature of Firebase is to allow data to be shared between multiple clients, where a "client" can take the form of apps running on Android and iOS mobile devices, or JavaScript running on a web server. The main goal of the system is to provide a secure, reliable and fast way to synchronize data with a minimal amount of coding effort on the part of the developer. The database system is also designed to scale to support millions of users.

The database is referred to as being "realtime" because the speed with which the data is synchronized across clients is probably as close to realtime as is currently achievable (taking into consideration the physical limitation of transmitting data over the internet and wireless connections). The elapsed time while a data change on one client propagates to another is visually imperceptible.

The Realtime Database also provides data persistence by storing data locally, thereby enabling the data to remain accessible even when a device is offline. When connectivity is re-established, the local data is automatically synchronized and merged with the remote data. The Figure 23 presents all the functions offered by Firebase. [36][37][38][39]

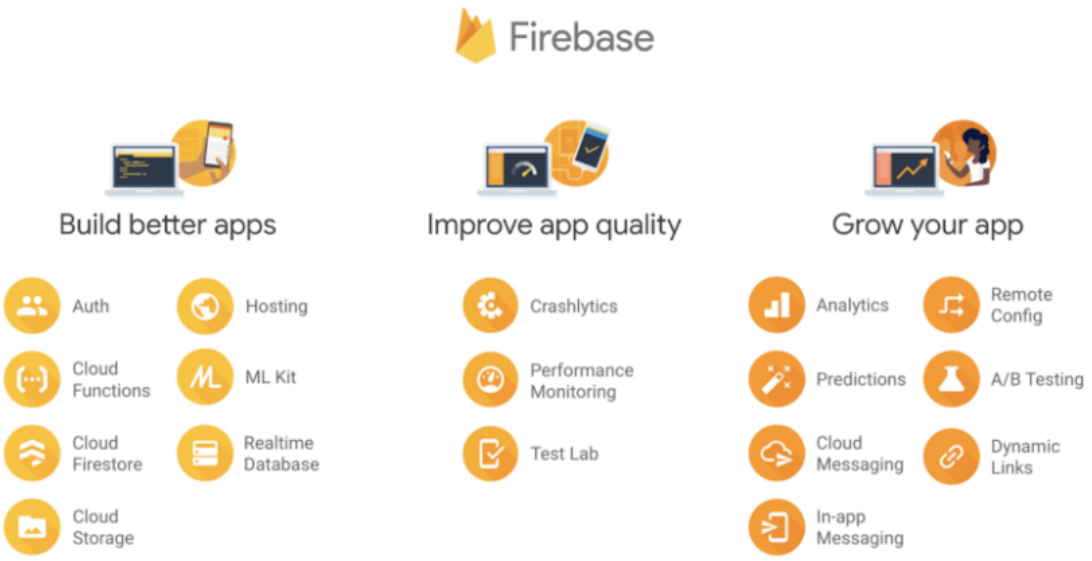

Figure 23. Firebase functions

Firebase uses what is known as a NoSQL database for storing data in a Realtime Database. As the name suggests this means that data is not stored in the tables and rows found in relational database management systems (RDBMS) such as Oracle Database or Microsoft SQL Server. Nor is the data accessed using Structured Query Language (SQL) statements. Instead, the data is stored in the form of a JSON object. JSON is an acronym for JavaScript Object Notation and it defines a syntax used to transmit data in a format that is both lightweight and easy for both humans and software to read, write and understand.

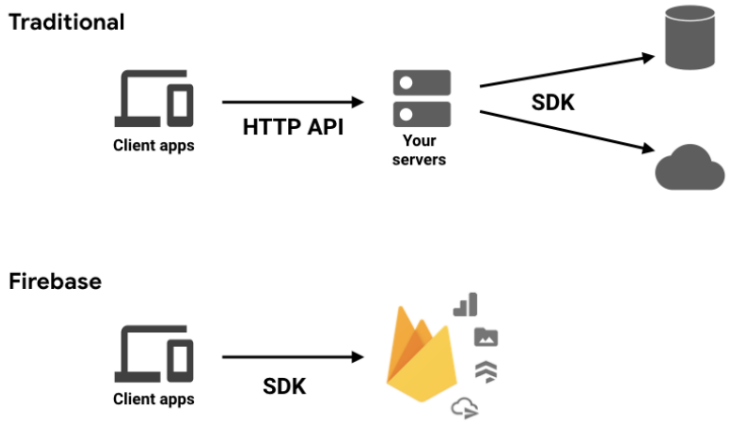

Figure 24. Difference between Firebase and SQL based platforms

JSON objects typically consist of a key/value pair, where the key uniquely identifies the object within the database and the value represents the data that is being stored. Multiple JSON objects are structured in the form of a JSON tree.

Figure 25. Example of values tree in Firebase Database

Chapter 4. Development and Implementation of Fritaxi

4.1 Application Architecture

In this project I will present an Android Application which allows users to register as Drivers and Customers. The users registered as Customers will be able to request a taxi and, via the GPS tracking system and considering both users coordinates, he will be allocated the closest Driver available. The main components of the application are:

- Registering and Logging in as Customers and Drivers;
- Requesting and finding the closest driver available;
- Taxi Driver services: receiving requests, map routing, location tracking;
- Managing the request on both users sides;
- Option to add information (phone number, name);
- Option to log out.

The authentication of the users is made by the Firebase Authentication Service and the Realtime Database Service which will split them in two: Drivers and Customers. Both types of accounts require a smartphone with the Android Operating System and a connection to the internet. The bond between the components mentioned before is better seen in Figure 26.

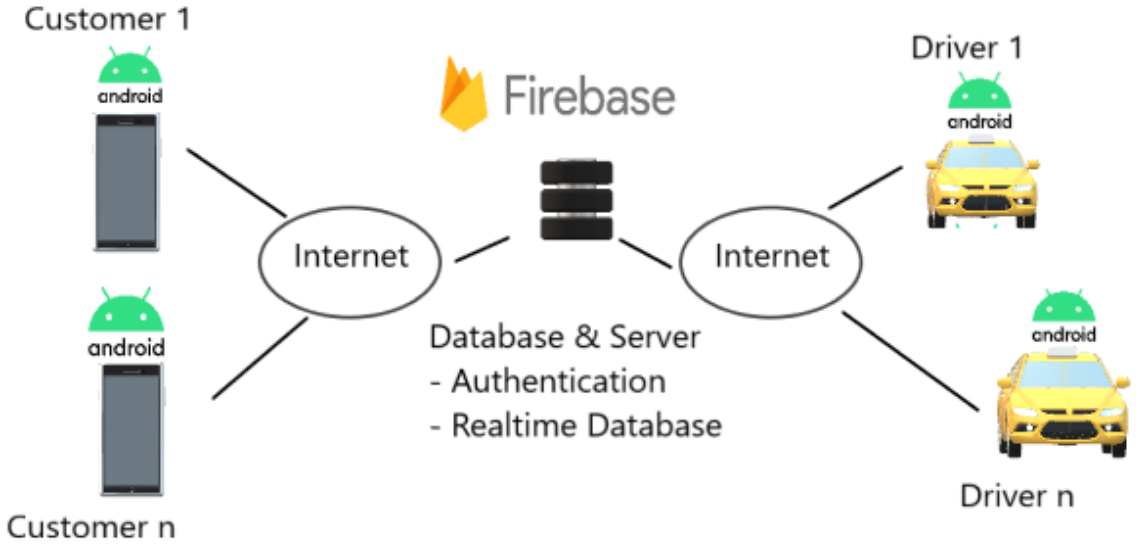

Figure 26. System Architecture

The figure above shows that there are a variety of users, both drivers and customers, who can connect to the application server/ database. The communication between them and the Firebase Database and Server is made via the Internet, which

stores information about customers and drivers, based on which functionalities of the application are fulfilled.

4.2 Use Cases

Use cases are a modeling technique used to render the functionality of an existing system, but in most cases it is used to describe a new system before it is implemented.

Use Case Diagrams are behavioral diagrams that highlight the type of users available in the application and the interactions between them and the system. These are made from the perspective of actors (users): customers and drivers. It is also used to specify how they communicate with each other and the relationships between them and the control part of the system (central server). Figure 27 shows the use case diagram, showing the main operations that the system modules can perform.

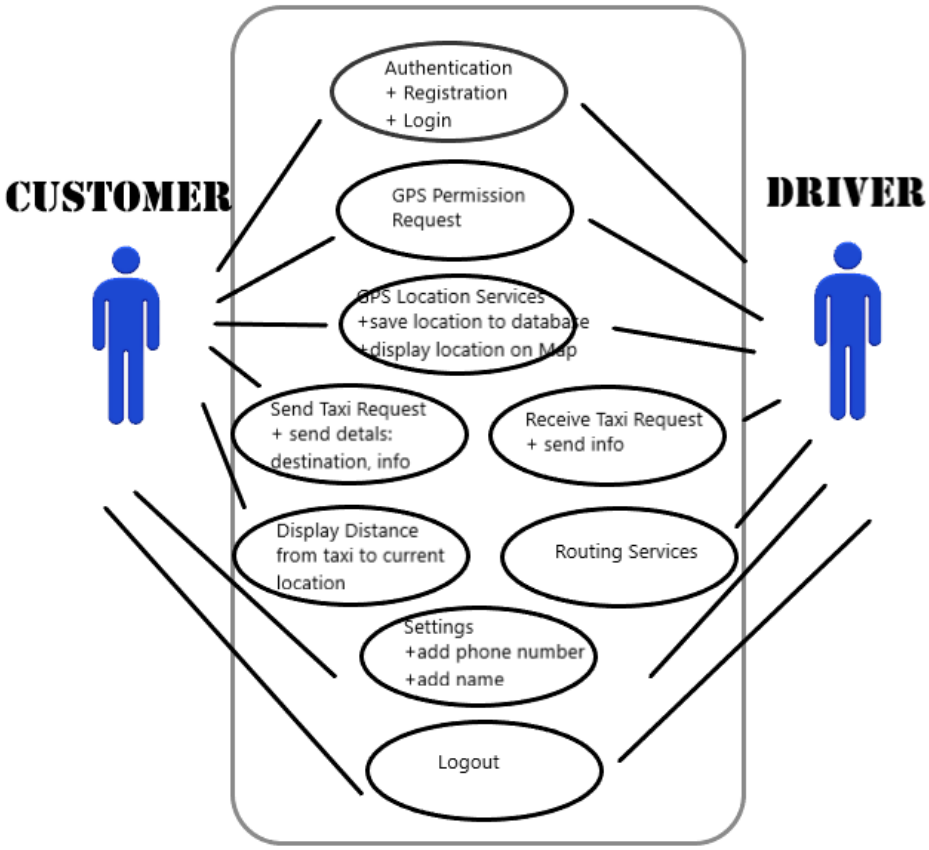

Figure 27. Use Case Diagram

Therefore, after a user successfully logged in as a Customer, a message requesting GPS Permissions will pop up. In other words, while the Google Map is loading, the server will try to access the user's current coordinates in order to pin his position on the map and save it to the database. The next step, after the page is fully loaded, is for the user to access one of the options: Logout, Settings or Request taxi. The option to logout

will take the user to the initial page before the authentication. The second option, Settings, will open a new page which allows the user to introduce his name and phone number, information which will be saved in the database and available for the taxi driver which will receive his taxi request. The last option, request a taxi, is the main function of the application. The user is able to introduce a destination before making the request but it is not mandatory. The driver who will pick his request will be the one closest to the customer's position according to the GPS coordinates saved in the database beforehand.

For the case of the user logged in as a Driver, some functions are similar to the Customer interface, suchs as Loggout, Settings, GPS Permissions. The difference between the accounts stays in the fact that the Driver, when available, will receive the requests from the users. He will also be given routing services from his position to the pick up location and to the destination, if existing.

The Firebase Server is the principal component that ensures the communication and control of the other modules, through the Firebase Database. The main purpose of the usage diagrams is to provide an overview of how the system will be used, to illustrate how the interaction between the system and the actors, providing an overview of the functionalities to be implemented in the newly developed system.

4.3 UML Diagrams

Just as for an architect, sketches represent the detailed design of a building, Unified Modeling Language (UML) diagrams allow software models to be built, observed, and manipulated during analysis and design.

Modeling is a technique used in many disciplines, especially in engineering, as it helps to understand certain more complex processes in the system and allows the evaluation of the project in relation to different criteria, such as flexibility, security, etc. UML is a language that provides a graphical description of the topics covered by OOAD (Object-Oriented Analysis and Design), through which software applications can be documented, specified, built and observed.

After carefully examining the model, any deficiencies or malfunctions of the system may be determined Thus it may be easier to establish possible possibilities for solving problems or improving the state of the system. The ultimate goal of these diagrams is to derive the programming language code. Generating code from UML models is known as forward engineering. There are several types of UML diagrams used to describe software systems, such as: sequential diagram, class diagram, object diagram, collaboration diagram and so on. [40]

4.3.1 Secvential Diagrams

Sequential diagrams are a type of interaction diagram that expresses the relationships and interactions between the processes involved in the system, as well as the order of actions that take place at the system level. Interactions are based on the sequences of messages transmitted from one object to another. This type of diagram also

highlights the lifespan of each process, in accordance with the duration of the messages connecting the processes, from the request to the response.

