Development of a mobile application for carpooling the elderly

Preprint · June 2019

DOI: 10.13140/RG.2.2.22674.20166

1 author:

Lamine Fetni
Universität Bremen
1 PUBLICATION **1** CITATION

SEE PROFILE

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
ABDELHAMID MEHRI CONSTANTINE UNIVERSITY 2
FACULTY OF NEW TECHNOLOGIES OF INFORMATION AND COMMUNICATION
DEPARTMENT OF COMPUTER SCIENCE AND ITS APPLICATIONS

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR IN COMPUTER SCIENCE

SPECIALTY: SCI

Theme

Development of a mobile application for carpooling the elderly

Submitted by:
Mohamed Lamine FETNI

supervised by:
Imene BENSETIRA

2018/2019
JULY SESSION

ACKNOWLEDGEMENTS

I would like to express my special thanks of gratitude to my supervisor Dr. Imene BENSETIRA for her experience and tips and for supervising the work we have been doing.

I would also like to extend my sincere thanks to our Jury who gave us the golden opportunity to do this wonderful project on the topic of developing a mobile application for carpooling the elderly, which also helped me in doing a lot of Research and gain a lot of Experience and made me came to know about so many new things I am really thankful to them.

Finally I would also like to thank my mother and friends and dear ones who helped me a lot in finalizing this project within the limited time frame.

MOHAMED LAMINE FETNI

ABSTRACT

In general, people have a hard time conciliating their schedules because of the way they move from one location to another. And elders suffer from this the most especially here in Algeria since transportation between cities is not that great, As students, we think there should exist more suitable transportation solutions to places where transportation networks are short and cheap and helpful for elders.

This thesis proposes a platform to help improve elder's mobility through carpooling, a way for car drivers to share their private vehicle with more elders in order to splitting and reducing costs. Carpooling may be one of the best solutions when there is no other mean of transportation to a specific location but naturally it is not the only one. Mobile applications take more and more part of everyone's lives, different services for carpooling with different features begin to compete with existing transportation solutions. Some people start to prefer using new carpooling services over the traditional services represented by taxi services. GoRide aims to promote carpooling by targeting elders making it easier for them to adhere and use this system.

By targeting elders people will more likely join the service since its users are primarily other people form the same environment. To put the carpooling system in place, we have designed and developed an Android mobile application with backend servers for users to access the carpooling service through their smartphones, additionally the application involves some features that are critical to the service. By using Android Development Tools and Libraries and efficient backend solutions we have managed to make the application simple but powerful as well, which makes this application very useful for the young and the old to use.

Our app GoRide will be a unique carpooling application that would take benefits of the advantages of carpooling and try to improve and eliminate the disadvantages, all while focusing on making it a good carpooling experience for elders.

The realization of our project will go through the conceptual phase and then development phase. Since making a good application requires good planning first.

Keywords: GoRide, carpooling, elders ,mobile application, Firebase, Android.

Table Of Content

List of Figures

List of tables

General Introduction

While transport is becoming easier everyday thanks to innovation and the evolution of technologies. And while carpooling is not a new concept. People spend hours looking for cheaper and easier ways to travel and have new experiences. Put all the benefits of carpooling together and you get GoRide. GoRide is a mobile application that aims to help elders find shared rides (Carpooling) for easier, cheaper and safer trips all over Algeria.

This application provides its users with an option to offer/take rides so that you can save your money and fuel to build an Eco friendly environment. It also helps elders get trusted rides to makes their travel more comfortable and easy.

The development process can be severed into three sections i.e., design, development, and implementation. The application design process requires good knowledge of Android operating system architecture and general understanding of the application's intent. The second part of the process, development, is the most important and crucial part in the whole process; it requires knowledge of several programming languages, environment setup, and code debugging. Implementation the final part of the development process which focuses on deploying the application on mobile devices after it has successfully been tested on virtual devices.

Ultimately this project is expected to implement an application with all of the components that serve for the purpose of carpooling. In this thesis we will discuss mobile technologies and general carpooling ideas to better understand the field we are working with and to understand the technologies available for us to build a better idea about what we are going to work with, after this is done we are going to focus on the core concept and idea of the application alongside with our needs for it. After the core concept is ready we will turn that concept into a real application by implementing it into a real Android Application with a simple and beautiful design and also a strong and efficient backend service.

Document Structure:

This document will be split into three main chapters. The first one will give important definitions about mobile systems and solutions and also give a formal definition of carpooling and of the problem that will be solved in this thesis alongside with a glimpse of the project description. The second chapter will describe the design and concept of the carpooling solution we have come up with the concept includes functional and non-

functional needs alongside with project specifications and diagrams to make the core idea of the final application. The third and last chapter will describe different tools that are used for the implementation of the solutions to the carpooling problem. It will also describe those solutions in detail. It will contain all final design implementations and solutions, it will briefly explain how each interface of the application work for a better understanding of how the transition from a concept into a real implementation was done and how it was achieved.

I. Chapter 01 : General Project Framework

1. Introduction

The mobile application industry is booming. Smartphones are becoming more popular every year almost everyone uses one, we'll take a look at what defines mobile technologies and understand how mobile phones work, and what's the best operating system for an application development. We also take a look at different types of mobile applications and the best one to use for our project, we will understand the mobile architecture and how we can use it to make a good ride-sharing application we will also understand carpooling and give a general project description.

2. Mobile Technologies

2.1. Architecture of a mobile system

Application architecture is a set of technologies and models for the development of fully-structured mobile programs based on industry and vendor-specific standards. As you develop the architecture of your application, you also consider programs that work on wireless devices such as smartphones and tablets. [1]

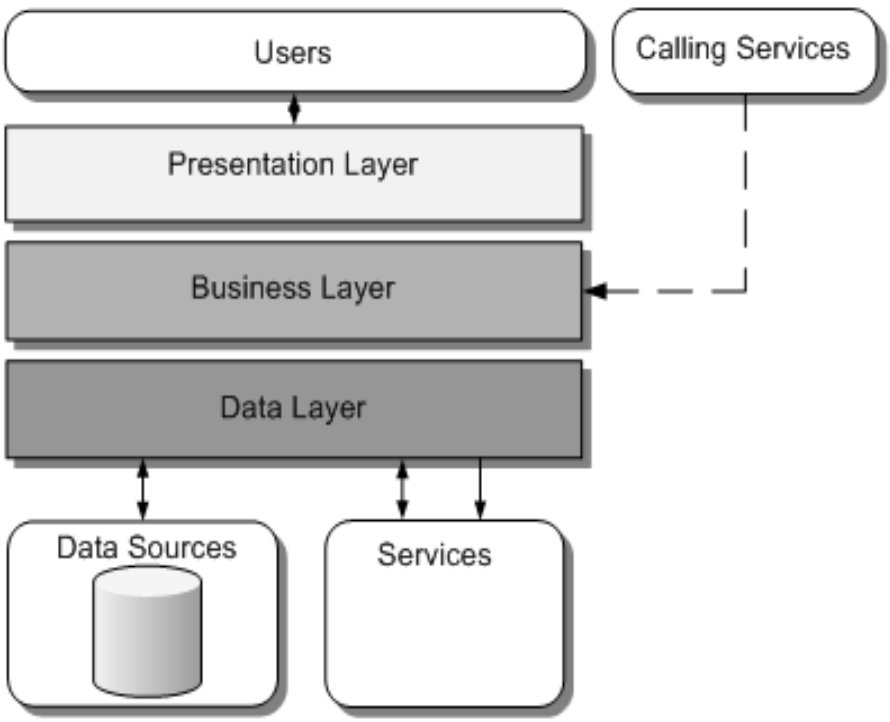

Figure 1 Simplified android architecture layers

Mobile application architecture design usually consists of multiple layers, including:

Presentation Layer - contains User Interface (UI) components as well as the components processing them.

Business Layer - composed of workflows, business entities and components.

Data layer - comprises data utilities, data access components and service agents.

A detailed Mobile System Architecture should also include these steps for better quality: [2]

Determining the Device

Here you need to keep the device types in mind. This covers the screen size, CPU characteristics, resolution (DPI), storage space and memory, and development tool environment availability.

Considering Bandwidth

There are times when connectivity is either intermittent or not available. Your application architecture needs to be built keeping in mind the worst network scenarios. Design your caching, data access mechanism…

Defining User Interface

You have the world and entire future ahead to show your creativity. Do not pour it all in at the very first stage. Keep your user interface as simple as possible. A muddled UI can become a major reason behind a mobile application's failure.

Navigation Methods

This one again comes on the design front. However, it requires expertise in both front and back end. There are numerous ways to navigate through application features, and it is important for you to analyze which one is good for your mobile application.

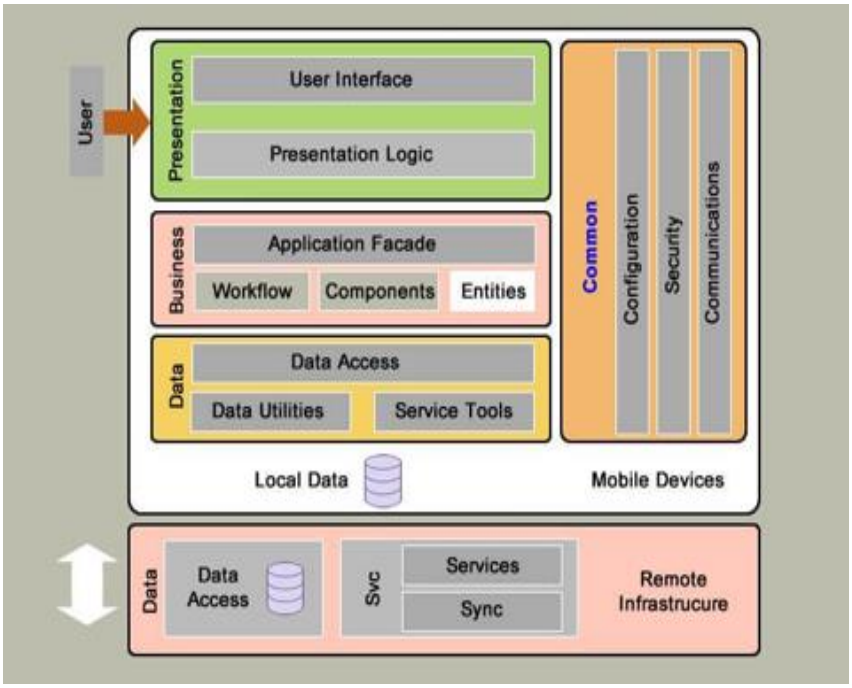

Figure 2 Detailed android architecture [3].

2.2. Mobile Application

2.2.1. Definition

A mobile application, most commonly referred to as an application, is a type of application software designed to run on a mobile device, such as a smartphone or tablet computer. Mobile applications frequently serve to provide users with similar services to those accessed on PCs. Applications are generally small, individual software units with limited function. This use of application software was originally popularized by Apple Inc. and its Application Store, which offers thousands of applications for the iPhone, iPad and iPod Touch.

A mobile application also may be known as an application, web application, online application, iPhone application or smartphone application.

Mobile applications are a move away from the integrated software systems generally found on PCs. Instead, each application provides limited and isolated functionality such as a game, calculator or mobile web browsing. Although applications may have avoided multitasking because of the limited hardware resources of the early mobile devices, their specificity is now part of their desirability because they allow consumers to hand-pick what their devices are able to do. [4]

2.2.2. History

The history of mobile applications dates back to the end of the 20th Century. In general, they were ringtone editors, small arcade games, calendars, calculators, and so forth. The new millennium saw the start of a swift market development of mobile applications and content. Operating systems for most smartphones (Symbian, Android, RIM, Mac iOS, Windows Mobile) allow the advancement of third-party software. This is different from the traditional programming environment of the ordinary cell phones.

The first ever recognizable mobile applications came about with the range of handheld computers from Psion. These were mostly PDAs that used the EPOC OS. In the early 90s, 16-bit machines (SIBO) were running on EPOC and allowed the users to access programs like Word processor, spreadsheet, database, and diary. Later on, models in the same range but running a 32-bit OS came with 2MB RAM that allowed users to install more applications through software packs or download if they were among the lucky few modem owners. EPOC was programmed in Open Programming Language (OPL) and allowed its users to create their applications[5].

2.2.3. How a mobile application works (Function)

There are mainly three kinds of applications- Native, Hybrid and Web-based. Each kind works differently as follows:

Native Applications

All Applications targeted towards particular mobile platforms are known as Native applications. Therefore, an Application meant for Apple device will never open in Android devices. This is why most businesses develop applications for multiple platforms.

While developing native applications, professionals incorporate best-in-class user interface modules. This accounts for better performance, consistency and good user experience. Users also benefit from wider access to Applications Program Interfaces (APIs) and make limitless use of all Applications from the particular device. Further, they also switch over from one application to another effortlessly. [6]

The main purpose behind creating such applications is to ensure best performance for specific mobile operating system.

Hybrid Applications

Concept of Hybrid Applications is a mix of native and web-based applications. Applications developed using Flutter, Xamarin, React Native, Sencha Touch and other similar technology fall within this category.

These are made to support web and native technologies across multiple platforms, hence the name hybrid. Moreover, these applications are easier and faster to develop. It involves use of single code which works in multiple mobile operating systems.

Despite such advantages, hybrid applications are slower in speed and performance. Often, applications fail to bear the same look n feel in different mobile operating systems.

Web-Based Applications

These Applications are coded in HTML5, CSS or JavaScript. Presence of strong internet connection is required for proper behavior and user-experience of this group of Applications.