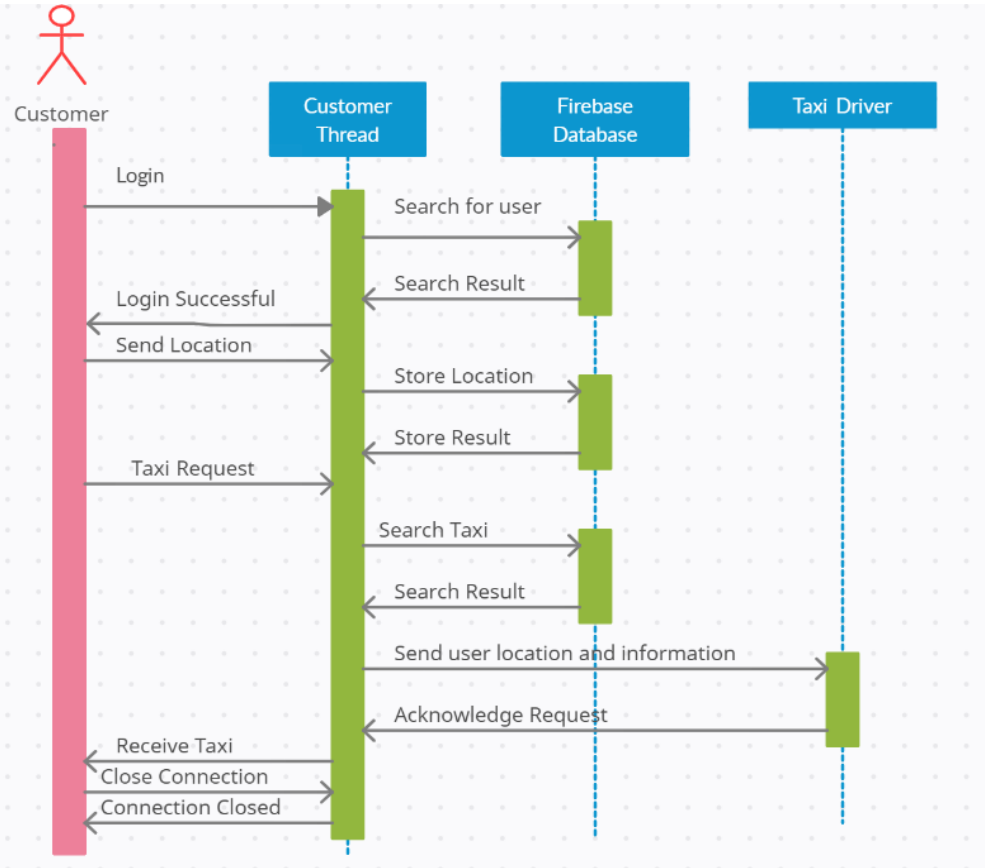

Figure 28. Customer Sequential Diagram

Figure 28 presents the flow of events in the case of the user registered as a Customer. When the user tries to login, his email and password are searched within the database to retrieve the user's details. For the next step, the login must be valid. Afterwards his location is sent and saved in the database in order to be able to show it on the map. When a taxi Request is made, the server will look in the database for available drivers and it will deliver the request to the driver that is the closest to the user's current GPS coordinates. When the request is received, the customer's information will be also delivered to the driver. Now the only remaining part is the closing of the connection.

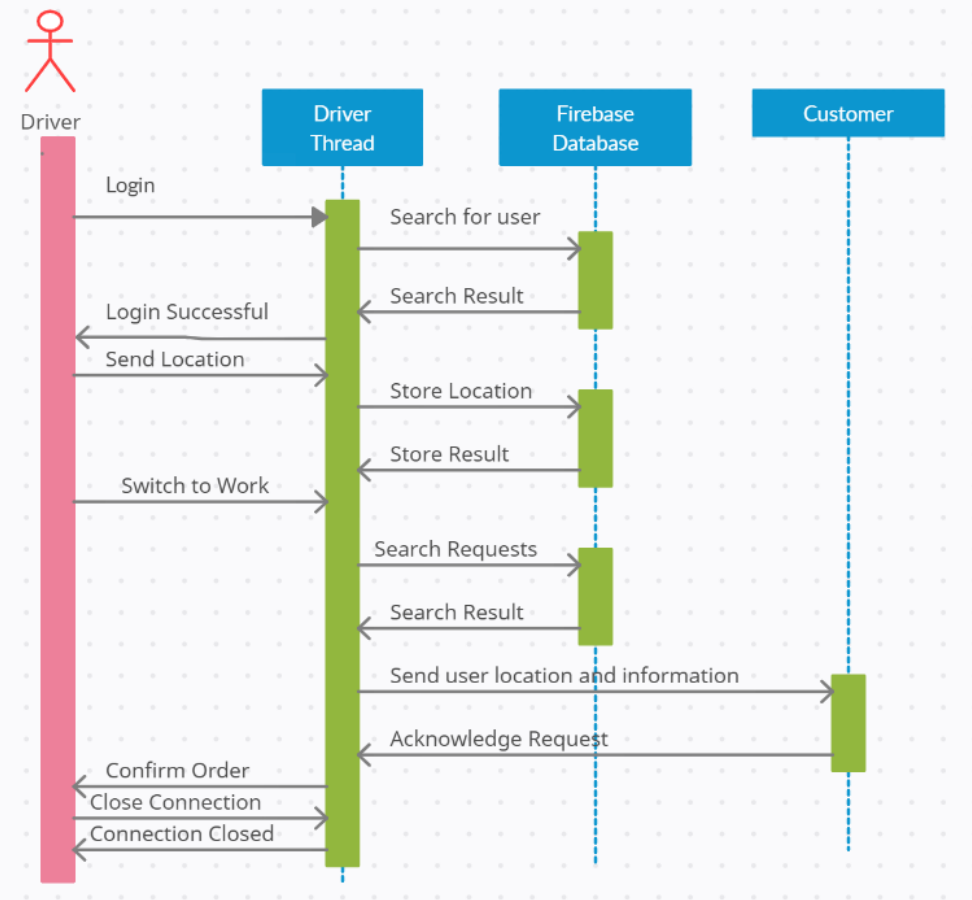

Figure 29. Driver Sequential Diagram

The diagram of the user registered as Driver is represented in Figure 29. Showing a similar flow of events as in the Customer's case, the user has some different functions. After storing his location in the database, the server will be on stand by until the driver decides to start working using the switch. Once he is available for work, the server will search in the database for the requests available and it will retrieve the one with the coordinates closest to the ones of the driver. The request is received, his information from the database is sent to the Customer and the only function left is to close the connection whenever the user is willing to.

4.3.2 Component Diagram

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that are often used to model the static implementation view of a system. [41]

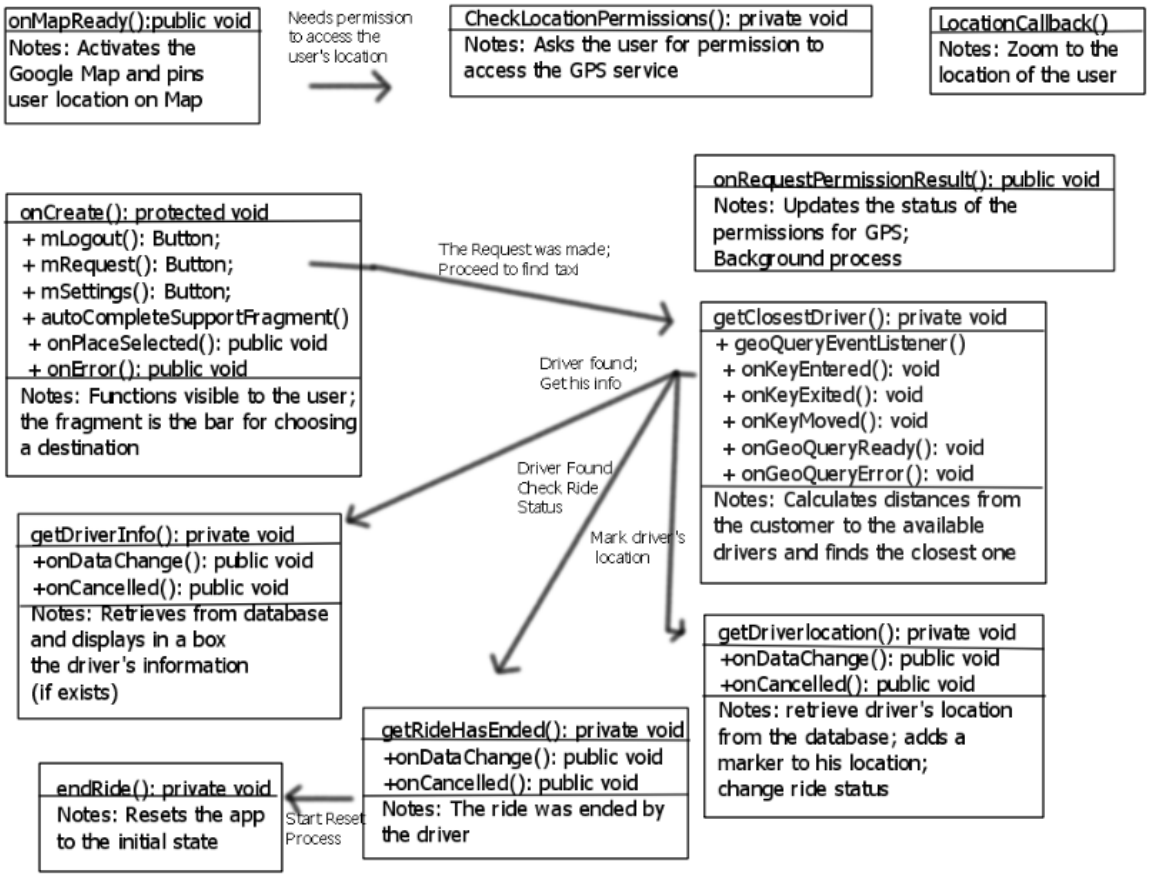

Figure 30. Customer Component Diagram

Since for the implementation of the Fritaxi application I chose to use Firebase, there isn't a bond between classes. Instead there are several functions each implementing a different module. In Figure 30 i represented the functions for the main page of the customer, for the *CustomerMapActivity*. After logging in, the *onMapReady* function automatically starts loading the Google map and it requests the GPS permissions for tracking. The *onCreate* function will consist of the Buttons from the layout and their respective calls. In the case of a request, this function will trigger the function *getClosestDriver*. After retrieving the GPS coordinates from the database, the function will proceed to call *getDriveinfo, getDriverLocation, getRideHasEnded*. The database and the ride process are reseted when the *endRide* function is activated.

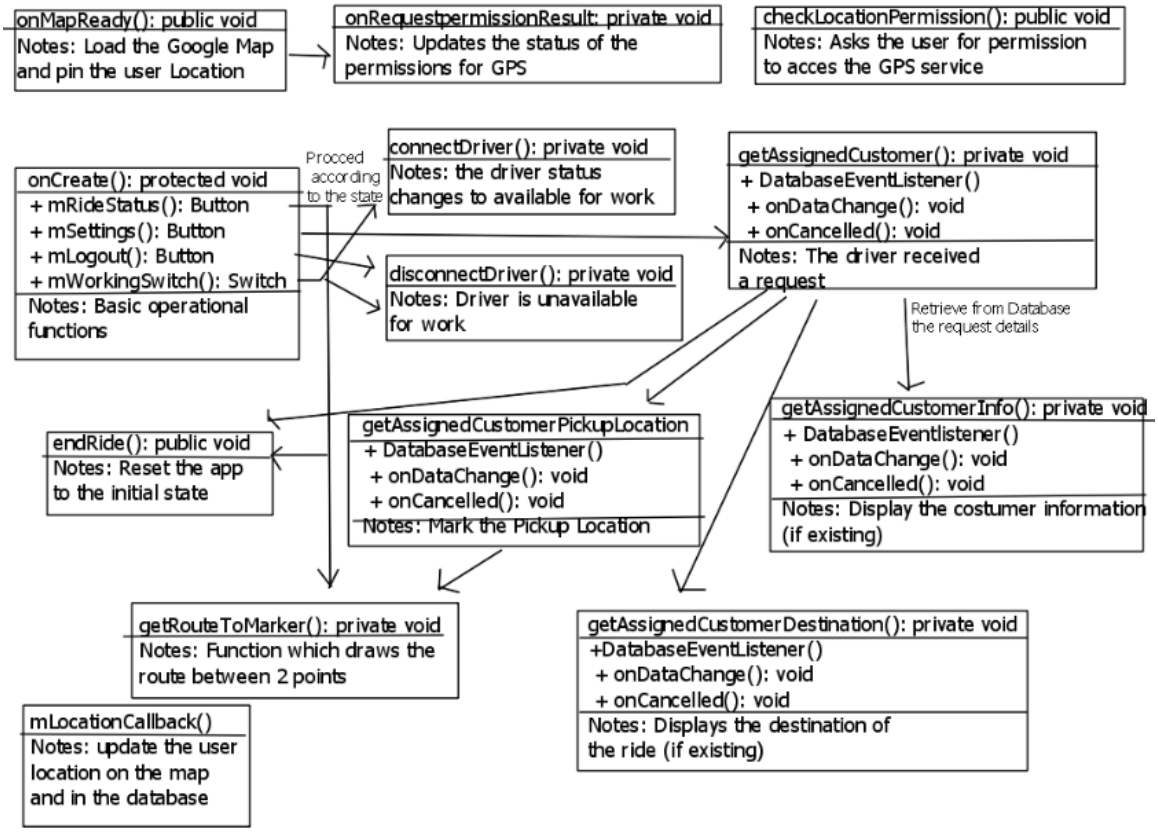

Figure 31. Driver Component Diagram

For the *DriverMapActivity*, the diagram contains a couple more functions as the main management of a ride is done here. As mentioned before, *onMapReady* will automatically start loading the Map and requesting for GPS permissions. *onCreate* provides the actions of the buttons from the layout: *Settings, Layout, Switch. mWorkingSwitch* is the action which adds or removes the user from *DriversAvailable* in the database through *ConnectDriver* and *DisconnectDriver. mRideStatus* represents the component that manages the ride. If the driver receives a request then *getRouteToMarker* (drawing distance between two points) is called and when the ride is ended by the customer, *endRide* is activated. The other functions *getAssignedCustomer, getAssignedCustomerInfo, getAssignedCustomerPickuplocation,* with actions corresponding to their description, are called as seen in Figure 31.

4.4 Implementation

For the development of my Fritaxi application I worked in Android Studio Version 3.5.2, with the SDK Tools 26.1.1 and the Android platform version API 29. The Firebase Database does not have a specific version and it will be presented in a later sub-chapter.

4.4.1 Creating a new Android Studio Project

The first thing when starting a new application is, obviously, creating a new project in chosen software. In this part i will explain with the help of figures how to start a new project in Android Studio.

Step 1: From the android main panel, choose the option Start a new Android Studio project. If there is already a project opened, then a new project can be created from *File - New - New Project.*

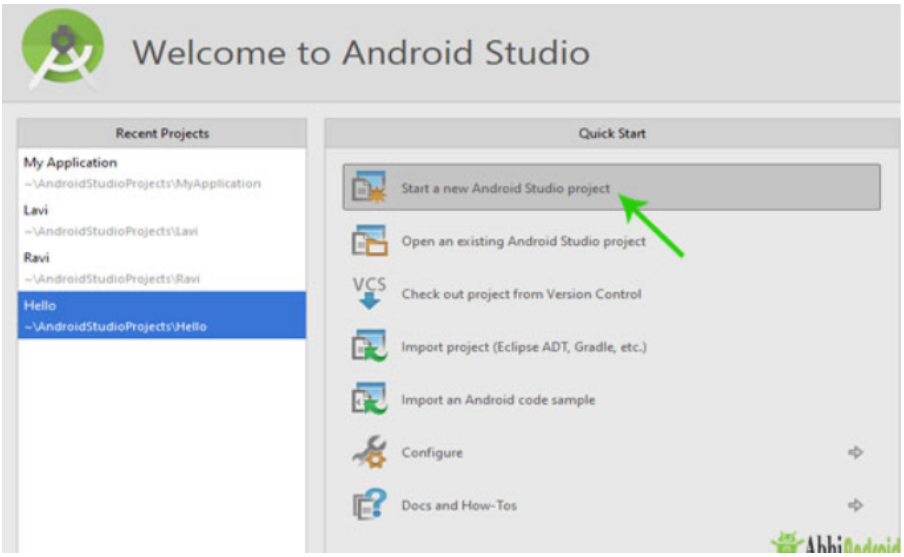

Figure 32. Android Studio New Project

Step 2: The wizard for project configuration has now opened. In this step, introduce a name for the project and a saving location. It is preferable to choose a name relevant for the application. Thus, I wrote *Fritaxi*.

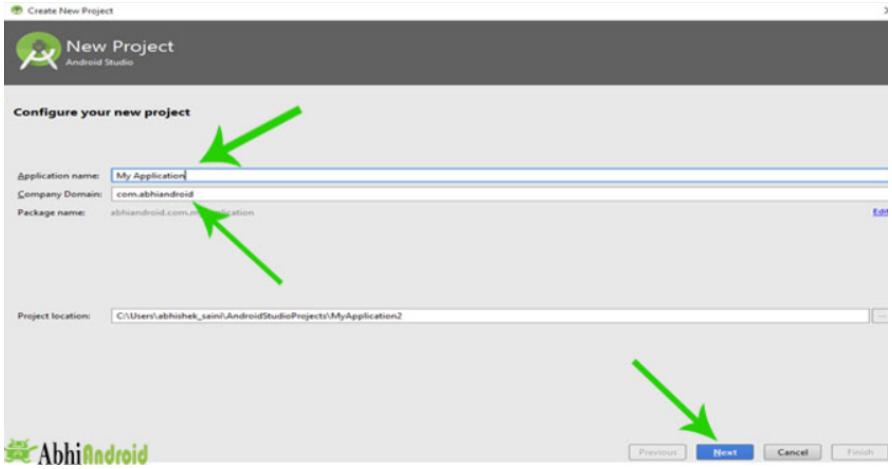

Figure 33. New Project Configuration

Step 3: For this step i did not add a Figure as it is only to choose the devices the application will work on. To be more precise, the window will present the following options to choose from: Phone and Tablet, Wear, TV, Android Auto, Glass. An important note is to select the right minimum SDK. In my case I used version 16 as a minimum.

Step 4: The window that opens after Step 3 is shown in Figure 33. For Fritaxi I started with an *Empty Activity* which I will edit to be the homepage, the first page that opens when the application loads on a phone.

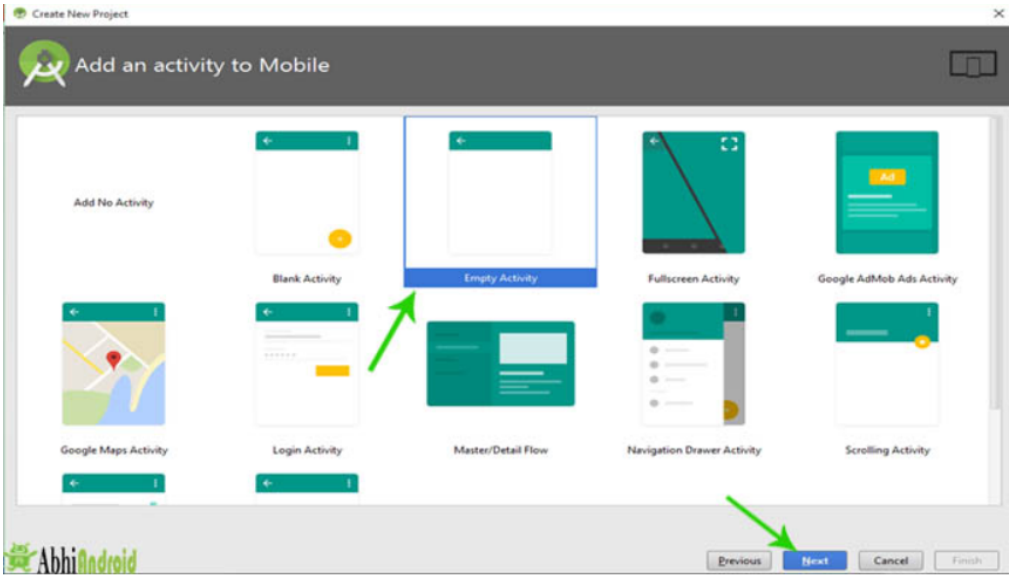

Figure 34. New Activity

Step 5: Lastly is the customization of the first Activity. For most projects, this Activity remains under the name MainActivity and so it is in my project.

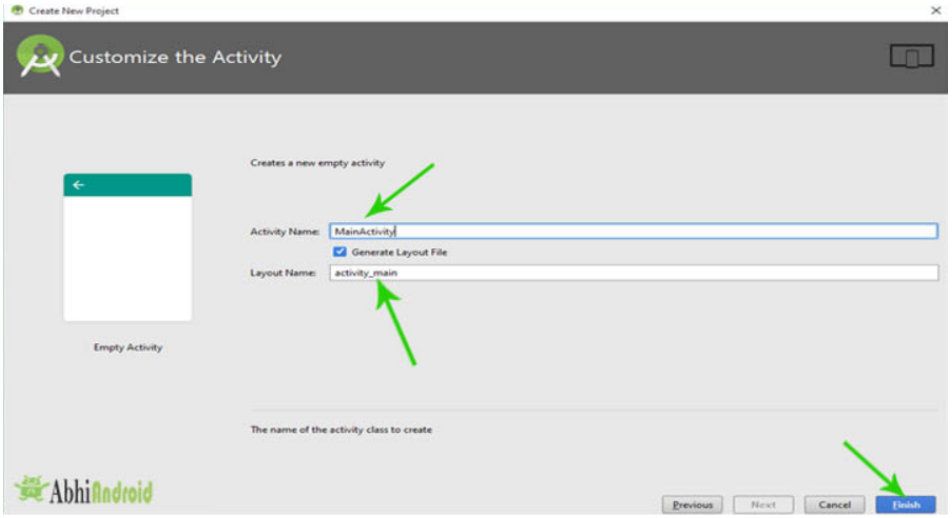

Figure 35. Activity Customization

4.4.2 Connecting to Firebase database

Before starting to implement the code, I made the connection to Firebase Database. To access the platform, one has to go to the website firebase.google.com and enter a valid google account to login. By going to the *Console*, the page with projects will open as shown in Figure 36.

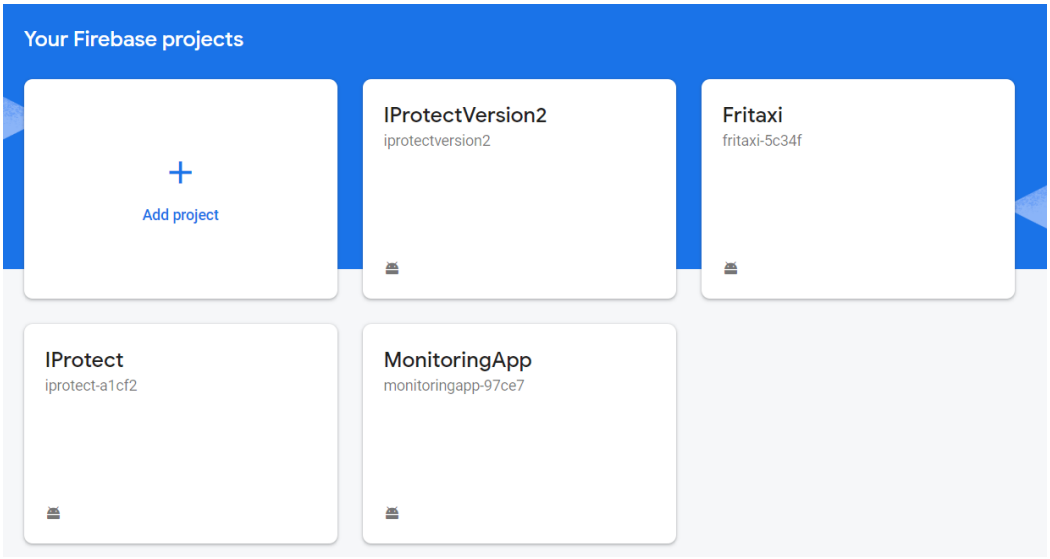

Figure 36. New Firebase project

There are a total of three steps when creating a new project: setting up a name for the project, enabling or disabling Google Analytics and selecting an account. For Fritaxi the name of the database is IProtectVersion2, unrelated to the current project as my initial idea for the final project was quite different. The platform doesn't allow to change the name of the database after it was created but it allows the project as a whole to be renamed.

The initial page that shows is the project overview. From the left side bar, the *Realtime Database* option is the one for creating the database. Figure 37 that can be seen below shows both the left side bar with options and the database for Fritaxi which I created at first. This is represented by a main child, *Users,* and two children *Drivers* and *Customers*. Therefore the user accounts which are created in the application will be separated in 2 groups.

As the application runs, the database will change accordingly by adding and deleting temporary children from the tree.

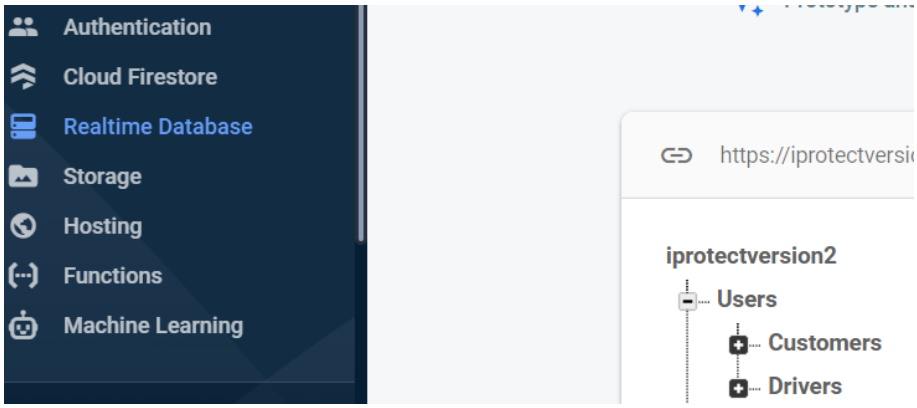

Figure 37. Firebase Realtime Database

Since I already presented a part of the database, I will also note that setting the database rules is essential. Without this, the application will not be able to retrieve and to write information in the database. Figure 38 presents the necessary rules. *Read* represents the ability to retrieve information from the database and *write* the ability to add information in the database from Android Studio. The other function, *geofire*, is an open-source library for Android that allows users to store and query a set of keys based on their geographic location and it only works with the rules set as in the Figure below.

```
{
  /* Visit https://firebase.goog
  "rules": {
    ".read": true,
    ".write": true,
       "geofire": {
          ".read": "true",
           ".indexOn": ["g"]
    }
  }
}
```

Figure 38. Database rules

Considering that the database is ready for the project, the next step is to connect the Android Project to Firebase. In the upper menu bar we select *Tools* - *Firebase*. This action will load a Firebase pop up with many options such as Analytics, Cloud Messaging, Realtime Database, Authentication, Storage and so on. The list can also be seen in Figure 39.