By default, these Applications captures minimum memory space in the user devices compared to Native and Hybrid Applications. Since all the personal databases are saved on the Internet servers, users can fetch their desired data from any device through internet. The only con is that application developers don't get sufficient access to mobile operating system API. :[7]

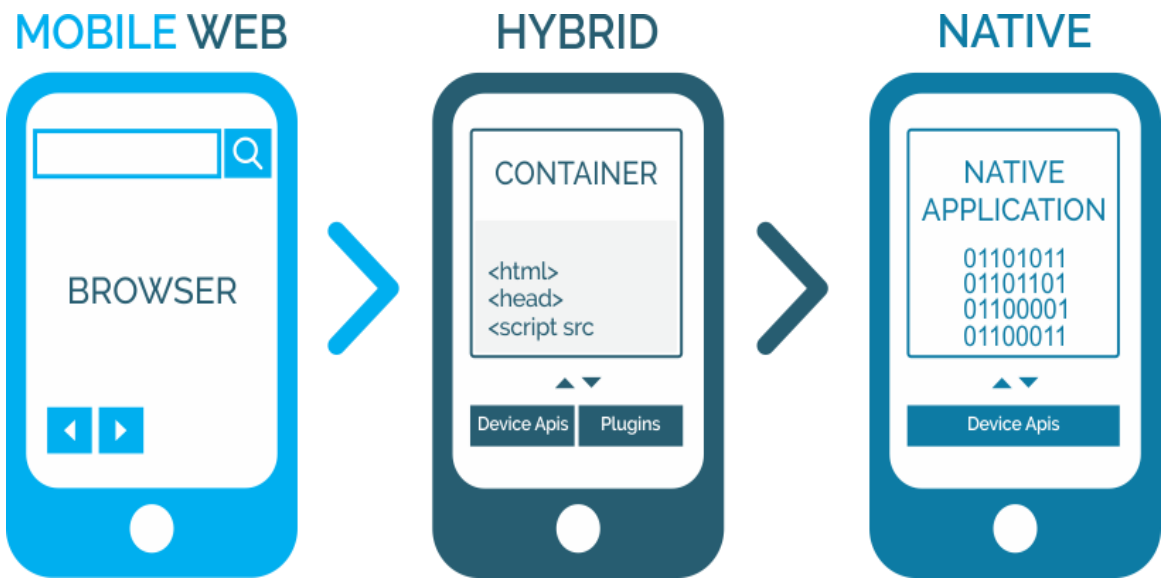

Figure 3 Web Applications vs Hybrid vs Native [7]

2.3. Mobile Operating Systems (OS)

2.3.1. iOS

Figure 4 iOS Logo [8]

iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod Touch. It is the second most popular mobile operating system globally after Android.

The iOS user interface is based upon direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch, all of which have specific definitions within the context of the iOS operating system and its multi-touch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching between portrait and landscape mode). Apple has been significantly praised for incorporating thorough accessibility functions into iOS, enabling users with vision and hearing disabilities to properly use its products.

The iOS SDK (Software Development Kit) allows for the development of mobile applications on iOS.

Combined with Xcode, the iOS SDK helps developers write iOS applications using officially supported programming languages, including Swift and Objective-C. Other companies have also created tools that allow for the development of native iOS applications using their respective programming languages. [8]

2.3.2. Windows Phone

Figure 5 Windows Phone Logo [9]

Windows Phone (WP) is a family of discontinued mobile operating systems developed by Microsoft for smartphones as the replacement successor to Windows Mobile and Zune. Windows Phone features a new user interface derived from Metro design language. Unlike Windows Mobile, it is primarily aimed at the consumer market rather than the enterprise market. It was first launched in October 2010 with Windows Phone 7. Windows Phone 8.1 is the latest public release of the operating system.

Windows Phone was succeeded by Windows 10 Mobile in 2015, it emphasizes a larger amount of integration and unification with its PC counterpart—including a new, unified application ecosystem, along with an expansion of its scope to include small-screened tablets.

In January 2019, Microsoft announced that support for Windows 10 Mobile would end on December 10, 2019, and that Windows 10 Mobile users should migrate to iOS or Android phones.

Later versions of Windows Phone support the running of managed code through a Common Language Runtime similar to that of the Windows operating system itself, as opposed to the .NET Compact Framework. This, along with support for native C and C++ libraries, allows some traditional Windows desktop programs to be easily ported to Windows Phone. [9]

2.3.3. Android

Figure 6 Android Logo [10]

Android is a mobile operating system developed by Google. It is based on a modified version of the Linux kernel and other open source software, and is designed primarily for touchscreen mobile devices such as smartphones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

Android is developed by Google until the latest changes and updates are ready to be released, at which point the source code is made available to the Android Open Source Project (AOSP), an open source initiative led by Google. The AOSP code can be found without modification on select devices, mainly the Nexus and Pixel series of devices. The source code is, in turn, customized and adapted by original equipment manufacturers (OEMs) to run on their hardware. Also, Android's source code does not contain the often proprietary device drivers that are needed for certain hardware components. As a result, most Android devices, including Google's own, ultimately ship with a combination of free and open source and proprietary software, with the software required for accessing Google services falling into the latter category.

Android Applications ("applications"), which extend the functionality of devices, are written using the Android software development kit (SDK) and, often, the Java programming language. Java may be combined with C/C++, together with a choice of non-default runtimes that allow better C++ support.
The Go programming language is also supported, although with a limited set of application programming interfaces (API).In May 2017, Google announced support for Android application development in the Kotlin programming language. [10]

2.3.3.1. Architecture

Android is structured in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. This architecture can be represented visually as outlined in Figure 7 Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.

On top of the Linux kernel, there are the middleware, libraries and APIs written in C, and application software running on an application framework which includes Java-compatible libraries. Development of the Linux kernel continues independently of Android's other source code projects like seen in Figure 8. [11]

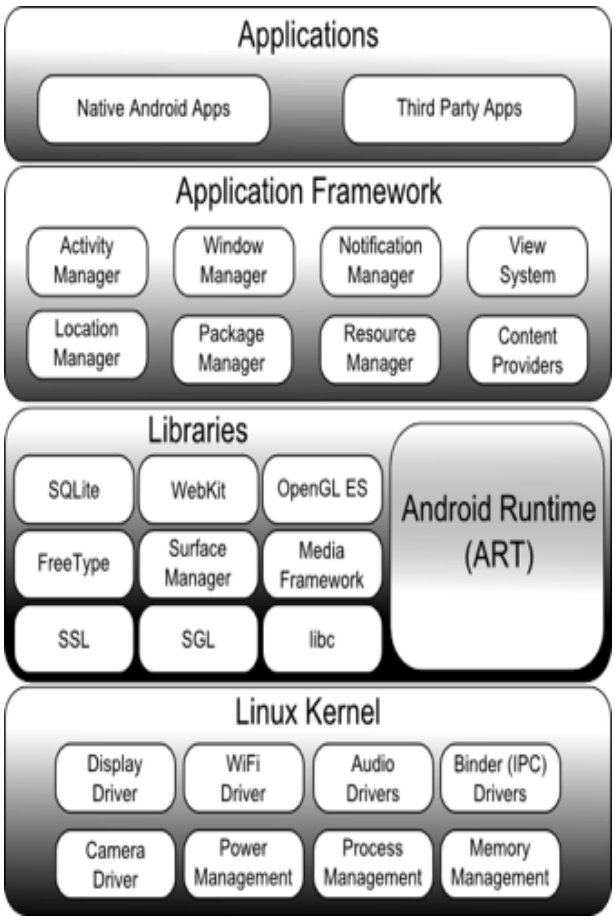

Figure 7 Android Architecture

Figure 8 The stack of Android Open Source Project [11]

2.3.3.2. Advantages and disadvantages

Advantages:

- Open source, so if you wanted you could make your own version
- Extremely customizable
- Has over 2 million applications available
- A wide choice of phones from many different brands gives you a better selection for a phone that fits you
- Most Android phones offer expandable storage via a MicroSD card
- Lots of applications still support KitKat (Version 4.4) which came out in 2013

Disadvantages:

- Often comes with tons of bloatware installed from the phone manufacturer
- Security standards aren't as high as iOS
- Lots of brands develop their own features which results in most phones not receiving the latest Android version for months if not a year
- Since there are many models of phones, cases are not in multitudes

3. Carpooling

3.1. Definition and general principle

Carpooling (also car-sharing, ride-sharing and lift-sharing) is the sharing of car journeys so that more than one person travels in a car, and prevents the need for others to have to drive to a location themselves.

By having more people using one vehicle, carpooling reduces each person's travel costs such as: fuel costs, tolls, and the stress of driving. Carpooling is also a more environmentally friendly and sustainable way to travel as sharing journeys reduces air pollution, carbon emissions, traffic congestion on the roads, and the need for parking spaces. Authorities often encourage carpooling, especially during periods of high pollution or high fuel prices.

Car sharing is a good way to use up the full seating capacity of a car, which would otherwise remain unused if it were just the driver using the car.

In 2009, carpooling represented 43.5% of all trips in the United States and 10% of commute trips. The majority of carpool commutes (over 60%) are "fam-pools" with family members.

Carpool commuting is more popular for people who work in places with more jobs nearby, and who live in places with higher residential densities. Carpooling is significantly correlated with transport operating costs, including fuel prices and commute length, and with measures of social capital, such as time spent with others, time spent eating and drinking and being unmarried. However, carpooling is significantly less likely among people who spend more time at work, elderly people, and homeowners.

Carpooling usually means to divide the travel expenses equally between all the occupants of the vehicle (driver or passenger). The driver does not try to earn money, but to share with several people the cost of a trip he would do anyway. The expenses to be divided basically include the fuel and possible tolls. But if we include in the calculation the depreciation of the vehicle purchase and maintenance, insurance and taxes paid by the driver, we get a cost around 100DA/km. There are platforms that facilitate carpooling by connecting people seeking respectively passengers and drivers. Usually there is a fare set up by the car driver and accepted by passengers because they get an agreement before trip start. [12]

3.2. Carpooling Types

3.2.1. Regular

The car is often perceived as an extension of the personal space, the driver, alone in his vehicle is in a closed space; he is free to do what he likes: listen to the radio, sing, call with headsets ... Carpooling regularly is to share a dialogue, experiences, stories.

In the United States an intermediate concept has developed between carpooling and the public transport line: the Vanpool. These are minibuses chartered by an employer, a public authority or a private company and made available to a group of people who regularly make the same journey.

3.2.2. Occasional

This type of carpooling is mainly used for leisure or last minute departures. The linking is often done through websites or mobile applications, which can significantly reduce travel costs, but usually requires to carpool with one or more unknown.

3.2.3. Eventual

Participants in an event (music festival, sporting event, wedding, associative or institutional meeting ...) can organize to carpool to the venue of the event. This one-time carpool has a special feature: all participants travel to the same place on the same date.

Carpooling is also used for departures on holidays or weekends, savings on a trip being even larger than the trip is long. So carpooling becomes an alternative of affordable and accessible transportation.

There are also "cultural" carpooling platforms to visit a cultural site: castles, museums, exhibitions, artists' studios, religious places, festivals, etc.

3.3. Existing Systems for carpooling

3.3.1. Websites

Algeria: www.nroho.com**,** www.m3aya.com**,**www.nsogo.net

Europe: BlaBlaCar.com**,** carpooling.com**,** GoMore.com

France: covoiturage.fr

USA: car.ma , www.rdvouz.com

World: Outpost.travel , joinntravel.com , www.letsride.in

3.3.2. Mobile Applications

Algeria: YAssir,Nsogo, AMIR

World: Uber, sRide, RideShare,

USA: Uber, Lyft

France: Karos, Wever, BlaBlaCar, OuiHop

3.4. Advantages and disadvantages of carpooling applications

Advantages of carpooling:

- The main advantage of rideshare solutions is cost saving, of course. Depending on the agreement, a ride can be twice as cheap than traveling by conventional way;
- The car is not a gas cost only. There are some cost items for the maintenance, repair, parts replacement in your vehicle. If you reduce the time of car utilization, you reduce these costs;
- A fewer number of cars on the road can reduce the CO2 emission in the roads and make the air we breathe cleaner;
- Saving time. Fewer cars - fewer traffic jams. That is to say, it is possible to reach a destination point faster and find a parking place.
- Meeting new people. Traveling together allows you to find good friends. [13]

Disadvantages of carpooling applications:

- Indecent passengers. Some people can try to discount the price or even ask the driver to visit the place that is not on the scheduled route. And it is important to screen out such individuals at the very beginning of traveling.
- Indecent drivers. Unfortunately, some drivers can play an unfair game as well. For example, drivers who take 5-6 people in a small car and ask for a high price.
- In some cases, a driver has to pick up each companion separately. It increases the time of traveling.
- Passengers can be not satisfied with the driving style of a car owner. In turn, the driver can be annoyed with an excessive volubility of companions, their untidiness or lack of manners. [13]

4. Project Description

This project (GoRide) aims to develop an Android based application for carpooling for elders, this application allows nonprofessional drivers to submit rides for specific targets and allows passengers to reserve/request rides from drivers all while being secure and having a simple interface.

This application will help elders save money and also reduce the pollution of the environment and effects of vehicles, this application focuses on serving needs of elders that may have disabilities and illnesses.

GoRide will be intended for the elderly in Algeria and it will support Android phones and Tablets, Users will need internet connection to use the application to offer or find a common route to travel to,

The application will have a simple and easy interface, Users must register at first before using the application, after that they must choose between a driver or a passenger, a driver can offer a drive to a specific location while a passenger can find or request a ride to a location.

Workflow of our Application GoRide:

The user downloads and installs the application in an Android device.

The user opens the application then he is welcomed with a Login/Register

A new user register via the register form (Name, Email, Birthday etc..)

The user must confirm the registration via Email or SMS.

After confirmation the user enters the application and chooses between Driver/Passenger

If the user is a driver he can post a trip by adding specific information (Origin, Destination...)

Passengers can search for rides or request a ride if it is not available

Drivers accept passengers into their ride until all seats are filled and the date is due.

Drivers can also accept requested rides from passengers.

Drivers can cancel, modify their rides.

Drivers/Passengers can view profiles and modify their own profile.

A main flowchart of the application can be seen in Figure 9, however this flowchart represents the main core of the application, many features will be added to improve the application as development goes on.