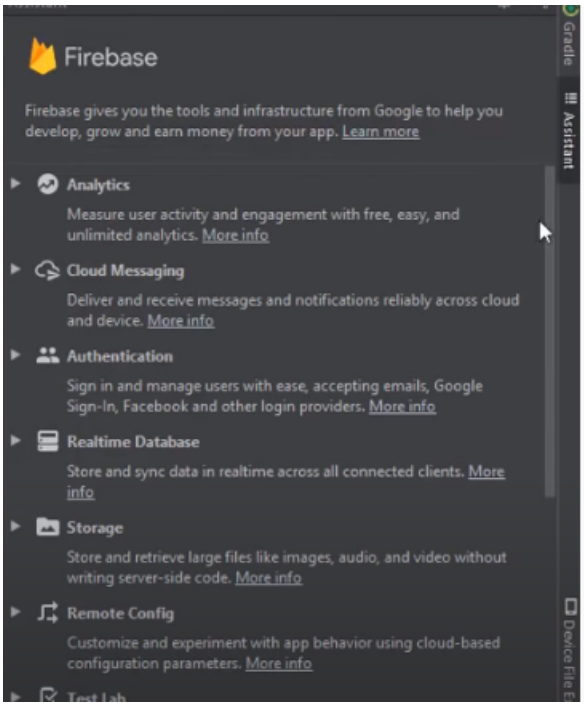

Figure 39. Connecting to Firebase from Android Studio

From the options in Figure 39 we select Realtime Database and the option Save and Retrieve Data. A new set of options will be displayed as shown in Figure 40. Going in order, the first step is to click the option Connect to Firebase.

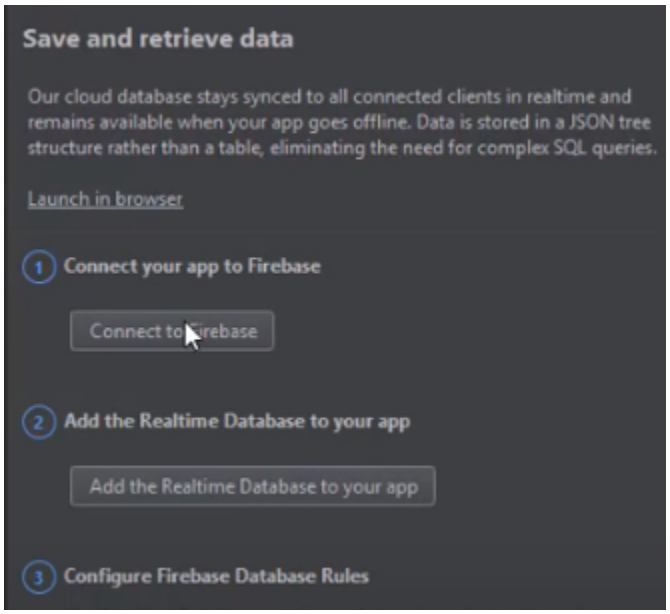

Figure 40. Connecting to Firebase

The default browser which is selected for the computer will open a web page for the user to connect to their google account. As i mentioned before, the platform Firebase requires a google account to open. After logging in to the account, there will be a pop up asking to allow Firebase to connect to Android Studio. Since the first step is done, next we have to click add the Firebase database to your app. The window that appears is presented in Figure 41. There are two possibilities: creating a new Firebase project or selecting an already existing project. In my case the database is made so i choose my project from the list. After selecting the country/region, click on Connect to Firebase and accept changes. The application will now synchronize with the database from Firebase.

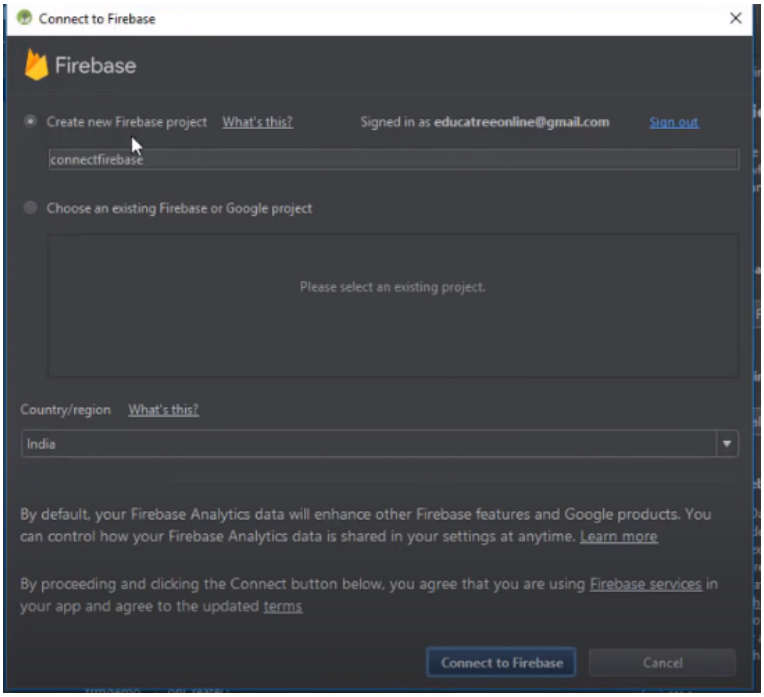

Figure 41. Connecting to the database

4.4.3 Petri Nets Representation

A Petri net, also known as a place/transition (PT) net, is one of several mathematical modeling languages for the description of distributed systems. It is a class of discrete event dynamic systems. These nets have two types of components: positions and transitions (positions = factors, transitions = processes). The dynamics of a Petri net is a sequence of transition "firing". It is possible to have multiple arown from the transitions and from the states.

Because Fritaxi is a client - server type of application, I made two petri nets for a better and easier explanation of the functions of the users registered as Drivers and Customers.

In Figure 42, I drew the Petri net for the driver account.

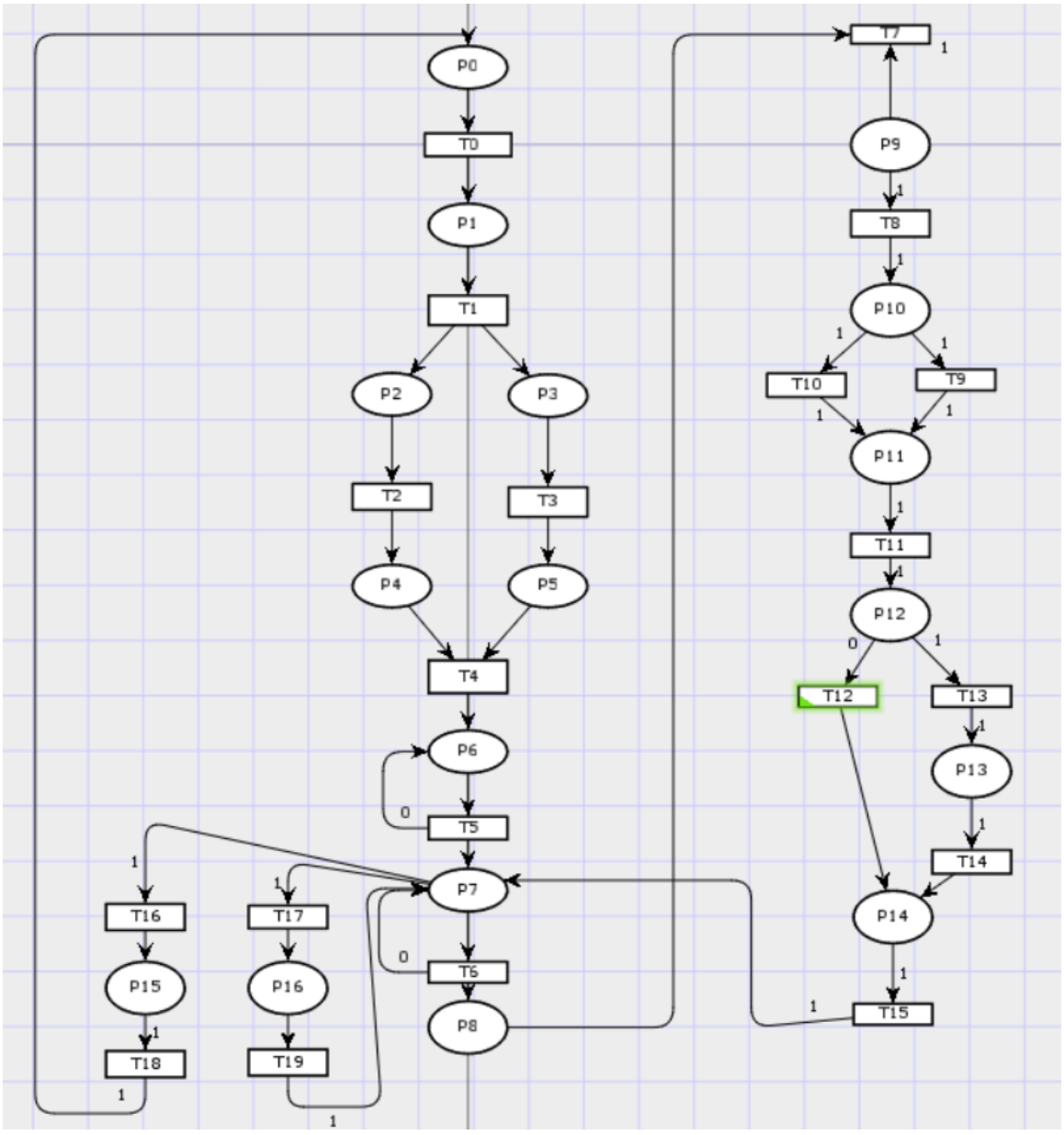

Figure 42. Customer Petri Net

In the case of a driver account, there are a total of 16 states (denoted with P0,..,P16) and 19 transitions (denoted with T0,..,T19). Table 1 presents an overview and explanation of the Petri Net. In other words, the Petri net is showing the flow of the events which occur from the moment that a user opens the application until the moment he leaves the application or logs out.

P0	Initial state; waiting to connect	T0	Processing connection
P1	Driver Authentication Page	T1	Confirm Connection
P2	Waiting for driver's registration	T2	L - search driver in database; if found continue
P3	Waiting for driver's login	T3	R - add driver's information in database
P4	L - account found	T4	Load the Google Map
P5	R - new account made	T5	Request GPS permission
P6	Waiting for Map connection	T6	Driver switch: 0: Stay in State P7 1: Add the user under DriversAvailable in Database
P7	GPS permissions granted	T7	Search for Requests in database
P8	Waiting for Database connection	T8	Request found: Add Request to driver in database, move driver to driversWorking
P9	Waiting for taxi Request	T9	Show route from driver to pickupLocation
P10	Taxi order received	T10	Send informations to Customer
P11	Waiting for driver to arrive	T11	Customer is picked up
P12	Waiting for destination	T12	Customer did not send a destination
P13	Destination received	T13	Customer sent destination at request
P14	Arrived at destination	T14	Draw Route on map to Destination
P15	Logout Button Pressed	T15	End the ride/request; Reset database
P16	Settings Button Pressed	T16	Prepare to Disconnect
		T17	Load Settings Page
		T18	Disconnect Driver from app
		T19	Add user's information to database

**R - registration, L - Logout

Table 1. Driver Petri Net Explanation

An important note, which is not represented on the net, is that the driver and the customer are both able to cancel/end the ride at any moment from state P10 to P14. If this happens, the user is sent to state P7 automatically.

For the users registered as customers, the situation stands somewhat different. The Petri Net is formed by 15 states (denoted with P0,..,P15) and 16 transitions (denoted with T0,...,T16). Figure 43 demonstrates how the net looks. Up until the state P6, it is the same as in the case of the driver account because it represents the part of registering, login, map loading and GPS permission request.

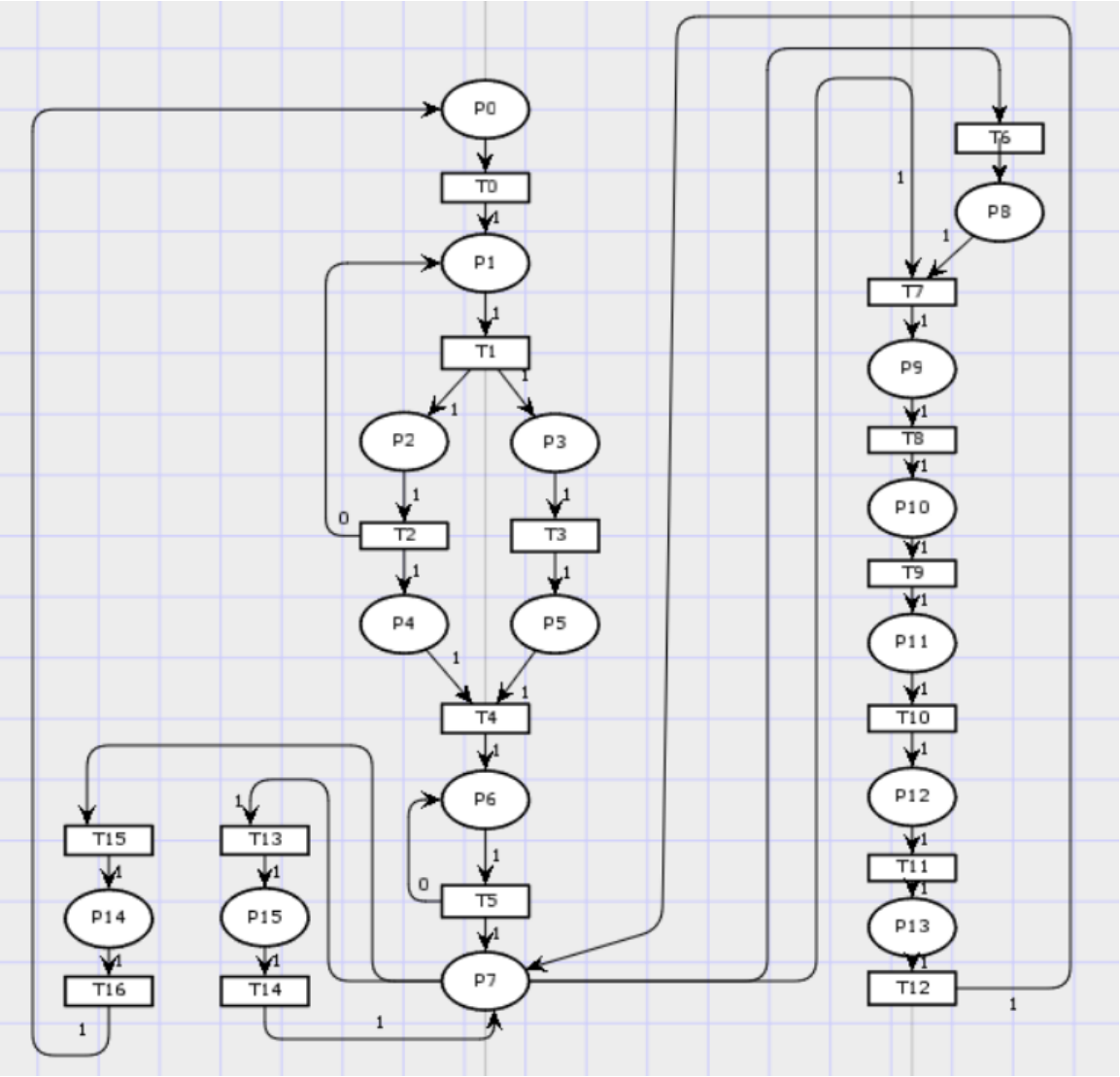

Figure 43. Customer Petri Net

The significance of the states and transition is described in Table 2.

P0	Initial state; waiting connection	T0	Processing connection
P1	Customer Authentication	T1	Confirm connection
P2	Waiting for login	T2	L-Searching for customer account; Continue if found in database
P3	Waiting for registration	T3	R-Add user to Database
P4	Account exists	T4	Load Google Map
P5	New account made	T5	Request GPS permissions ; If denied return
P6	Waiting for the Map	T6	Introduce Destination to database
P7	GPS permissions granted	T7	Add Request to database
P8	Waiting for user to do something	T8	If driver found proceed and delete the request from database
P9	Destination was introduced	T9	Calculate and display distance to driver;Display Driver's information
P10	Waiting for Driver to take request	T10	Driver arrives; display message
P11	Waiting for the driver's arrival	T11	Display option to cancel ride
P12	Customer goes to the taxi	T12	The driver ends the ride
P13	Arriving to the destination	T13	Open Settings Page
P14	Logout Button Pressed	T14	Add user's information to database
P15	Settings Button pressed	T15	Prepare to disconnect
		T16	Disconnect user from application

**R - registration, L - Logout

Table 2. Customer Petri Net Explanation

Similar to the flow of functions from the driver net, the ride is also possible to be canceled/ended at any moment from state P10 to P13. In this situation the user will be automatically returned to the state P8.

The implementation of the functions presented in the Petri Nets will be further detailed in the coming sub-chapters.

4.4.4 Login and Registration

When a user opens the application, a page showing two options will pop up. These options are *Driver* and *Customer.* Each button will take the user to a registration and login form. There must be two different forms because the users accounts will be saved in the database as *Drivers* and *Customers* under the child *Users*. Therefore, in this subchapter i will present the following activities:

- MainActivity;
- CustomerloginActivity;
- DriverLoginActivity.

The layout of *MainActivity* is represented by the file *activity_main.xml*. This file contains a palette with several options such as: buttons, textview, imageview, recyclerview, switch and so on. In order to obtain full control over the layout, I used a *RelativeLayout* instead of *ConstrainLayout.* The type can be changed in the text tab at the beginning of the code. Furthermore i added a background image using the following line: "*android:background="@drawable/frexi2*". *Drawable* represents the folder where the images have to be saved and *frexi2* is the name of the file. I used this method through the application for design purposes.

In the *activity_main* file, I added two buttons with the drag and drop method. Each button has a designated id, in this case they are: *Customer_btn* and *Driver_btn*. The functionality of the buttons is made through the id in the activity. Figure 44 shows an example for the case of the button for drivers.

```
Driver_BTN = (Button) findViewById(R.id.Driver_btn);

Driver_BTN.setOnClickListener((v) → {
        Intent intent = new Intent( packageContext: MainActivity.this, DriverLoginActivity.class);
        startActivity(intent);
        finish();
        return;
});
```

Figure 44. Driver button connection

In the figure above, *Driver_BTN* is a variable declared as a *private Button* and it is connected to the corresponding id of the button from the layout, *Driver_btn*. The method *setOnClickListener* is activated through the action of a click. Then by using an intent the user is sent from the current *MainActivity* to *DriverLoginActivity*. The intent acts similar to a glue and it binds the two activities together. Following this example, the *Customer_BTN* is attached to *Customer_btn* and the listener will send the user to *CustomerLoginActivity* if the user desires to register as a *Customer*.

The files containing the layouts for the login activities are *activity_customer_login.xml* and *activity_driver_login.xml*. The design is made with the same approach as for the previous layouts. The difference is that in this case I used two edit text fields, two buttons and one text view. Their meaning is in order email, password, login, register and create an account. At first, the button to register will not be visible but it will become visible after the user clicks on create an account. This process is implemented in *CustomerLoginActivity.java* with the function *setVisibility(View.Visible/Invisible)*.

The first thing I did in the login activity was to retrieve the current user id from the database. The process is done with a Firebase authentication state listener which is doing exactly what it says, verifies the current state of the user id in the database when the application is started. In other words, if the user left the application without logging out, he will be automatically redirected to the next activity. If the user logged out before or it is his first time opening the application, then nothing will happen.

For the registration part, i added texts and loading bars for a better using experience. The user, driver or customer, will receive messages such as "Please write email" and "Sign up error". The code used for adding a new customer user account in the database is:

```
String user_id = mAuth.getCurrentUser().getUid();
DatabaseReference current_user_db =
FirebaseDatabase.getInstance().getReference().child("Users").child("Customers").child(
user_id);
current_user_db.setValue(true);
```

A very important mention is that the registration is made through the mAuth.CreateuserWithEmailAndPassword() method while the login is made through the mAuth.signInWithEmailAndPassword() method. Both methods are provided by Firebase for authentication.

The full list of user accounts made in the application can be found in the Firebase Authentication page and it will look similar to Figure 45.

Figure 45. Firebase Authentication Page

4.4.5 Google Map API Setup

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to communicate with each other.

To add an API to an Android Studio Project, I accessed the Google Cloud Platform Console. There I created a project and from the menu on the left side I chose *API Manager - Dashboard*. The APIs that I enabled are: Google Maps Android API and Google Places API for Android. Afterwards, to connect this project to Android Studio, i went to *Menu - API Manager - Create Credentials*. The window which appears is presented in Figure 46.

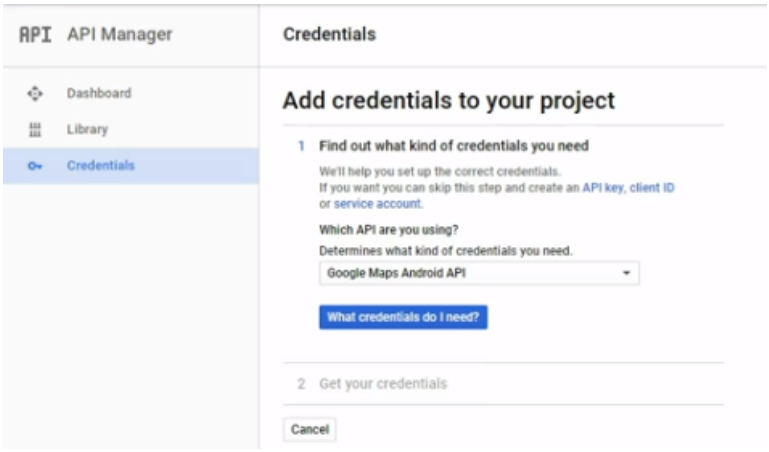

Figure 46. Getting the Google Map API Key

The credentials represent the API Key which has to be added to the project for the connection between the google cloud platform project and the Android Studio project. Thus the next step is to go to the application and add a new activity of the type Google Maps Activity. For Fritaxi I made two such activities, *DriverMapActivity* and *CustumerMap2Activity*. After a map activity is created, the platform opens a file like the one in Figure 47. At the bottom of the file, the *YOUR_KEY_HERE* value has to be replaced with the string obtained at the previous step.

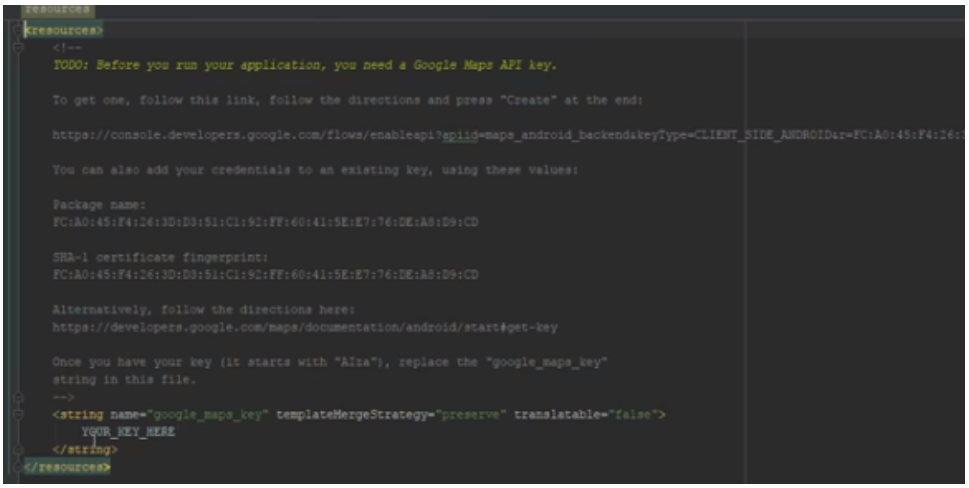

Figure 47. Adding the API Key

4.4.6 Google Map Functions

For the layout of the activities that contain a map, I applied a frame layout. The map is integrated through a fragment containing a link to the map such as:

android:name="com.google.android.gms.maps.SupportMapFragment"

To start loading the map, in the *onCreate* function of the *DriverMapActivity* and *CustomerMap2* activities, i used the following line:

```
mapFragment = (SupportMapFragment)
    getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
```

The map requires time to load. Therefore it is not possible to add functions for the map in *onCreate* where it was called to start loading. The behavior of the map is managed by the function *onMapReady*. Just like the name says, when the map finishes loading and it is ready for use, one can add more tasks for the map. The code that displays the current user's position on the map is the following:

```
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap();
mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(1000);
mLocationRequest.setFastestInterval(1000);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
if(android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
{ if(ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED){} else{ checkLocationPermission(); }}}
```

In the code sequence from above, the *mLocationRequest* variable is the one which retrieves the GPS coordinates of the current user position and displays it on the map. The interval and the accuracy are meant to update the coordinates every second (1000) and I set the accuracy high since this project is not made to be uploaded on Google Play. The drawback when using a high accuracy is the memory used.

In order to display the user's position on the map, the application will request access to the GPS Service of the smartphone. For this part I made two functions *CheckLocationpermission()* and o*nRequestPermissionResult()*. The objective of these functions is to always keep in check if the location permissions are enabled or not.