5. Application Flowchart

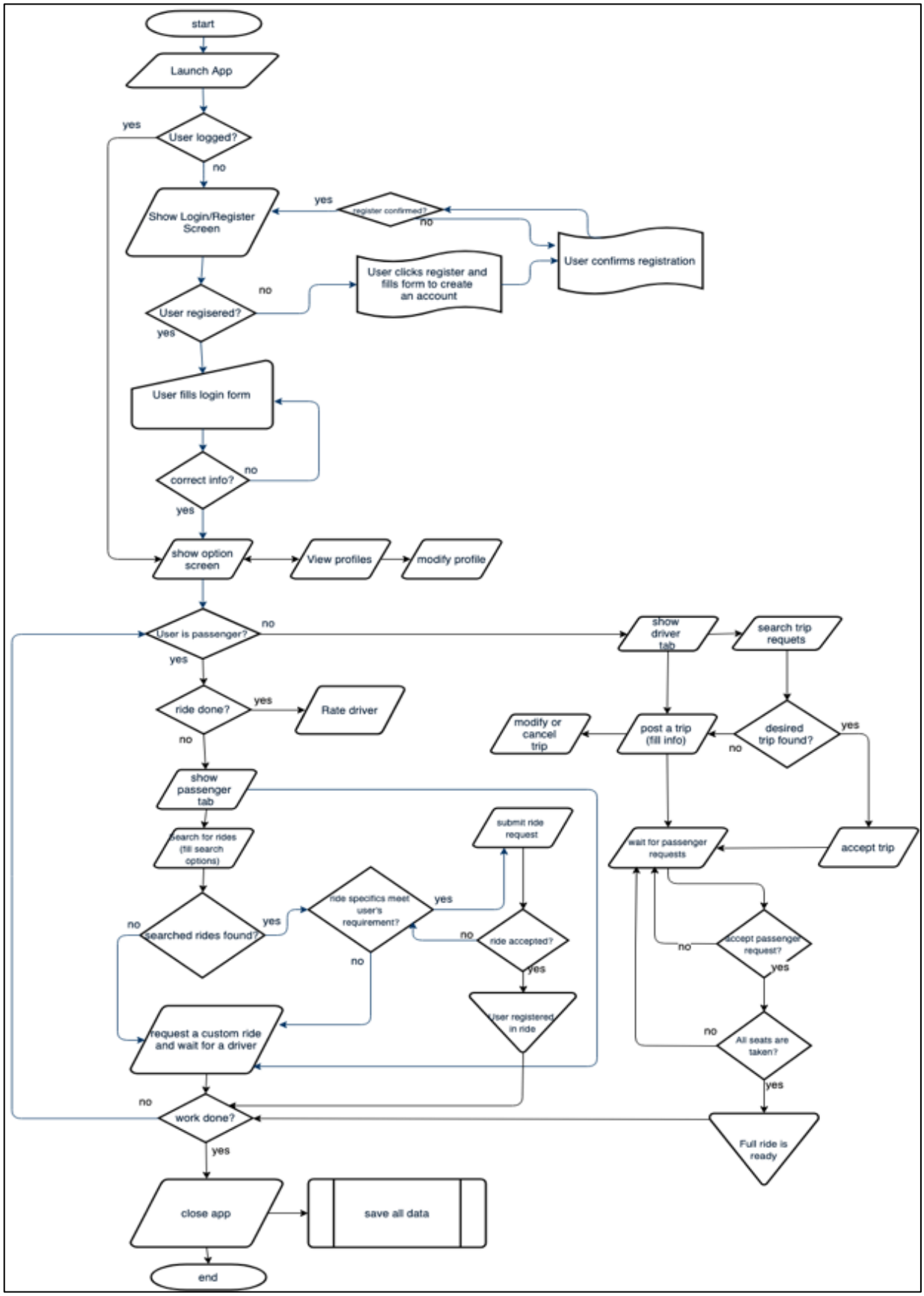

Figure 9 Application Flowchart

6. Conclusion

Now we understand that mobile technologies are a vast and a deep field to learn, there exist so many options for us to use and we also know that application development is complicated and requires deep planning and optimization, mobile OS's offer tools that help developers make applications in easy ways, we will take an advantage of this to make our carpooling application "GoRide".

Our Application will take full advantage of carpooling but at the same time we will minimize the disadvantage by making our application secure and efficient and make sure it's satisfying for all its users.

Finally and after understanding mobile technologies and solutions available to use, and after understanding how carpooling works, we can determine the best options to use to make our application but before the implementation comes the analysis and conceptual phase, which will cover the project plan and needs.

II. Chapter 02 : Project Analysis and Design

1. Introduction

The design of a project is important for the structure of the application by using UML (Unified Modeling Language) which is a general purpose modelling language, that aims to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

Reasons to use UML for project analysis and design are:

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

These project designs will try and make the overall idea of the project more understandable and clear by identifying Actors and functional / non-functional needs, And also all the diagrams needed to give a clear view about this project.

2. Specifications

2.1. Identification of needs

Application architecture is a set of technologies and models for the development of fully-structured mobile programs based on industry and vendor-specific standards. As we develop the architecture of our application, we also consider programs that work on wireless devices such as smartphones and tablets.

2.1.1. Functional requirements

-All **users** can:

- Create an account.
- Login.
- Choose an account type (Driver , Passenger).

- View Profiles.
- Modify their profiles info (phone, email, etc.).

-After identification **drivers** can:

- Submit a ride with specifications.
- Cancel a ride while notifying passengers.
- Modify a ride (date, number of seats, target, start point, etc.).
- Accept or Decline a ride request from a passenger.

-In addition **passengers** can:

- Search for a ride.
- View available rides on the map.
- Reserve a ride from a driver.
- Request a ride.
- Rate/Comment on a driver after a trip.
- Report drivers.

-also **Administrators** can:

- Disable accounts.
- Remove rides.
- View Application Statistics.
- Send Push Notifications and Updates.
- Database Management.
- Add more features.

2.1.2. Non-functional requirements

- Software extensibility:
- Ability to add or modify some features
- Multi-Platform Support (Smartphones, Tablets, IOS (*optional*)).
- Implementing an online backend for easier management (Firebase).
- Accessibility options (Blind, Deaf, etc..)
- Small Application size.
- Fast response from server.
- Secure system.
- Simplicity of interface.

- Development using Android Studio's Java/Kotlin.
- Clear Information and Communications.
- Google Maps API + Geolocalization.
- Clear Privacy Policy.

2.1.3. Optional requirements

- Drivers communicate with a passenger (Text messages).
- Drivers rating passengers
- Drivers can add comments on rides (Smoker, No Luggage, Bags, Pickup area...)
- SMS Verifications for extra security.
- Tutorial on how to use the application.
- Low API Target to support older phones.
- Payment Options

2.2. Actors identification

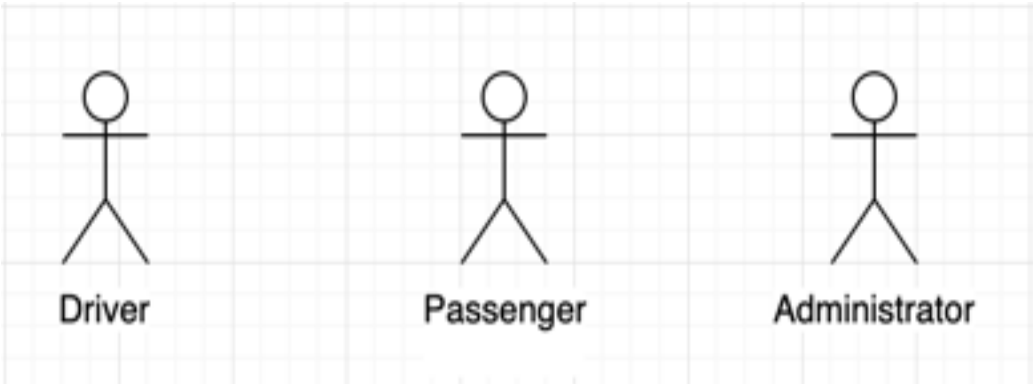

Figure 10 Actors identification

3. Use Case Diagram

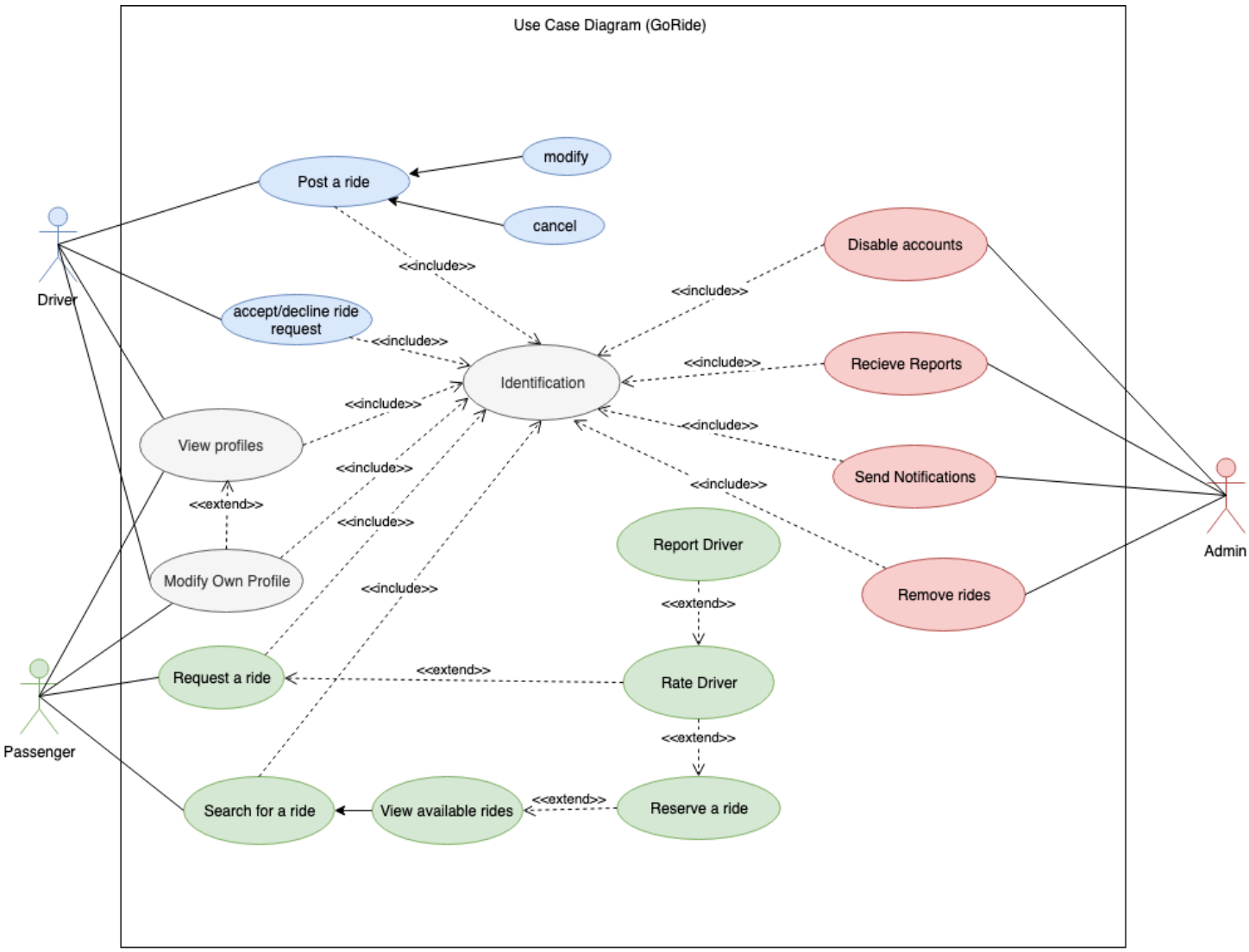

Figure 11 Use Case Diagram

3.1. Textual description of use cases

3.1.1. Identification

Name	Identification
Actors	Driver/Passenger/Admin
Objective	Login/Register to the system by filling either the login form or register form with the requirement information to access the Application's functions
Precondition	
Flow of events	1- The user opens the application 2- The Login/Register Interface is displayed 3-The user logs in or register an account by filling the required form 4- The system checks if the information are correct 5- The system saves the information and logs the user
Alternative flow	3.b- Wrong information -Go back to 2- 3.b.1 Error message is shown
Special Requirements	Internet connection
Post conditions	

Table 1 Identification case

3.1.2. Post a ride

Name	Post a ride
Actors	Driver
Objective	A Driver can submit a ride with a specific date with other specific information to let passengers know.
Preconditions	Logged in
Flow of events	1- User clicks button to submit a ride 2- Ride form interface is shown 3- User fills the ride form with the correct info 4- The system checks if the information are correct 5- The system saves the information and posts the ride
Alternative flow	3.b- Wrong information -Go back to 2- 3.b.1 Error message is shown. 6.User modifies the ride. 7.User cancels the ride.
Special Requirements	Internet connection on
Post conditions	A ride post is posted.

Table 2 Post a ride case

3.1.3. View user profile

Name	View user profile
Actors	Driver/Passenger/Admin
Objective	An application user can click on a user's profile through a ride post and view any user's profile public info and message them, in return any user can modify their own profile
Preconditions	Logged in.
Flow of events	1- User clicks on a user profile icon 2- System shows profile interface. 3-User views information on profile.
Alternative flow	3.b- User sends a message to another user. 1.b.1- User clicks on his own profile. 1.b.2 - User clicks modify profile. 1.b.3 - User changes profile information.
Special Requirements	Internet connection.

Table 3 View user profile case

3.1.4. Search a ride

Name	Search a ride
Actors	Passenger
Objective	A Passenger can search for a ride and view available rides with specific times and also reserve a ride and rate/report a driver after the ride is done.
Preconditions	Logged in
Flow of events	1- User clicks search button to find rides. 2- Search interface is shown. 3-User fills search information and submits. 4-System shows all available rides. 5-User browse all rides. 6-User reserves preferred ride. 7-System sends notification to ride owner. 8-Ride reservation is saved. 9-User rates Driver after the ride is done.
Alternative flow	3.b- Wrong information -Go back to 2- 3.b.1 Error message is shown 4.b No available rides are shown. 4.b.1 User can request a ride 5.b User does not find preferred ride 5.b.1 User can request a ride 9.b User reports driver
Special Requirements	Geolocation is turned on, Internet connection on
Post conditions	Reservation notification is sent to ride owner A ride reservation have been saved

Table 4 Search a ride case

3.1.5. Request a ride

Name	Request a ride
Actors	Passenger
Objective	A Passenger can request a ride if no preferred ride is found when searching for a ride.
Preconditions	Logged in, Searched for a ride.
Flow of events	1- User clicks request a ride button. 2- Request ride form is shown 3-User fills the form and submits. 4-Systems checks information. 5-System saves information and posts a ride request. 6-Drivers in the area get notification.
Alternative flow	4.b- Wrong information -Go back to 2- 4.b.1 Error message is shown
Special Requirements	Internet connection turned on.
Post conditions	Request notification is sent to drivers. A request has been saved

Table 5 Request a ride case

3.1.6. Accept/Decline a Ride

Name	Accept/Decline a Ride
Actors	Driver
Objective	A Driver can accept or decline a ride request from a passenger.
Preconditions	Logged in, Setting turned on.
Flow of events	1- Driver sets accept/decline setting on. 2- Passenger requests a ride 3-Drivers gets notification that a user wants a ride 4- A Driver accepts the request 5-System saves information and posts a new ride. 6-Passenger gets a notification.
Alternative flow	4.b- Driver rejects notification-Go back to 3-
Special Requirements	Internet connection.
Post conditions	A new ride have been saved