To zoom in to the user's position the later sequences are used:

```
mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
mMap.animateCamera(CameraUpdateFactory.zoomTo(11));
```

Another important function of the map is storing the positions in the database. When a taxi request is made the application will display the customer's position for the driver and vice versa. In order to store GPS coordinates to the database I utilized GeoFire. GeoFire is an open-source library for Android that allows users to store and query a set of keys based on their geographic location. For example, the next code sequence is used to add a the location to a request:

```
DatabaseReference ref =
FirebaseDatabase.getInstance().getReference("customerRequest");
GeoFire geoFire = new GeoFire(ref);
```

In the example, *ref* represents a reference to the child customerRequest from the database (this child will be explained in the ride management sub chapter). The second line adds the coordinates to the reference made. In order to delete coordinates from the database I used: *geoFire.removelocation()*.

Another function which can be seen multiple times in my project is adding a marker to pin the user's position on the google map. The marker is similar to the one found in the Google Maps application and i used it so the customer will see the driver's location and vice versa. For example, the following sequence adds a marker which displays the position for pick up:

```
pickupMarker = mMap.addMarker(new MarkerOptions().
        position(pickupLocation).title("Pickup Here"));
```

4.4.7 Implementing customer's ride functions

The management of the ride for a customer account is made in the *CustomerMap2* activity and the layout is represented by the file *activity_customerr_map2.xml*.

The layout is composed of a settings button, logout button, cardview instance for destination, the map fragment, two textviews to display the driver's information and a button to request a taxi driver. The cardView is a fragment which supports a bar similar to google. This allows the user to introduce destinations such as a different city or street. The bar is implemented through an autocomplete support fragment and the destination introduced is converted to geographical coordinates with the following sequence:

```
destination = place.getName().toString();
destinationLatLng = place.getLatLng();
```

The function for request is represented by *mRequest*. When a user is made, the application adds a new child to the tree from the database with the name *customerRequests*. Under this child will be displayed the id's and geographical coordinates of all the customers who requested a taxi driver. While the customer is waiting, the button for request will display the message "Getting your driver...". The function *getClosestDriver()* will take the request and proceed to the next step, finding the closest taxi driver. In order to find a driver, I made an instance to the driversAvailable child from the database. Under this child are all the id's of the available taxi drivers and their respective geographical coordinates. When the function starts there is a radius of 1 supposingly kilometers, a value which increases as the function keeps searching for a driver. After a driver is found the *customerRequest* from the database is moved under *Users - Drivers - DriverId*. In this way the application will exit from the loop of searching a driver for the request. While the function *getDriverLocation()*, *getDriverInfo()* and *getRideHasEnded()* are being called, the text displayed is "Looking for driver location...". The *getDriverLocation()* function is calculating and displaying the distance from the driver to the customer. When this distance is smaller than 100, the application will show the message "Driver is here".The getDriverInfo() function is retrieving the driver's information from the database (name, phone number) and it displays it as a list on the screen. Lastly the *getRideHasEnded()* is making a callback to the *endRide()* function and represents the action of canceling/ending the ride by one of the users.

The last part of managing the ride for the customer is ending the ride. This process is done by the function *endRide()*. The found driver's id is removed from the database, all the markers and listeners are removed, the panel with the driver's information is removed and the map is reseted. Therefore, the customer user will be able to make a new request and to start a new ride process.

4.4.8 Implementing driver's ride functions

In order to improve the quality of the user experience while connecting as a driver, I implemented a switch which is managing the driver position in the database. If the switch is on then he is working and his id is added to the database child *driversAvailable*. For this process I made two functions *ConnectDriver()* and *DisconnectDriver()*. When the driver is connected, the application will display his

position on the map and it will start waiting for requests within his area. On the other hand, the disconnect function will remove the driver's id from the database child *driversAvailable*.

The function that is searching and allocating requests to the drivers is *getAssignedCustomer()*. Here I retrieved from the database the id of the customer which made a request. Further I called the other functions which manage the flow of the ride. As in the case of the customer, the *endRide()* function is also called in case any of the users cancels or ends the ride.

The location for the pick up of the customer is obtained in the function *getAssignedCustomerPickupLocation()* by making a reference to the customerRequest child of the database. After the coordinates are reclaimed, the application will add a marker to the position of the customer and using the function *getRouteToMarker()* i drew on the map the route to get to that position. The personal data of the customer, his name and phone number, is recovered and displayed through the function *getAssignedCustomerinfo()*. Since the destination is optional i made a specific function for it named *getAssignedCustomerDestinatio()*. The coordinates of the destination are saved in the database under the child *customerRequest*. In the case of an existing destination for a ride, after the pick up operation, the application will proceed to add a marker for that location and to show the route up to that point.

In order to draw the distance between two points on a map, I utilized a library named jd-alexander. The main part is represented by the sequence:

```
private void getRouteToMarker(LatLng pickupLatLng) {
        if (pickupLatLng != null && mLastLocation != null){
                Routing routing = new Routing.Builder()
                        .key("AIzaSyBG0LGAhEgk2rH3qmRhkNW8O67Fcy8gyc8")
                        .travelMode(AbstractRouting.TravelMode.DRIVING)
                        .withListener(this)
                        .alternativeRoutes(false)
                        .waypoints(new LatLng(mLastLocation.getLatitude(),
                           mLastLocation.getLongitude()), pickupLatLng)
                        .build();
                routing.execute(); }}
```

The exemple above is drawing the route between the last saved position of the driver to the position for pick up of the customer. The string noted is key represents the API Key which makes the connection to the map routing service. The actual drawing of the line is done with polylines in the function *onRoutingSuccsess()*. I also added a message in the case of routing failure.

The status of the ride is managed in *onCreate* through *mRideStatus()*. Here I used the case function to sort the case of request with destination and the case of a sudden ending or cancelation of the ride while the driver is going to the customer pick up location. When the ride has ended, the function *endRide()* deletes all the markers,

polylines, removes the request from the database, removes the panel with the customer's information and the user will be able to receive new requests.

4.4.9 Logout and Settings Functions

For the *CustomerMap2* and *DrivermapActivity* activities I mentioned that i introduced two buttons for logout and settings.

Primarily the sign out operation is realized using a function from the Firebase library, more exactly the sequence: *FirebaseAuth.getInstance().signOut()*. After this function i used an intent to redirect the user to the *MainActivity* which represents the homepage of the application.

Since there are two types of accounts for the application, I had two use two different activities for the settings action as the information will be saved in two different categories. These activities are *CustomerSettingsActivity* and *DriverSettingsActivity*. The layout is simple, it contains two EditText types of fields, for name and phone number, and two buttons, confirm and back. For example, the button confirmation from the *CustomerSettingsActivity* will save the introduced data in the database under *Users - Customers - UserId*. The information is retrieved from the fields using the *dataSnapsho*t function from Android Studio and they are saved in the database with the sequence: *mCustomerDatabase.updateChildren(userInfo)*. Thus the database will be updated with the user's new added personal data and this data will be delivered to the driver when a request is accepted.

Chapter 5. Testing and Results

5.1 Testing Methods

A. Android Studio Emulator

The Android Emulator simulates Android devices on the computer so that it is possible to test an application on a variety of devices and Android API levels without needing to have each physical device. The emulator provides almost all of the capabilities of a real Android device.

In order to create a new device, select from the toolbar Open AVD Manager. This option is between the build and run functions of the Android Studio. Next a window for creating a new device will open. After pressing the button Create Virtual Device a new window will pop up which allows the users to select a device as it is represented in Figure 48. The following step is naming the device and the api level. For example i left the initial name, Pixel 2, and i put API 29 as it is the maximum API level selected for Fritaxi. Before finishing the virtual device, the user is also allowed to select the orientation, portrait or landscape.

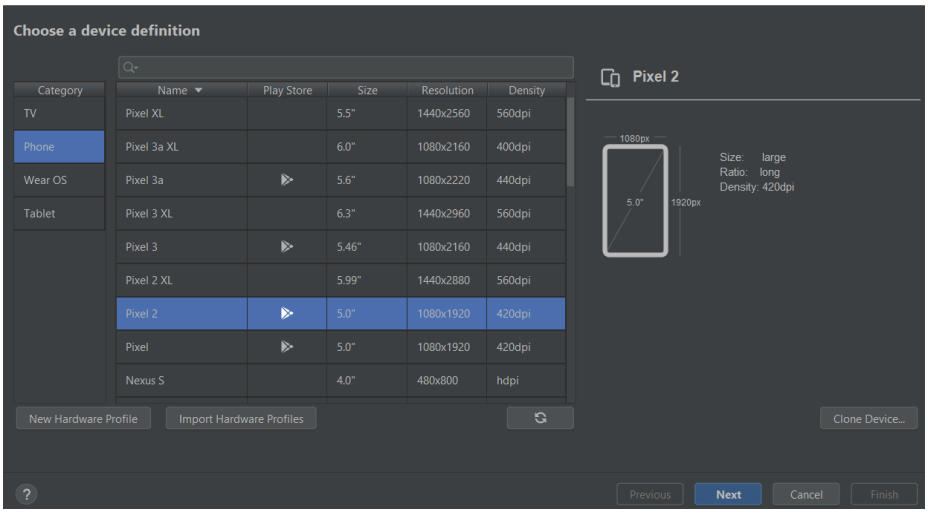

Figure 48. Emulator Configuration

After the new device is created, the AVD Manager window will display all the existing devices with a summary as in Figure 49.

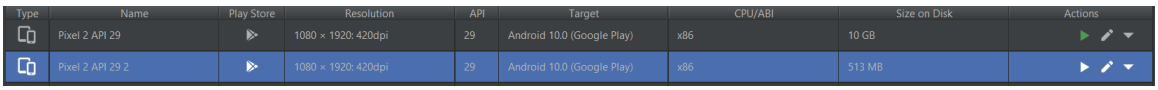

Figure 49. AVD Manager

The location of the users on the emulators are random, but they can be set by sending the emulators real geographical coordinates like in Figure 50.

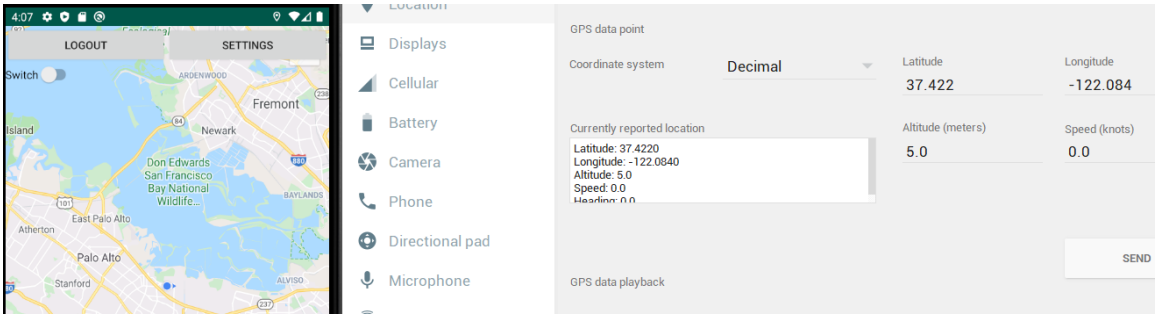

Figure 50. GPS coordinates for emulators

It is important to note that it is possible to run the application on multiple devices at the same time. In my case I needed at least two devices for a driver and a customer.

B. Mobile device with Android System

As mentioned in the previous chapters, Fritaxi is available for all devices with Android Operating System. Since the project is not uploaded to the Google Play Store, the application has to be deployed on the device through Android Studio.

The first step is to enable the developer mode on the device. For example, on the smartphones made by samsung this can be done by pressing the device number in Settings six times. Once the developer mode is enabled, the device has to be connected to the computer with a USB cable in order to connect to Android Studio. Once the platform recognizes the device, the option to Run the application on the smartphone instead of on an emulator should be visible.

5.2 Test Cases

The initial page after the application is installed on a device is presented in Figure 51. The layout is simple, consisting of two buttons which lead to the registration/ login pages for each type of account.

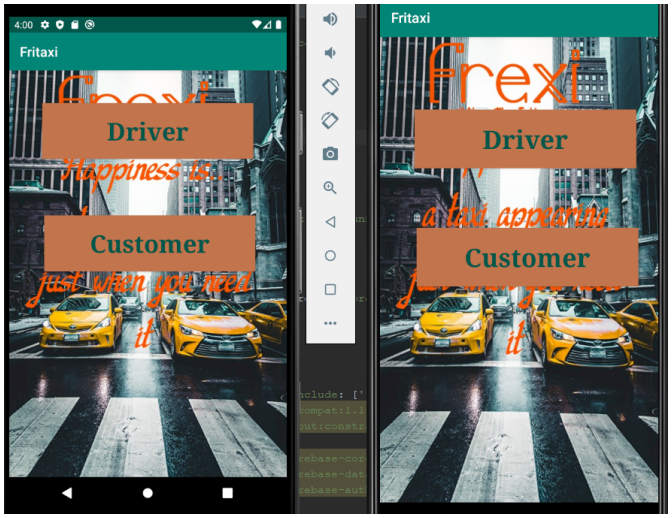

Figure 51. The initial activity

A summary of all the test cases is presented in Table 3 and below the table all the steps required for the tests are explained.