Table 6 Accept or Decline a ride case

4. Sequence Diagrams

4.1. Identification case

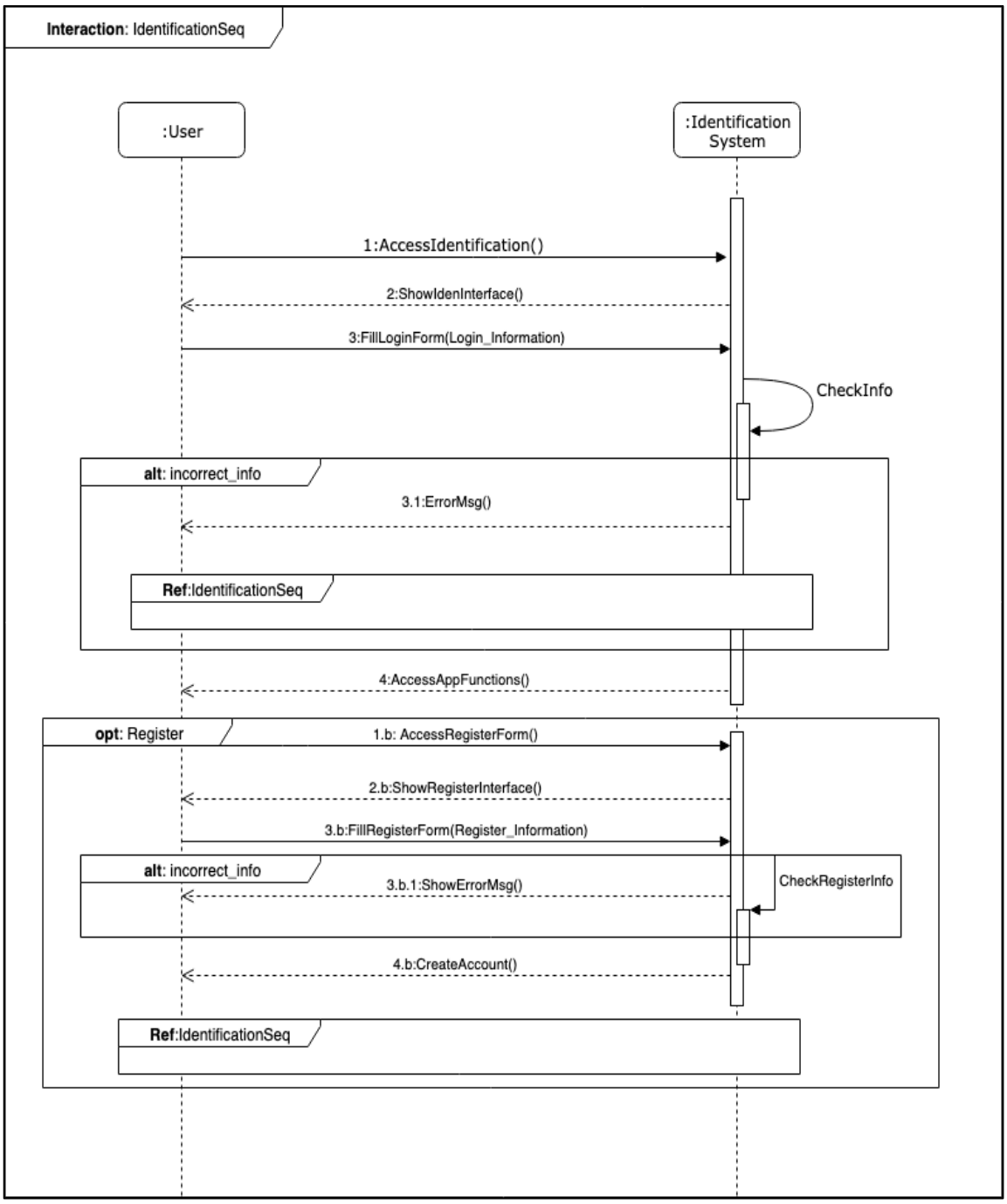

Figure 12 Identification sequence diagram

4.2. Post a ride case

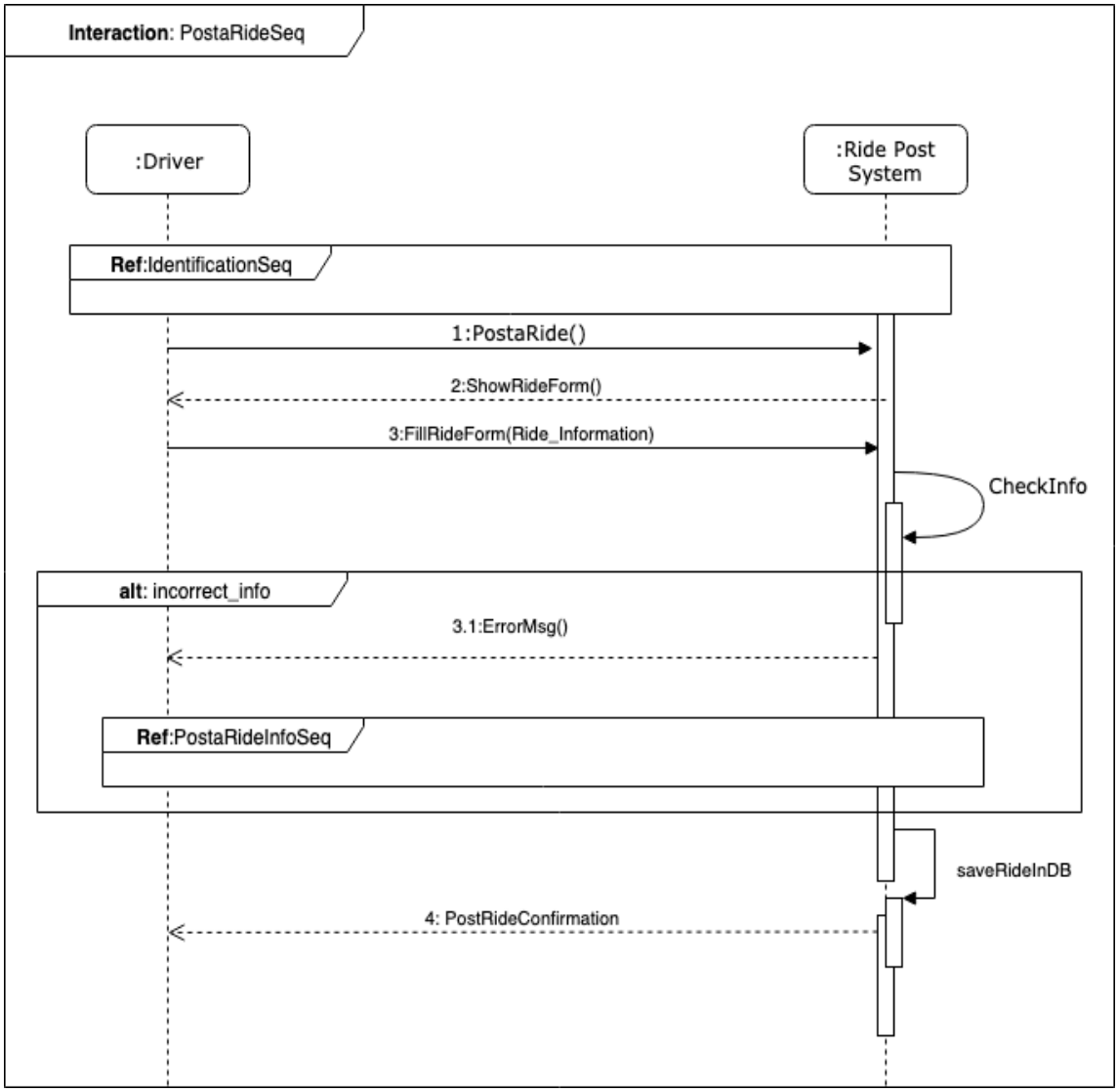

Figure 13 Post a ride sequence diagram

4.3. Search a ride case

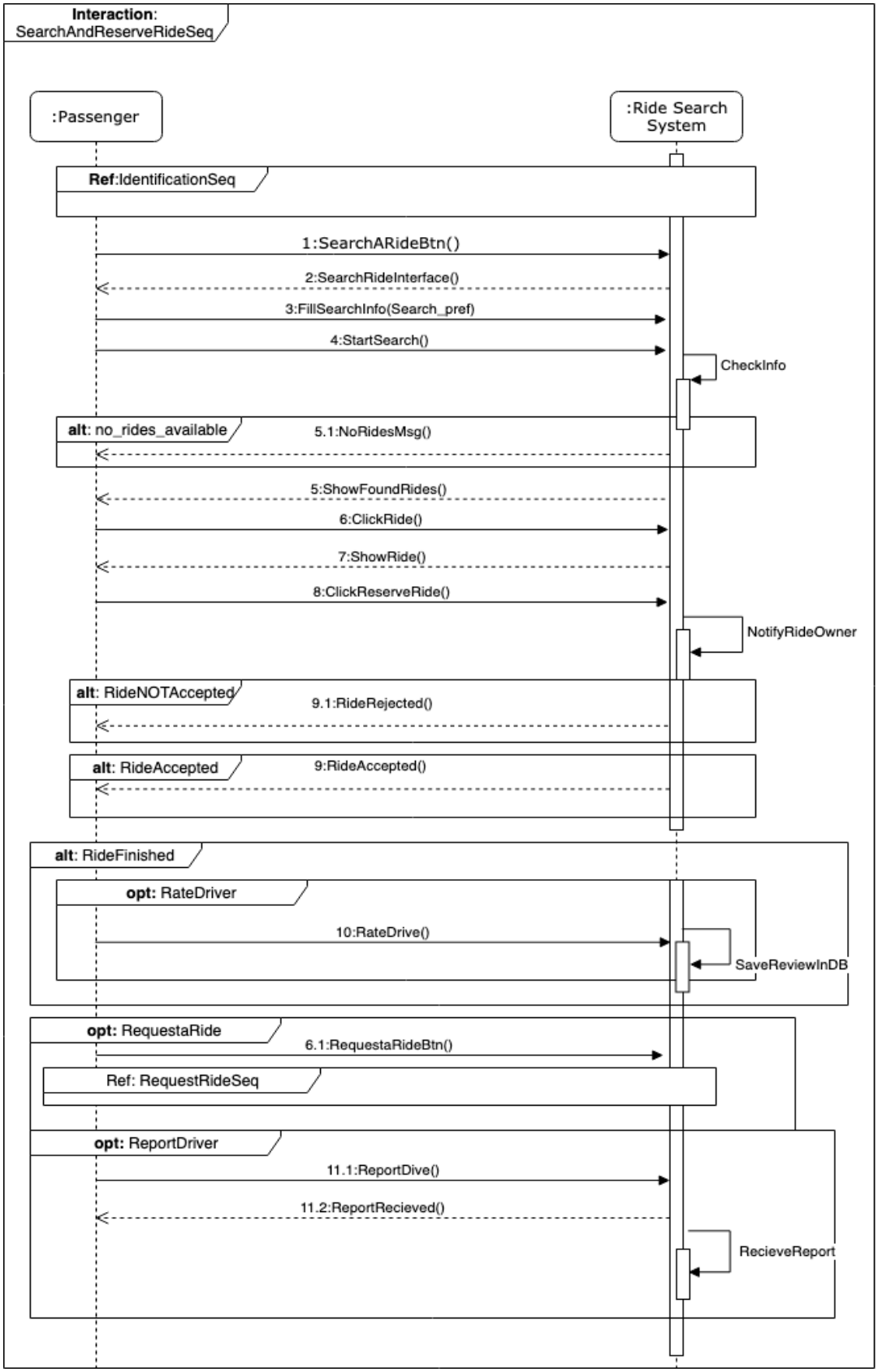

Figure 14 Search a ride sequence diagram

4.4. Request a ride case

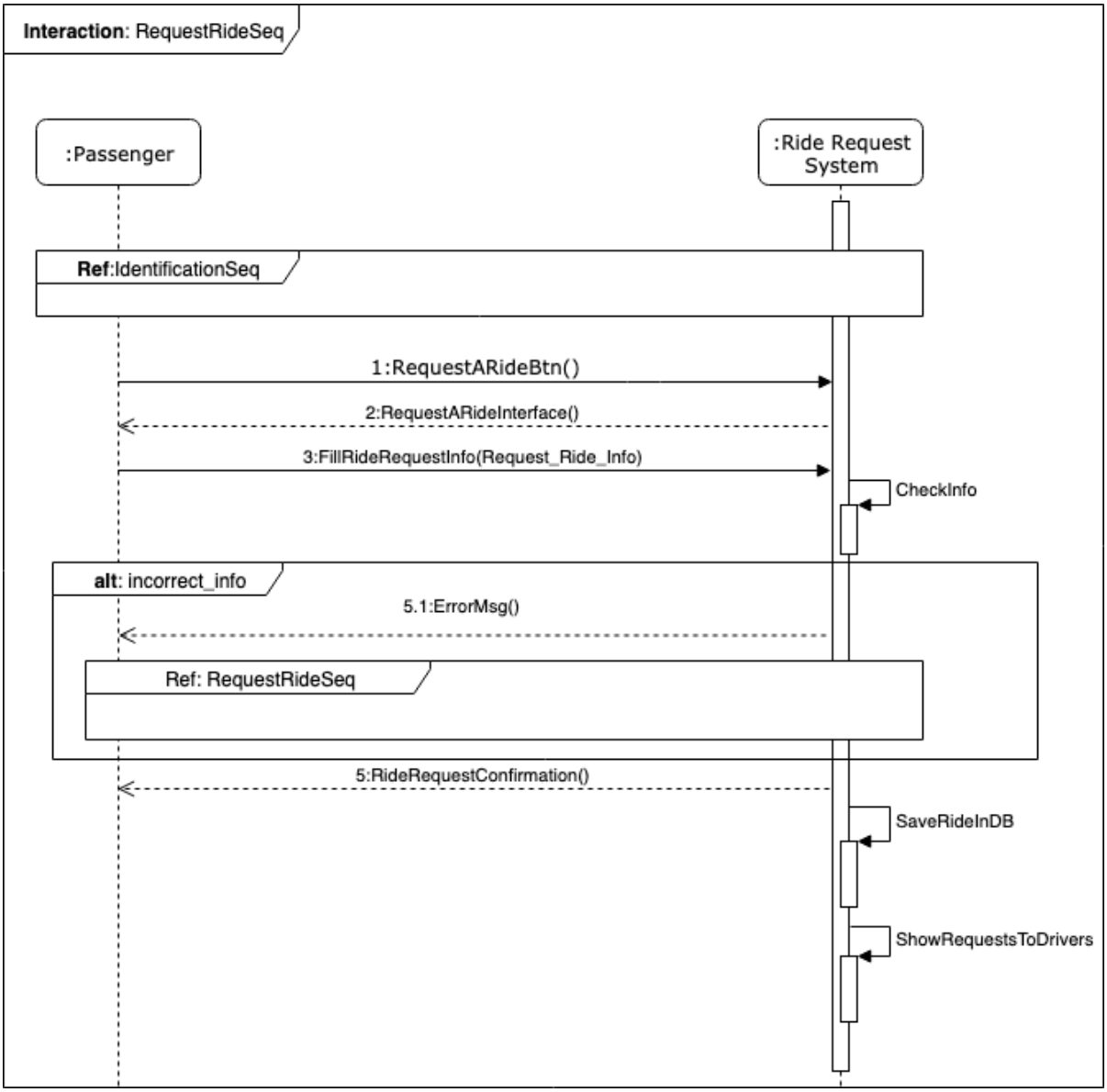

Figure 15 Request a ride sequence diagram

4.5. Accept or reject ride request case

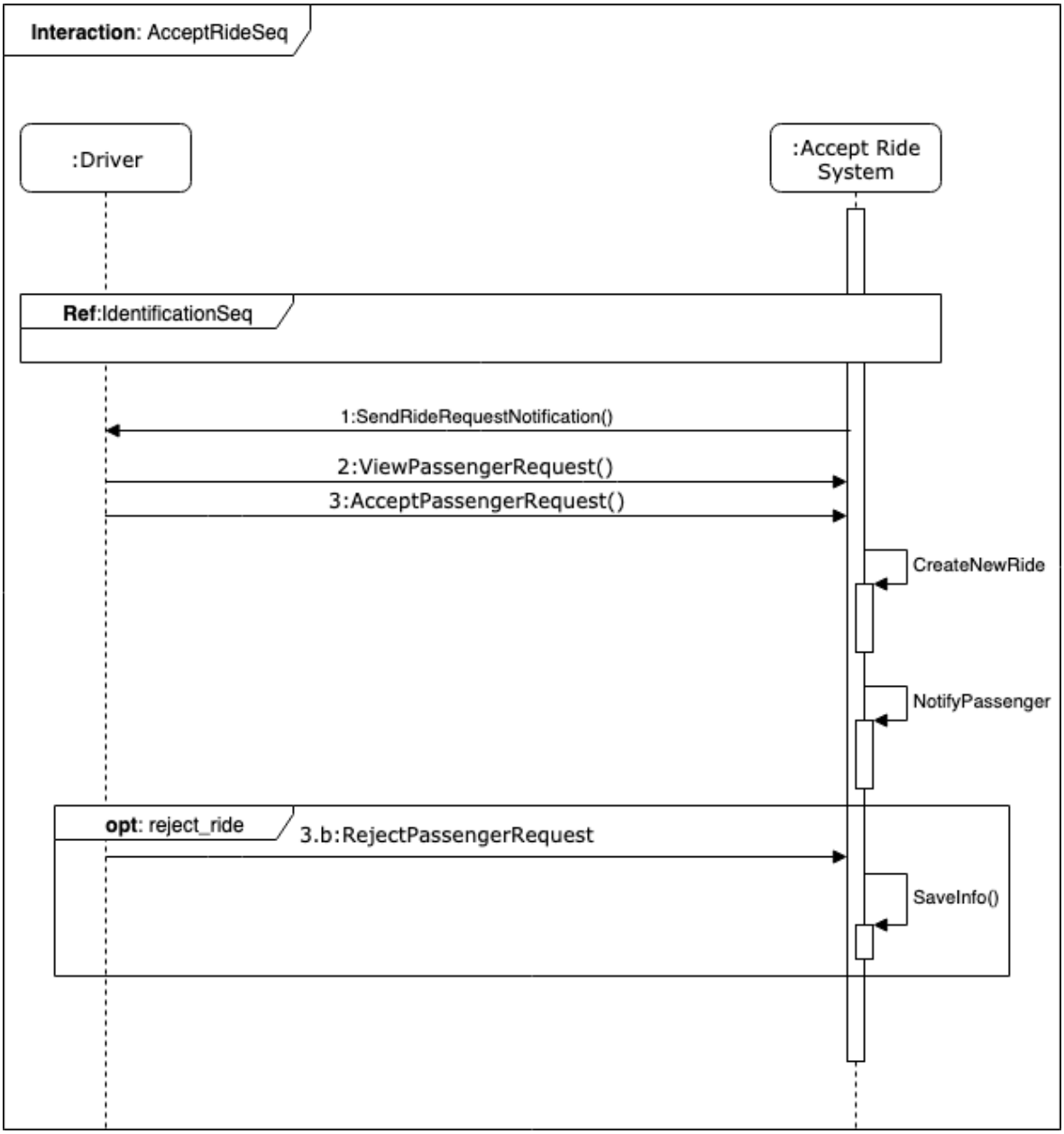

Figure 16 Accept or Reject ride sequence diagram

4.6. View Profile case

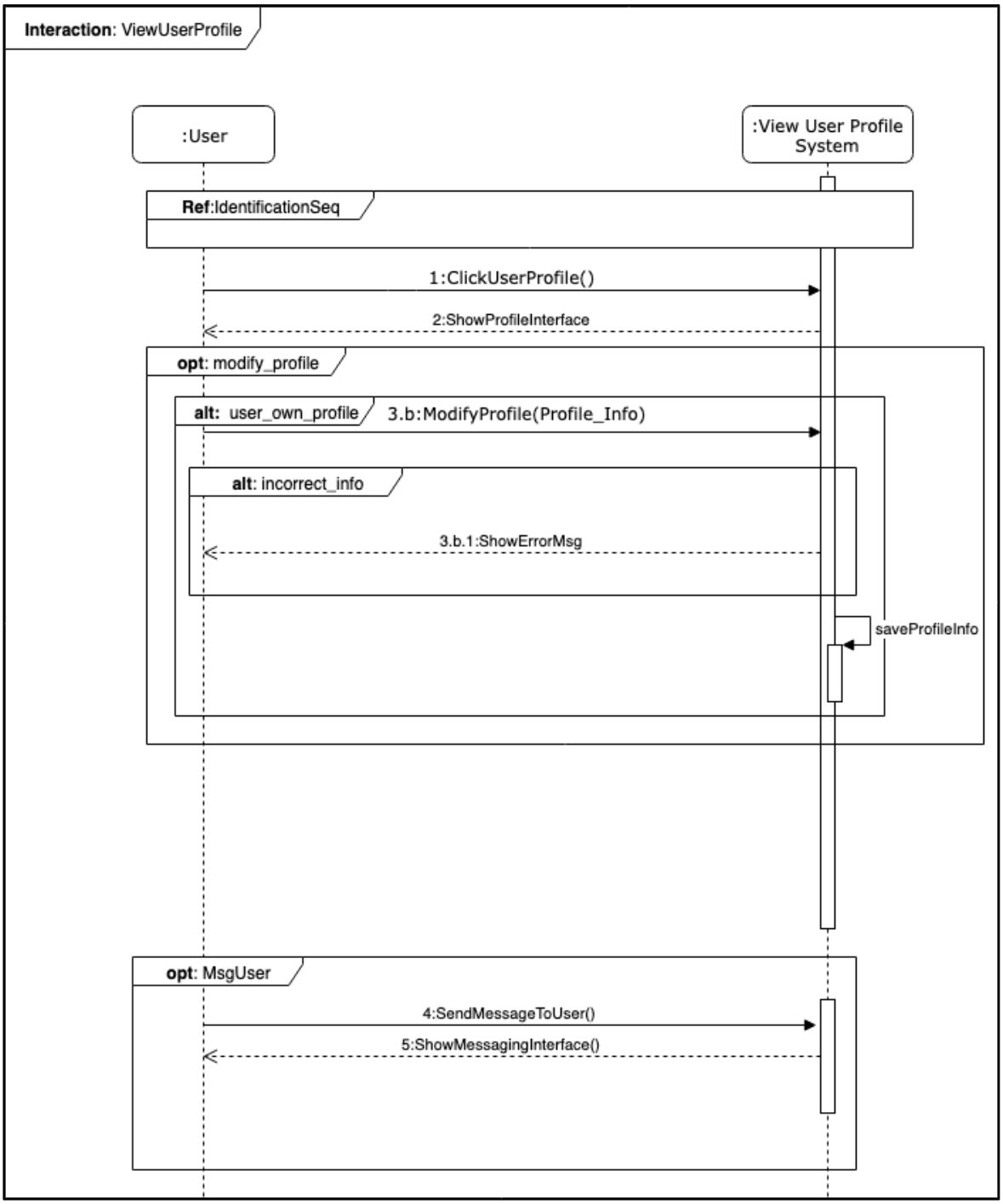

Figure 17 View Profile Sequence diagram

4.7. Disable accounts as admin case

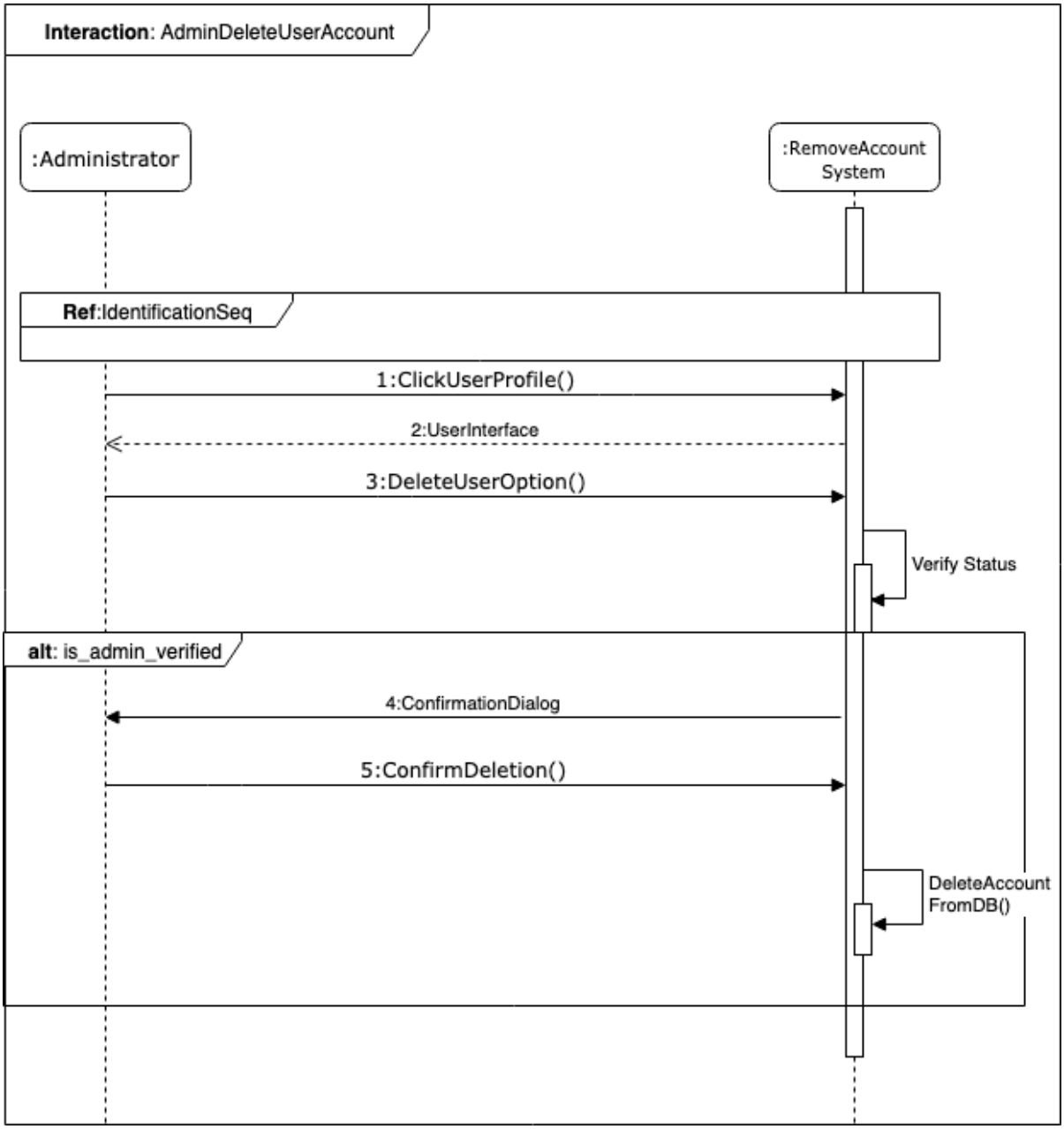

Figure 18 disable account as admin diagram

4.8. Delete Trips as admin case

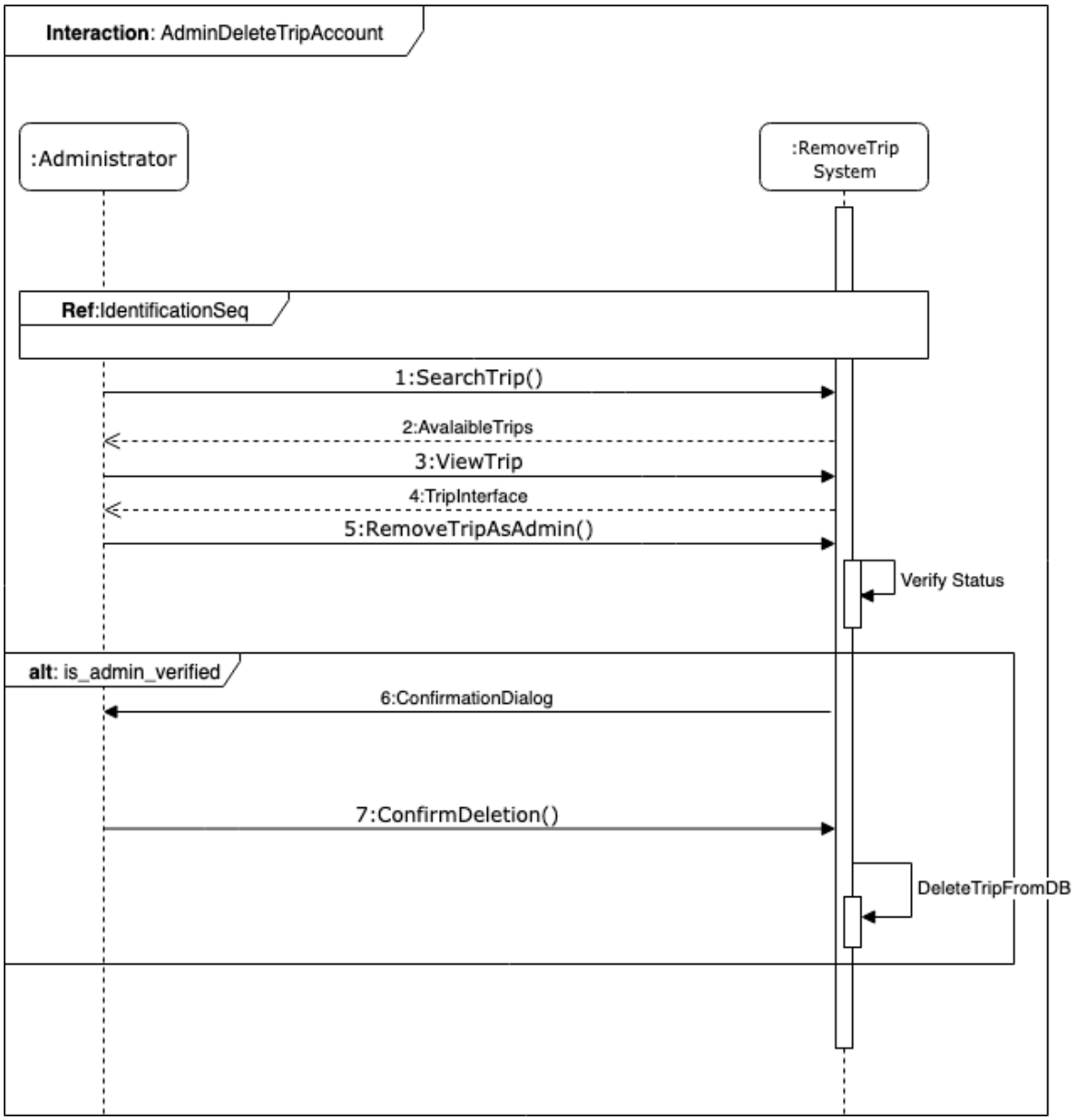

Figure 19 Delete trip as admin diagram

4.9. View reports as admin case

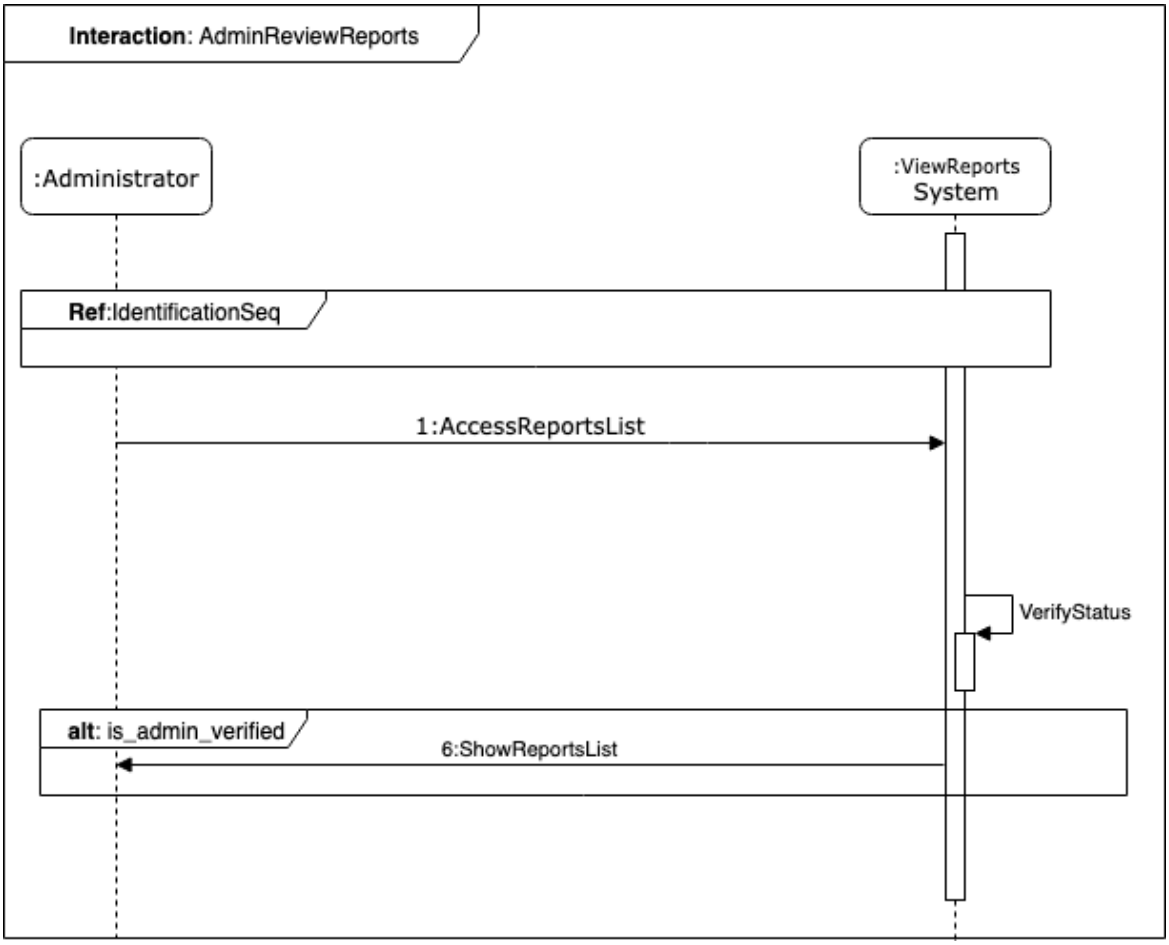

Figure 20 View reports as admin diagram

5. Activity Diagram

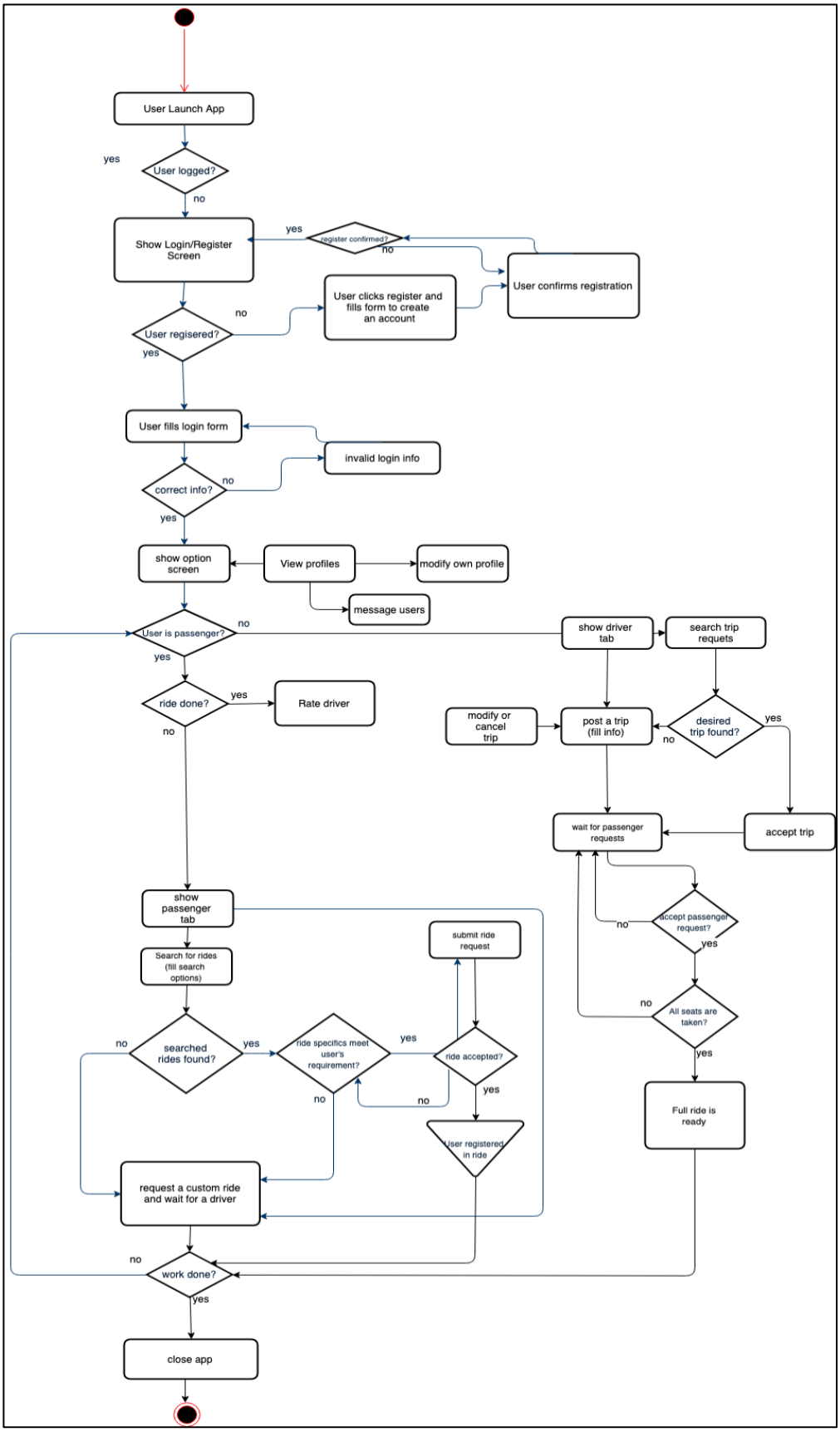

Figure 21 Activity Diagram

6. Class Diagram

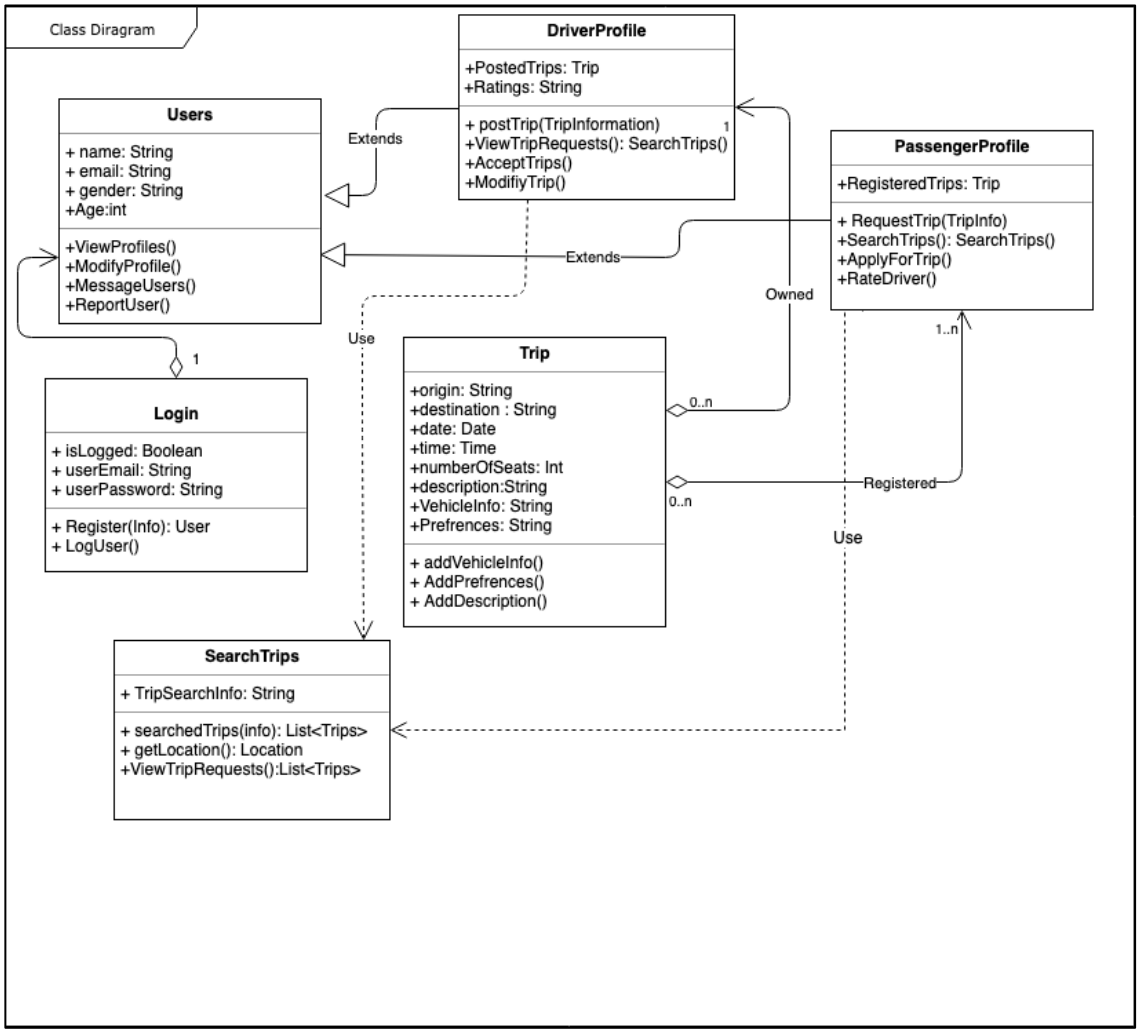

Figure 22 Class Diagram

7. Conclusion

UML class diagrams and project specifications are useful when modeling data. By accurately modeling attributes and associations of class entities, we can easily map these class diagram specifications to better understand them. Furthermore it’s hard to map all the project’s features before doing all the planning of the features because some changes may occur during the project's development some features will be added depending on the situation and availability but these specifications and modeling are the core of the application and it will be made with these specifications as the core design.