Case Nr.	User action	Expected behaviour of the application	Pass/ Fail
1.	Registration	A new account is added to the database as driver or customer with respect to the user's choice	P
2.	Login	The corresponding Map activity is started; the app requests access to the GPS service	P
3.	Driver Switch ON	The driver is marked as available for requests	P
4.	Driver Request	The request is added to the database	P
5.	Settings Button	Users are able to add a name and phone number to their accounts	P
6.	Ride process	Varied responses representing the ride from the request to the customer pick up	P
7.	Destination	The driver receives the destination of the ride	P
8.	End/Cancel ride	The whole process and the database is reseted	P
9.	Logout	Users are redirected to the home page	P

Table 3. Summary of the test cases

Test case 1: Figure 52 displays the registration pages which become available once the user presses a button from the home page.

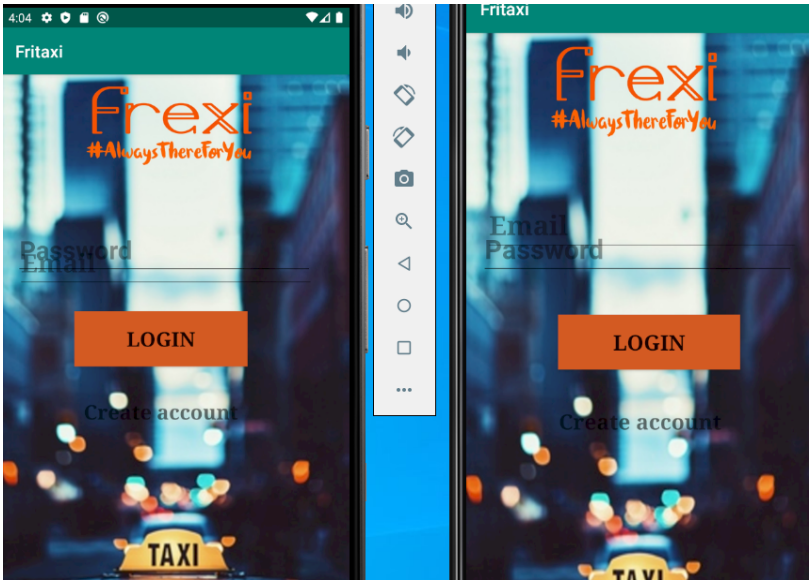

Figure 52. Registration/Login Activities

In Figure 52, on the right side is presented the login page for the drivers and on the left side the login page for the customers. When the Create an Account option is pressed the login button turns invisible and the registration button is visible. Other important checks before testing the registration is the password hashing and the minimum characters accepted for the password is six. Both functions passed the check by giving an error and refusing the registration of an account.

For this case, i made two accounts with the following credentials:

- Driver account - Email: cased@gmail.com

  Password: 123456

- Customer account - Email: casec@gmail.com

  Password: 123456

The last step is to verify the Firebase database for the entrance of the new accounts as shown in Figure 53.

casec@gmail.com		21 Jan 2021	21 Jan 2021	UrqEy8uWCqgKl6q2k6MiB4As09z1
cased@gmail.com		21 Jan 2021	21 Jan 2021	z8H8BeN2JxehRxnGORdGko53Sjf2

Figure 53. Firebase authentication

Test case 2: Before logging in with an existing account, I tried using an inexisting email to check the message displayed. The application passed the test by displaying the error from Figure 54.

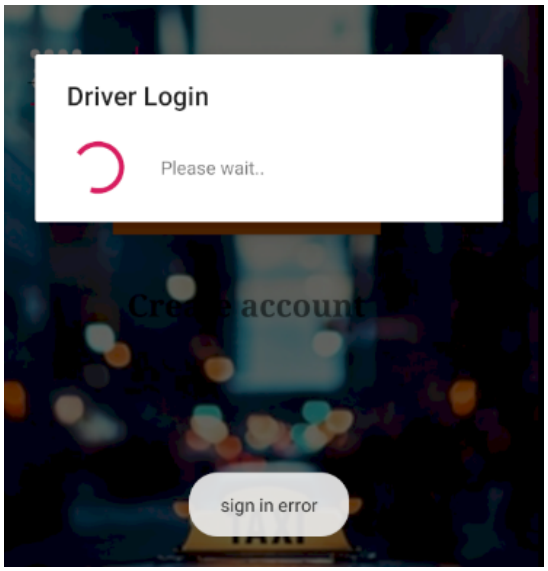

Figure 54. Sign in error

For the next step I used the accounts made in the previous test case. The application has to recognize the type of the account, driver or customer, and proceed to open the respective Map Activity. More importantly, before being able to access the GPS functions, the app will request permissions as presented in Figure 55. If the location service is allowed, then the coordinates of the user's position are saved in the database.

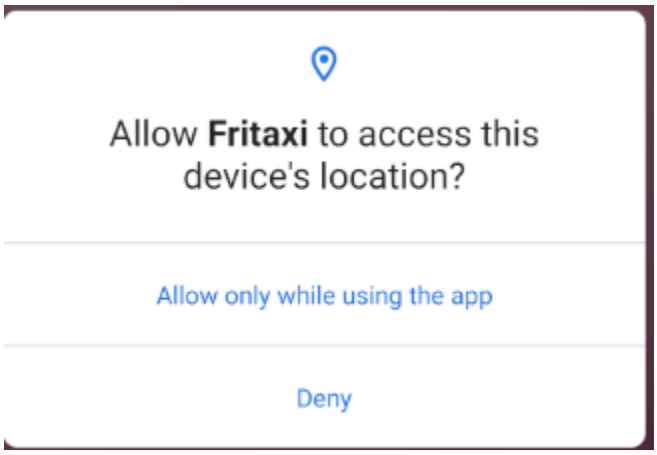

Figure 55. Location permission request

Test Case 3: The users logged in as drivers have a switch that is turned off when they enter their accounts. This switch controls their status in the database. If it is off, the driver is not working at the moment and when it is on, the driver is available. Thus, when on, a new child is activated in the database named *driversAvailable* consisting of the ids of all the drivers which are available to receive requests from customers. The database at this point can be seen in Figure 56.

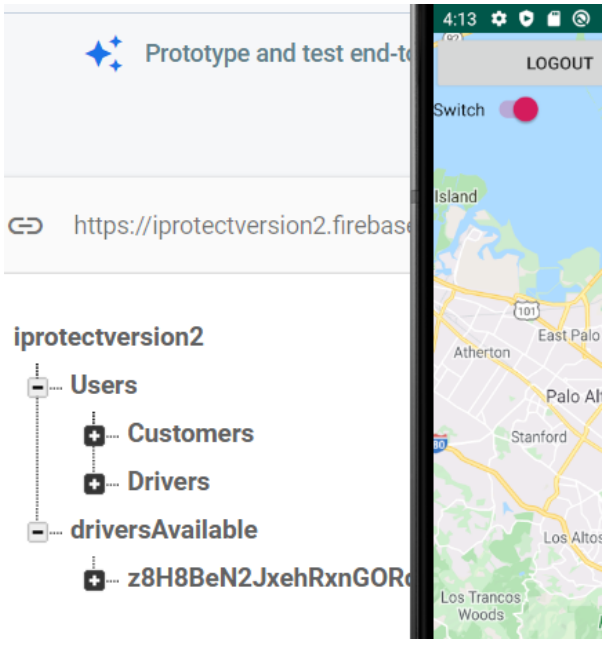

Figure 56. Database with available drivers

Test case 4: When a user logged in as a customer presses the button for driver request, a new child is active in the database named *customerRequest*. This child consists of all the requests made by the customers. The database, at this point, is presented in Figure 57.

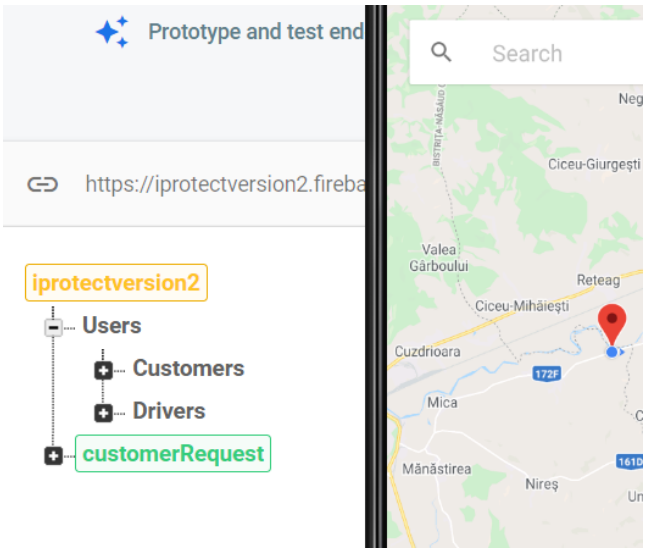

Figure 57. Database with driver request

Test case 5: Both types of users are provided with a button to access the settings page which allows them to add personal information to their accounts, specifically a name and a phone number. For the test I used the name *Customer* and the phone number *333333*. The point is that when the button confirm is pressed the informations are added to the database under the id of the current user. The outcome of the test is presented in Figure 58.

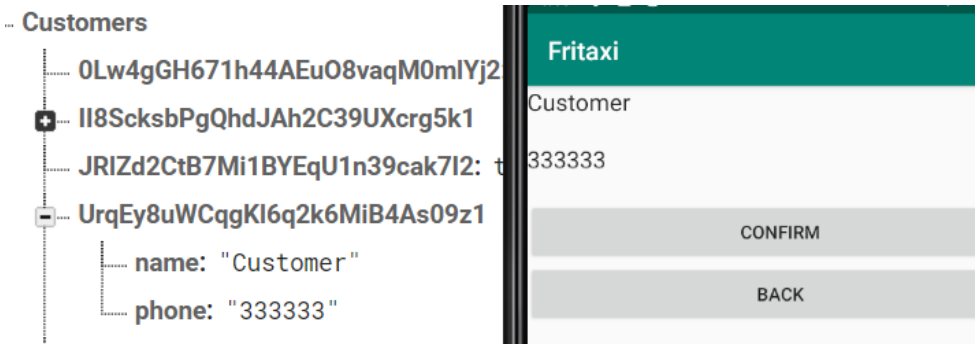

Figure 58. Personal information added to database

Test case 6: First i logged in with driver accounts on two emulators having different geographical coordinates and i made them available. On a smartphone I logged in with a customer account and placed a driver request. The application successfully found the closest driver as displayed in Figure 59.

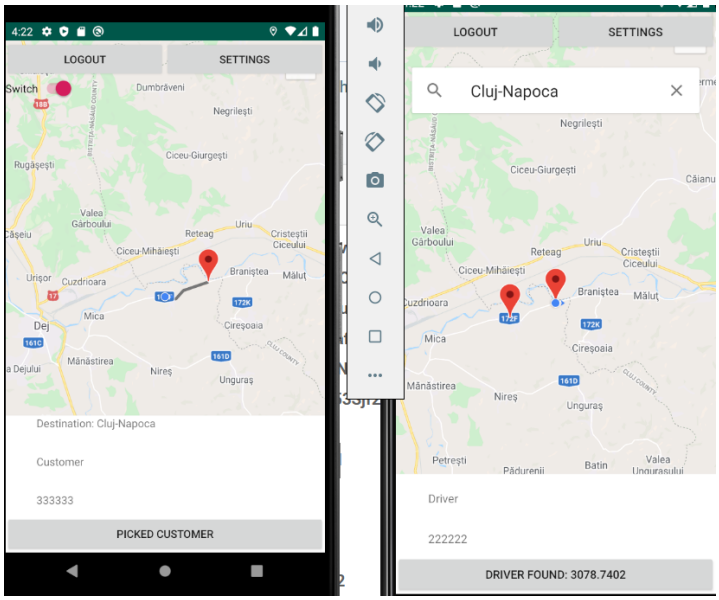

Figure 59. On the left driver and on the right customer activities during a ride

From the previous figure it is also demonstrated that the driver received the customer's information and vice versa. The route from the driver's location to the position of the customer has been displayed and the customer is able to see the location of the driver through a marker. Moreover, the customer is also provided with a number representing the distance from the driver to his current location.

In the database, it is important to check the current status of the id of the driver. Since the driver is working, he is moved from the *driversAvailable* child to the *driversWorking* child as presented in Figure 60.

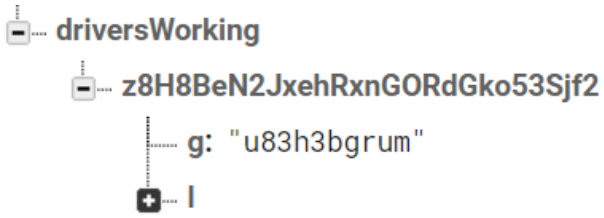

Figure 60. Database with working drivers

The request made by the customer is also moved in the database under the id of the drivers which picked up the request. The request includes all the information of the customer such as name and phone number.

When the driver arrives at the location of the customer, the message *Driver is here* is displayed for the customer.

It is also important to note for the following case that it is clear from the figures in this case the correct transmission of the destination.