Modeling Data is a very important step of the project realization and by turning our idea into a design concept we are going to make the implementation more easy and organized. after designing our project we will turn concept into real functions which would be the implementation to an android application.

III. Chapter 03 : Implementation of the Application

1. Introduction

For the final step of the project we are going to turn the idea and concept into a real Android Application, while implementing the application we have to make sure that the application provides all the features that users require both as a Driver and as a Passenger, the application will also contain a simple Admin Interface alongside the default Firebase Admin Interface, for the application's Database we are going to use Firebase since it's easy to setup while being very powerful, Firebase will provide the application with Authentication for both email and mobile verification, and we will also use it's Real-time Database which is a NoSQL Fast Database. As for the development and application programming we'll use Android Studio with Kotlin, our main focus for this application is speed and optimization by using complex searching algorithms made from scratch and using Google Maps API to give accurate results, the other main focus is a simple and clean interface. we achieved that by following Google's material design guidelines while using 3rd party libraries to achieve a fast, slick and beautiful design. In this chapter we will go into detail about every part of the implementation and how we could achieve professional looking application with a strong and optimized backend.

2. Work Environments

2.1. Hardware platforms

For the development of this application we have used an Apple MacBook Pro (15-inch Mid 2015) it fairly powerful and could handle the usage of multiple emulators at once. Full Specifications:

- Processor : 2.2Ghz Intel Core i7
- Memory : 16GB 1600 MHz DDR3
- Graphics : Intel Iris Pro 1500MB
- SSD Drive : 256GB

For the testing of the application on a real device we have used an Android Phone OnePlus 6

2.2. Software platforms

As for the software part of the work environment we have used several tools and frameworks alongside different versions of android.

2.2.1. Work Tools:

- **Operating System** : MacOS Mojave
- **IDE** : Android Studio 3.2.1
- **VCS (Version Control System)** : GitHub
- **Emulators** : Pixel 3 Android 9 / Nexus 5 Android 6
- **Real Device OS**: Android 9 Pie (OnePlus 6)
- **Backend Management**: Firebase Platform
- **Places and Map Tools**: Google Cloud Platform

2.2.2. Programming languages:

- **Application Programming** : Kotlin

Motivation: Kotlin code is inherently safer than Java code because it prevents common programming mistakes by design, resulting in fewer system failures and application crashes. When using Java, certain error causes are more likely to occur again. Kotlin is also the new official programming language for Android.

- **Layout Design** : XML
- **Database Language** : NoSQL
- **Others:** JavaScript for Firebase Cloud Functions, Java for a few classes in Android Studio

2.2.3. Frameworks used:

- **Firebase Auth** : this framework that lets you implement easy Authentication in this application email and phone number authentication was used.

- **Firebase Real-time Database** : this framework is used as our main Database for this application it lets you read and write data from firebase in a NoSQL structure.
- **Firebase Storage** : this framework is used to upload and download data such as photos and videos, this is used to store user pictures for our database.
- **Firebase Messaging** : a framework that makes real time messaging easy and possible using firebase, it's also used for push notifications, this used for our messaging function between users in our application.
- **Google Maps API** : a great API used for MapView that enables the application to show places on the map like the origin and destination and the way between them, it's used in the map to trace the destination on the map for users.
- **Google Places API** : a very powerful API used to give information about places in the application and used to AutoComplete search queries, it's used to autocomplete search queries for our users for easier searching, it's also used to optimize our search function.
- **Material Design Library** : Material Design Library with all of its components such as Material Buttons, Material EditText etc..
- **RuntimePermission** : a small library to help manage Android Permissions with no problems.
- **DxLoadingButton** : a small library with a loading button used for Authentication.
- **StfalconImageViewer** : a simple and customizable Android full-screen image viewer with shared image transition support, "pinch to zoom" and "swipe to dismiss" gestures.
- **Picasso** : A powerful image downloading and caching library for Android.
- **SpinnerDatePicker** : A styleable DatePicker for Android using the old spinner style (NumberPickers) used for Birthday date picker.
- **CircleImageView** : A circular ImageView for Android.
- **Tooltip** : Simple to use customizable Android Tooltips library based on PopupWindow.
- **EasyValidation** : A text and input validation library in Kotlin for Android. Used to validate user input info.
- **Retrofit** : Type-safe HTTP for Android and Java.

2.3. Database Management System

To manage the backend of our application we decided to use Firebase Real-time Database alongside with Firebase Auth and Firebase Storage.

- **Firebase Real-time Database :**

it's a database structured in a NoSQL way, it's usage is very easy and very fast, we have used this database to store User objects which contain all the information about the user such as name, email, birthday etc., we have also used it to store Trip and TripRequest objects and messages and meta-data for messages and notifications (Chatlist and Notification Tokens).

example of the user node in the database:

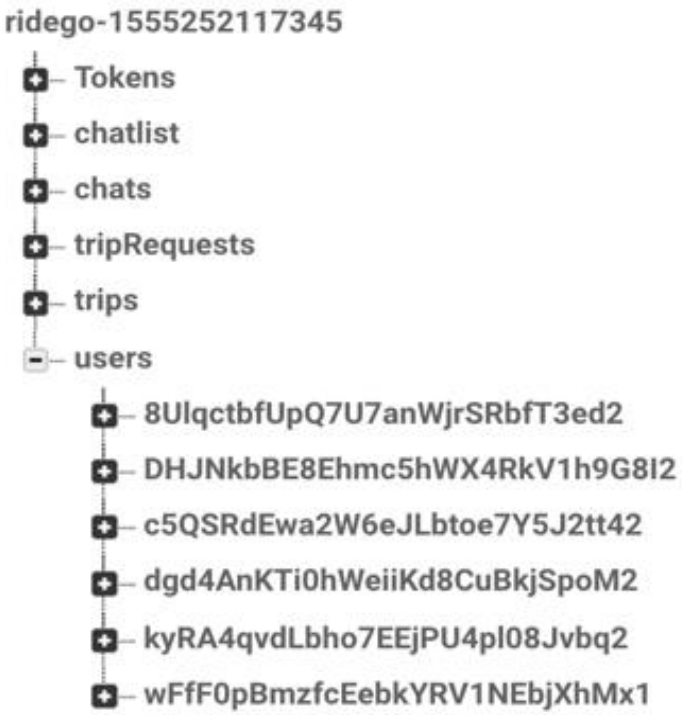

Figure 23 example of a user mode in the database

- **Real-time Database Usage:**

For the usage of this Database a separate Database class was made on Kotlin this class contains all Database related functions the functions require a listener interface that has *onStart(), onSuccess(), onFailed()*, the interface is used to listen to changes in other classes.

The reason for creating a separate class with a listener interface is to make migration to another Database easier. If we ever decide to switch to another database we will only change the content of the Database class which would make changes faster, more professional and effective.

Firebase's Real-time Database allows you to use different query methods to fetch data from its Database:
The first method is using *addListenerForSingleValueEvent*, this method listens to changes in data for one time and does not trigger until it's called again, this is good for fetching data only once.

The second method is using *addValueEventListener*, this method listens to data every time it changes and it also triggers every time it changes this is good for things like messages and real time updates.
In our real.

We have decided to use *addListenerForSingleValueEvent* for most of our data fetching, the reason is each time data has been read, Firebase charges for it, which means it the less data is fetched the better, for this we have avoided using *addValueEventListener* Instead we fetch the data once and save it locally, when the data is changed the user is required to refresh the page to get updates. However in cases where it's important to get updates in real time like in messages we have used *addValueEventListener.*

- **Firebase Storage:**

Firebase Storage is used to store images for our application, photos being stored are user profile pictures, and vehicle images.

- **Storage Usage:**

To store images in our storage and link it to the real-time database,
First we check if the image is correct, we upload the image into the storage database then get the link and link it to the real time database.

For example if a user uploads a profile picture, the image will be uploaded and a download link is returned, once that happens the link is going to be saved to the user's node in the RealTime Database under *profileURL*, that URL will contain the link of the image which is going to be linked to the user.

- **Firebase Authentication:**

Firebase Authentication is a very fast and secure way to sign users into our application it is used to Login and Register users into our database in our case we are using the email and SMS verification, a user must verify his phone number to complete the registration.

- **Authentication Usage:**

When a user registers, his basic information (Name, Email, Password) are registered in the Auth Database but not the RealTime Database, after that users are welcomed with a finish registration activity, in this activity they have to verify their phone number using SMS verification, this method is used to prevent spam and multiple account creations, after verification and filling other information user data is saved on the RealTime Database. As for the login the system checks data and logs users if it's correct.

3. Presentation of the Application's Interfaces

3.1. Presentation of the Application's Logo

The logo was designed in an original and a professional way that meet's the *Google Play logo guidelines*, the logo was made using Adobe Photoshop CC 2019, all the icons are not copyrighted and the colors match the application's theme as scene in Figure 24.

Figure 24 design of the application's logo in Android Studio

3.2. Authentication Interfaces

The authentication interface is made as simple as possible while also being very beautiful and professional, it utilizes Firebase Auth as a backend which allows for a secure connection to the database plus encrypted passwords for extra security, this interface also contains options to recover password in case the user has forgotten their password. This interface is made out of 3 layouts and activities, LoginActivity, RegisterActivity, RegisterExtraActivity.

3.2.1. Login Interface:

This is the interface that shows up to the user when he's not logged in into the application, this interface is composed of the application's name and a logo alongside an Email and a Password custom EditText and a Custom Login Button, the interface is also composed of "Forgot your password" option which lets the user recover the password by referring them into a special URL to change the password, there's also a "Create one" option which lets the user create an account by sending them into the RegisterActivity.
The EditText fields are error friendly and would show the name of the input error that is thrown.

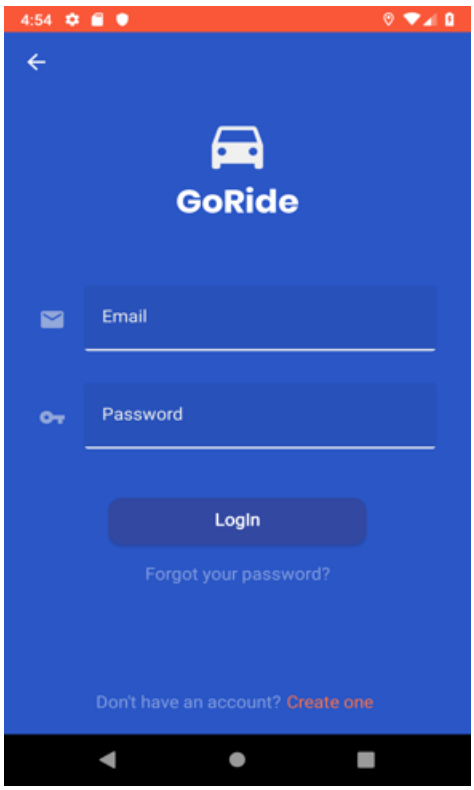

Figure 25 normal login Interface

Figure 26 login Interface with error

3.2.2. Register Interface:

This interface functions the same way as the Login Interface the only difference being instead of login the user into MainActivity this takes the user into RegisterExtraActivity, there's another EditText field for the user's name, and an option to go back to the login activity. There are error checks for weak passwords and bad email formats as well, after the registration is successful a verification email is sent to the user's email and he's taken into the next activity to finish the registration.

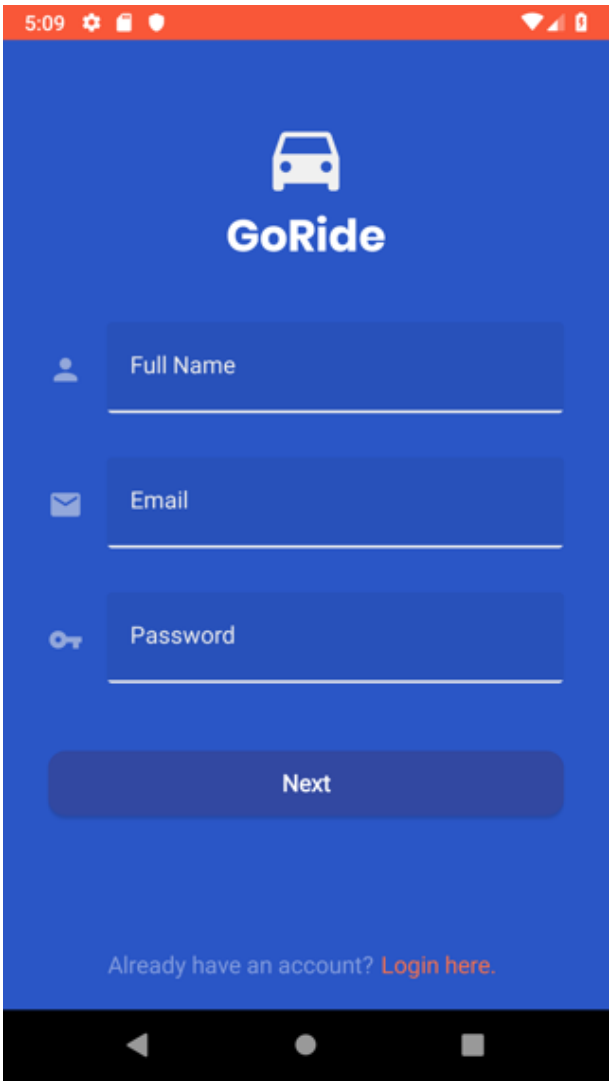

Figure 27 Register Interface

3.2.3. Extra Registration Interface:

This interface is used by the user to complete the registration by filling the forms in a ViewFlipper, the user is also required to verify his Phone Number to avoid spam and to add extra security, phone number verification is managed by Firebase, once the verification

is successful the user is supposed to fill information such as Birthday, Gender, Type, Description, Profile Picture etc. all the fields contain error verification inputs. After the user completes the forms and accepts Terms of Service he's taken to the Main Interface of the application depending on what user type he is. If the user's birthday is less than 60 years he cannot register as a passenger, he must be 60+ to register as a passenger and 18+ to register as a driver.