Test case 7: Since it was mostly demonstrated at the previous test case, I will present the changes in the database. The request made by the customer includes the name and the geographical coordinates of the destination. As exemple Figure 61 has the destination Cluj-Napoca. Also in the figure, the request has already been received by a driver as the request is found under the id of the driver.

Figure 61. Database with request including a destination

After the driver picks up the customer, in the case of a ride with a destination, the application will automatically display the fastest route to the driver. The process is exemplified in Figure 62.

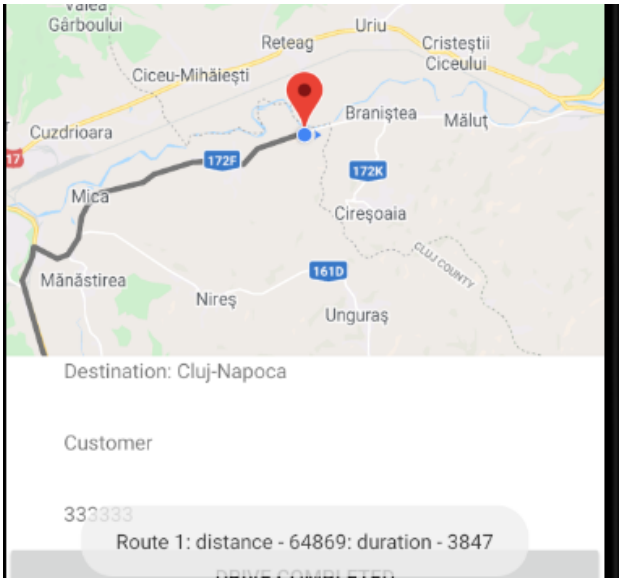

Figure 62. Route to destination

Test case 8: Continuing the ride from the previous case, first I canceled the process from the customer account and then I restarted the process of the ride and ended the ride from the driver account. In both cases the result was the same. The new childs added to the database were deleted, the markers and routes from the map were removed and the users

were able to continue using the application without problems. Therefore the test of canceling and ending the ride was successful.

Test case 9: The last test case is represented by the pressing of the button logout. For this test i logged out from the previous accounts of the driver and customer implicated in the ride and was successfully redirected to the initial activity. Without pressing the logout button, the user remains connected in the account and will skip the registration and login activity.

Chapter 6. Conclusion

6.1 Contribution Summary

For my final project I focused on the implementation of an application designed for devices with Android OS. Considering a simpler approach to the issues studied for the development of this system, in the initial state of the project, the description of the system functionality was made through use case diagrams. Along the way, it was possible to obtain a modeling of the general algorithm of the application, with the help of Petri Nets, simulated and tested in the CPN Tools software environment, resulting in a viable solution. Once certain points in the system development were reached, sequential diagrams could be made, which describe the flow of messages exchanged between the component modules. and Finally the components diagrams for drivers and customers were drawn, describing the bond and the flow of actions during the process of a ride.

6.2 Reaching the objectives

Fritaxi represents an alternative solution to the existing taxi booking applications, and its main objective is proving a taxi for a customer in the smallest time interval as possible, while keeping the interface simple and user friendly for a better experience. The application also allows the users to register as taxi drivers for anyone looking for a way to earn extra income.

Considering the successful tests which were made, it can be said that Fritaxi has a very strong connection to the Firebase server and database and it was developed in a way that takes in consideration all possible errors. The customer is able to request a taxi whenever and wherever as the GPS localization is not limited. It is also doable to add a destination from a different country. To put the previous sequences in a single word, the application is limitless as of geographical potential. If the customer wants to finish the ride after the request is active, for any reason, he has the option to cancel the ride without affecting the connection. Both users caught in the respective ride session will be brought back to the initial page and will be able to request or work immediately. In the case of the drivers, they are also provided with functions to connect, disconnect and to end the ride at any moment without affecting the server. These situations were taken into considerations as they were presented in the testing chapter.

6.3 Further development possibilities

While the project i proposed may have several advantages, there will always be functions that can be added or aspects that can be improved, algorithms that can be streamlined, which would lead to an increase in the effectiveness of the application.

In the future, Fritaxi could be improved from several points of view. When it comes to the implementation, the application could have a better system which is searching for available drivers, improved routing services which would find the fastest route to the customer and destination considering the actual traffic and it could have its own server even if just as a back up. One of the test cases which i could not test is the

situation in which over ten customers and drivers would be active. A back up would be required so the system would not crash and cause problems for both users and drivers.

From the point of view of functions which could be further developed, it is necessary to compare Fritaxi to the existing taxi applications and find what it lacks. For example it is common nowadays for applications to provide services which allow the users to pay with a card using the application after a ride is completed. This is called billing service and it represents a reason for customers to prefer a taxi application over another one. Other functions include displaying the waiting time to the customer while he is waiting for the driver to arrive at his location, providing a chat service for drivers and customers so they would be able to communicate with each other in case of delays or if the customers desires a different pick up location, implementing different car services such as fancier cars or delivery options. The list of possible functions which could be further implemented is long and it shows how far the technology has reached in terms of taxi services.

To conclude my thesis, I am content regarding Fritaxi as a taxi application which is available even for smaller cities as it is my hometown. As it is at the moment, the application provides the functions necessarily to replace calling for taxi services, and the interface is straightforward and easy to use. With many options as further development, Fritaxi is still at the beginning and it could reach a different state with further improvements.

Bibliography

[1] Engineer's Forum Master Account, "The evolution of the mobile phone" , posted in Articles, 2013, website link: http://www.ef.org.vt.edu/archives/951

[2] Daniel Dudley, "The Evolution Of Mobile Phones: 1973 To 2019", article posted on the flauntdigital.com blog, November 2018, website link https://flauntdigital.com/blog/evolution-mobile-phones/.

[3] Anonim writer, "What Everybody Ought to Know About Android : Introduction, Features & Applications", article posted on elprocus.com, direct link:https://www.elprocus.com/what-is-android-introduction-features-application[

[4] Anonim writer, "Android (operating system)", article posted on wikipedia, direct link: https://en.wikipedia.org/wiki/Android_(operating_system).

[5] Cammeron Summerson, Justin Duino, "What's the latest version of Android?", updated October 2020, article posted on howtogeek.com, direct link: https://www.howtogeek.com/345250/whats-the-latest-version-of-android/

[6] Anonim Writer, "Android Revolution In Mobile Technology Computer Science Essay", posted in Computer Science, January 1970, direct link: https://www.ukessays.com/essays/computer-science/android-revolution-in-mobile-technology-computer-science-essay.php.

[7] Anonim Writer, "Android Operating System Analysis", article posted in Computer Science, ukessays.com, June 2017, direct link: https://www.ukessays.com/essays/computer-science/android-operating-system-revolution-in-technology-computer-science-essay.php.

[8] Agicent App Development Team, "Why Android Studio is Awesome", posted on medium.com, August 2019, direct link to the article website: https://medium.com/@agicent/why-android-studio-is-awesome-c606c94366e6

[9] Deepam Goel, "Understanding Android Architecture", article posted on medium.com, May 2018, direct link to the article website: https://medium.com/@deepamgoel/understanding-android-architecture-1f0fb4b52f90.

[10] Android Studio Guide, "Guide to app architecture", in Jetpack Get Started, direct link: https://developer.android.com/jetpack/guide.

[11] Tam H. Doan, "Virtual Machines in Android Studio: Everything you need to know", posted on Android Hub blog, August 2019, direct link:

https://android.jlelse.eu/virtual-machine-in-android-everything-you-need-to-know-9ec695f7313b.

[12] Himanshu Verma, "JVM vs DVM", article from towardsdatascience.com, December 2019, direct link to website: https://towardsdatascience.com/jvm-vs-dvm-b257229d18a2

[13] Unknown Write, "Android fundamentals 02.2: Activity lifecycle and state", on developer.android.com, November 2020, direct link to the website: https://developer.android.com/codelabs/android-training-activity-lifecycle-and-state#0

[14] Ashok Kumar, “Android Activity Lifecycle - a Brief”, article posted in Android App Development on loginworks.com, November 2017, direct link: https://www.loginworks.com/blogs/android-activity-lifecycle-brief/.

[15] Anonim Writer, “Android SDK”, article from techopedia.com, October 2020, direct link: https://www.techopedia.com/definition/4220/android-sdk.

[16] Official linux guide, “ADT Plugin”, direct link to website: https://www.linuxtopia.org/online_books/android/devguide/guide/developing/tools/adt.html.

[17] Joe Hindy, “10 best GPS apps and navigation apps for Android”, in Apps, on androidauthority.com, September 2020, direct link: https://www.androidauthority.com/best-gps-app-and-navigation-app-for-android-357870/

[18] Aurelian Mihai, “Top cele mai bune aplicatii taxi pentru mobil”, article posted on got4it.ro, Romania, March 2019, direct link: https://www.go4it.ro/content/internet/cele-mai-bune-aplicatii-de-taxi-pentru-mobil-17988699/.

[19] Anonim Writer, “24 best tracking apps for Android & IOS”, in Industries Mobile Apps Tracking and Delivery Apps, on redbytes.in, September 2020, direct link: https://www.redbytes.in/best-gps-tracking-apps-for-android/

[20] Wikipedia Article “Object-Oriented Programming”, direct article link: https://en.wikipedia.org/wiki/Object-oriented_programming

[21] Edpresso Team, “What is Object-Oriented Programming”, in Object Oriented Design Beginner Templates, April 2020, direct website link: https://www.educative.io/edpresso/what-is-objectoriented-programming

[22] Howard Austerlitz, “Java Programming Language”, in Data Acquisition Techniques Using PCs (Second Edition), 2003, direct link: https://www.sciencedirect.com/topics/computer-science/java-programming-language

[23] Wikipedia Article, “Java (programming language)”, direct website link: https://en.wikipedia.org/wiki/Java_(programming_language)

[24] Article provided by O’reilly, “The Client-Server Model”, direct link: https://www.oreilly.com/library/view/java-network-programming/1565928709/ch02s05.html

[25] Aerospace Corporation, “GPS Primer: A Student Guide to the Global Positioning System”. Los Angeles, USA, 2003, available at : http://www.aero.org/education/primers/gps/GPS-Primer.pdf

[26] ICD-GPS-200C, “Interface Control Document: Navstar GPS Space Segment/ Navigation User Interfaces”, US DOD, I RN-200C-005R1, 14 Jan 2003, available at http://www.navcen.uscg.gov/pubs/gps/icd200/default.htm.

[27] Kijewski-Correa, A. Kareem and M. Kochly, “Experimental Verification and Full-Scale Deployment of Global Positioning Systems to Monitor The Dynamic Response of Tall Buildings”, in Journal of Structural Engineering, august 2006, available at:

https://www.researchgate.net/publication/228638596_Experimental_Verification_and_Full-Scale_Deployment_of_Global_Positioning_Systems_to_Monitor_the_Dynamic_Response_of_Tall_Buildings

[28] Provided by the website founders, "What is GPS", direct article link: https://www.garmin.com/ro-RO/aboutGPS/

[29] Sea & Space Corporation, "How does GPS work?", in Navigation, available at: https://www.eso.org/public/outreach/eduoff/seaspace/docs/navigation/navgps/navgps-3.html

[30] Member #23999, "GPS Basics", in tutorials, at learn.sparkfun.com, direct link: https://learn.sparkfun.com/tutorials/gps-basics/all

[31] Wikipedia Article, "General Packet Radio Service", available at https://en.wikipedia.org/wiki/General_Packet_Radio_Service

[32] Simson Garfinkel, "What is GPRS", article on wired.com, January 2002, available at: https://www.wired.com/2002/09/what-is-gprs/.

[33] Margaret Rouse, "GPRS (General Packet Radio Service)", in Data and Infrastructure - Wireless and mobile, January 2002, direct article link: https://searchmobilecomputing.techtarget.com/definition/GPRS

[34] Wikipedia Article, "Wi-fi", available at https://en.wikipedia.org/wiki/Wi-Fi.

[35] Kaiti Norton, "Wi-Fi Definition and Meaning", in Definitions, on webotopia.com, direct link: https://www.webopedia.com/definitions/wi-fi/

[36] Firebase Guide, "Firebase Realtime Database", in Guides, available at: https://firebase.google.com/docs/database.

[37] Techopedia Firebase Guides, "Firebase Realtime Database", website link: https://www.techotopia.com/index.php/Firebase_Realtime_Database

[38] Doug Stevenson, "What is Firebase? the complete story abridge", from medium.com, January 2002, article available at: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

[39] Geeky Ants, "Introduction to Firebase", posted on hackernoon.com, December 2017, available at: https://hackernoon.com/introduction-to-firebase-218a23186cd7

[40] Amit, "All you need to know about UML Diagram: Types, +5 Examples", tallyfy.com, 2018, available at: http://tallyfy.com/uml-diagram/.

[41] Article provided by Visual Paradigm, "What is a Component Diagram", available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

List of Tables

List of Figures

ACRONYMS

GPS – Global Positioning System

OS – Operating System

SDK – Software Development Kit

ADT – Android Development Tools

JVM – Java Virtual Machine

DVM – Dalvik Virtual Machine

API – Application Programming Interface

Wi-Fi – Wireless Field

GSM – Global System for Mobile Communication

AVD - Android Virtual Device