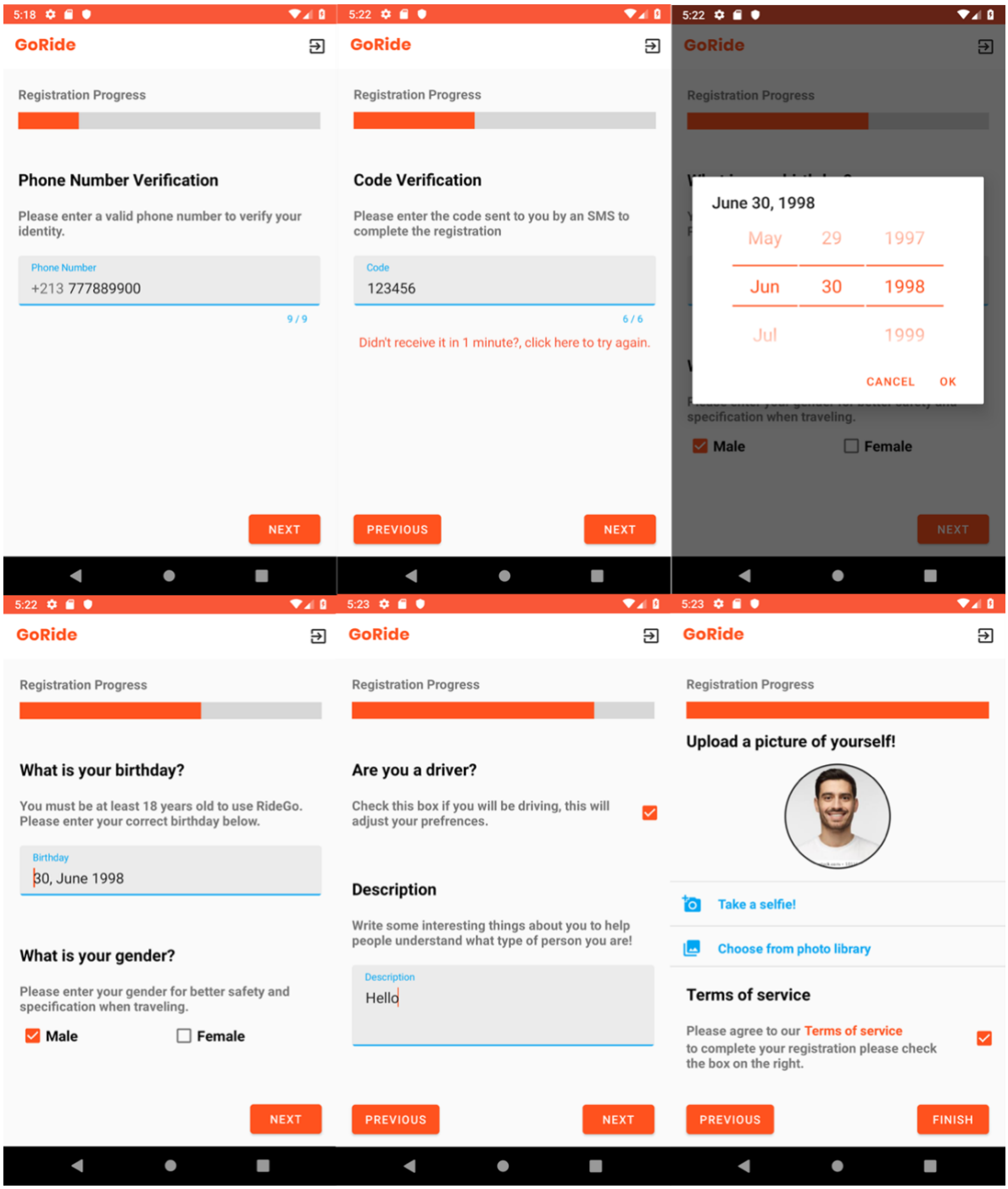

Figure 28 Extra Registration Interface

3.3. Driver Interfaces

3.3.1. Main Driver Interface

This is the main driver's interface once the user is logged, this is what the user see's once they open the application, this interface is a Fragment of MainActivity which contains 2 Fragments (Driver, Passenger), the driver can use the passenger fragment and options however they cannot use its primary functions such as book trips. The Driver Fragment contains a RecyclerView which contains trip objects posted by the driver, once a driver posts a new trip it will be visible in this interface after refreshing the fragment.

The main interface also contains an Add Trip FloatingActionButton, a notification and a messages button in the AppBar and a DrawerLayout which contains Profile, Settings, Logout buttons.

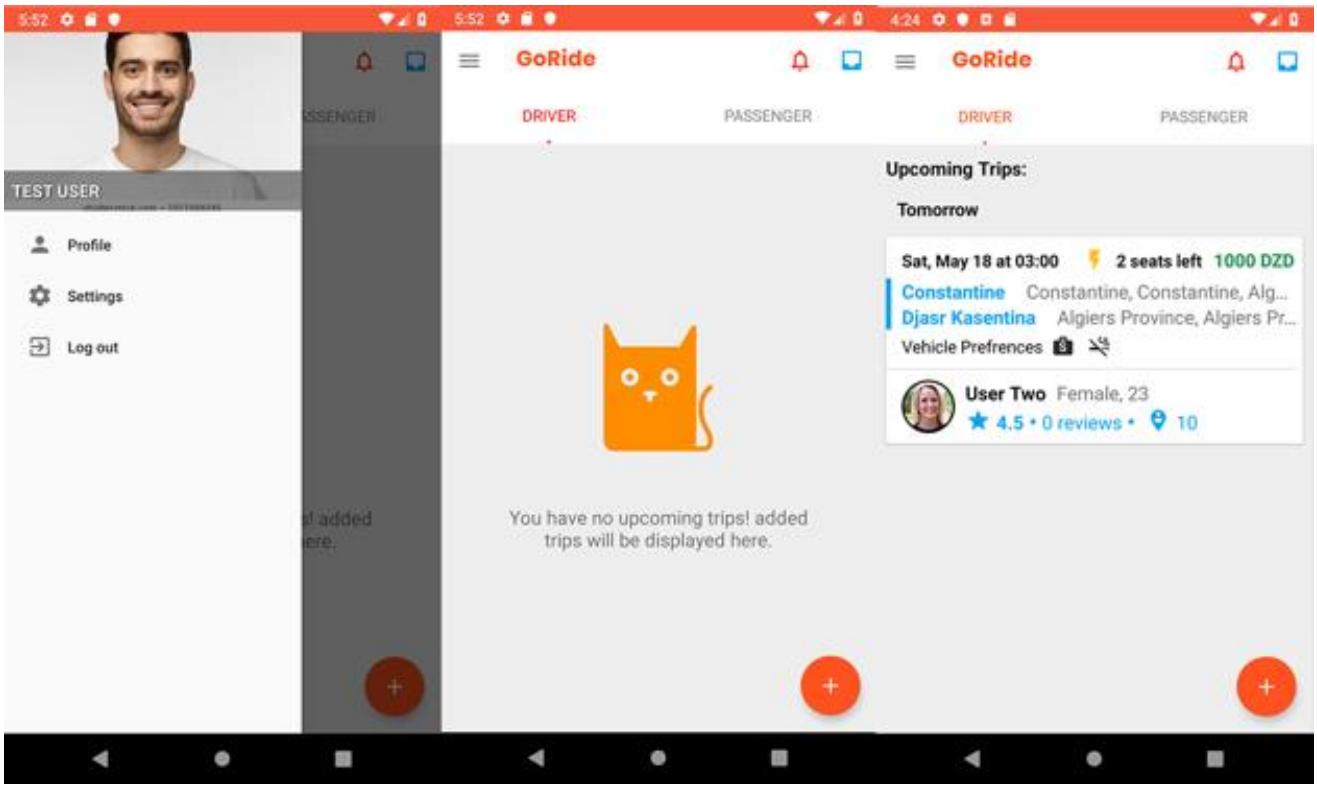

Figure 29 Main Driver Interface.

3.3.2. Post a trip Interface

This interface is used by the driver to fill the form required to post a trip for Passengers to see and search for, this form is extremely customizable and professional, by using Google Places API in this interface we have created a very organized and efficient

Algorithm to optimize the results when Passengers search for a trip, When the driver picks an origin and a destination using Places AutoComplete API, the algorithm uses a combination of Geolocation and Places API to find the Full Address, City and Sub City, then it converts the city into a number and saves all the information in the Trip object for it to be saved into the database later.

For example if the driver picks “University of Constantine 2” as his destination, the Algorithm takes the address and looks up the Full Address, the city and Sub City, the results of the lookup would be “University of Constantine 2” as the full address, “Ali Mendjli” as the Sub City and “Constantine” as the City, after that the algorithm decodes the city into its code number which is “25” in this case. All this information will be saved into the Trip Object for us to use later.

The Motivation behind this algorithm was to make searching for trips more efficient and fast and less resource taking, this was achieved by decoding both the origin and destination cities into codes and storing the Trip object under a special composed of “OriginCode_DestinationCode”

For example if a user picks “Algiers” as an Origin and “University of Constantine 2” as a Destination, after the form is completed correctly the trip is going to be saved into the NoSQL Database under this path : *“trips/**16_25**/{trip_id}/...”* This would make searching for trips so much easier since when the Passenger searches for trip, he’s only going to look under the {originCode_destinationCode} tree, so all of the unrelated trip results won’t be downloaded nor fetched, which is great for performance.

This Interface contains 9 sections each section gives the Trip more details to help users understand each other more, the 9 sections being:

- **Trip Form**

This section contains origin and destination fields alongside with the option to add multiple stops on the way.

- **Ride Schedule**

This section contains the date and time for the trip, a DatePickerDialogue and a TimePickerDialogue is used for this, it includes error checks to prevent bad formatting and drivers must pick a date that is at least a day after their today’s date.

- **Vehicle Details**

This section is optional and it contains optional details and information about the driver’s vehicle this includes an image of the vehicle , the vehicle type (car, bus , van etc.), vehicle

color, year, license plate. All information in this section is used to help the passenger better identify the vehicle.

- **Trip Preferences**

This section is a preferences section to help driver's and passenger's be comfortable with their choices, it contains a luggage option which is composed of No Luggage, Small Luggage, Medium Luggage, Large Luggage.

This section also contains other preferences such as Smoking, Pets etc.

- **Empty Seats**

This section is used to pick how many seats available in the vehicle an option from 1 to 7 is given.

- **Price per Seat**

This section is used to pick a price for the trip, the driver controls the trip and a label asking them to keep the price cheap is shown in this section.

- **Booking Preferences**

This section gives the drivers 2 option to either accept drivers automatically without letting the driver review them, or letting the drivers review the requests gives them the option to accept or decline whoever they won't, the first option is called Request to book and the second one is called Instant Book.

- **Policies**

This section contains information about the application's policies for the driver's the information are there to let the driver his rights and rules so no one gets confused or scammed, it's used to protect both passengers and drivers.

- **Trip Description**

This section is used to provide the driver an option to write whatever they want to let passengers know about things or preferences not available in this form.

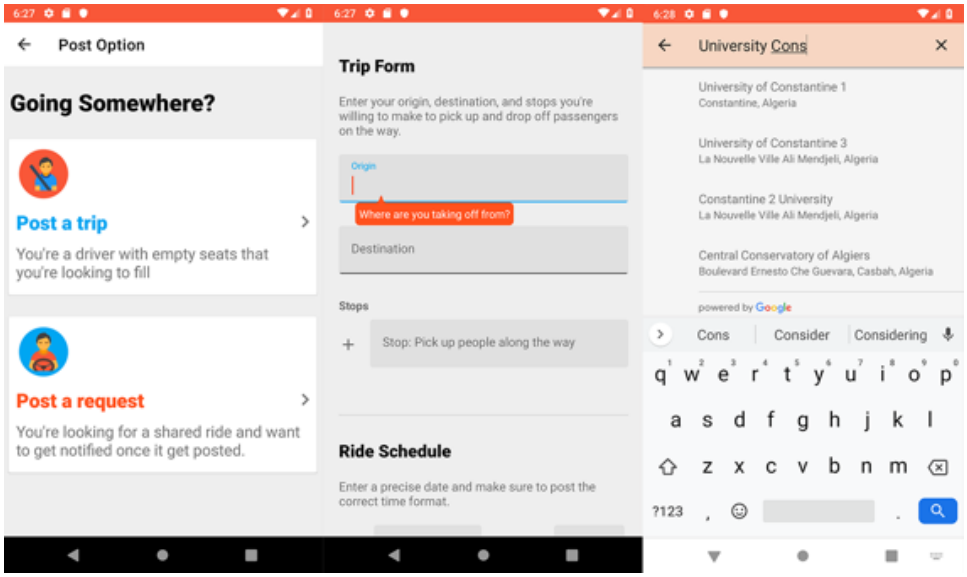

Figure 30 Post Options, Trip Form and Places AutoComplete Interfaces

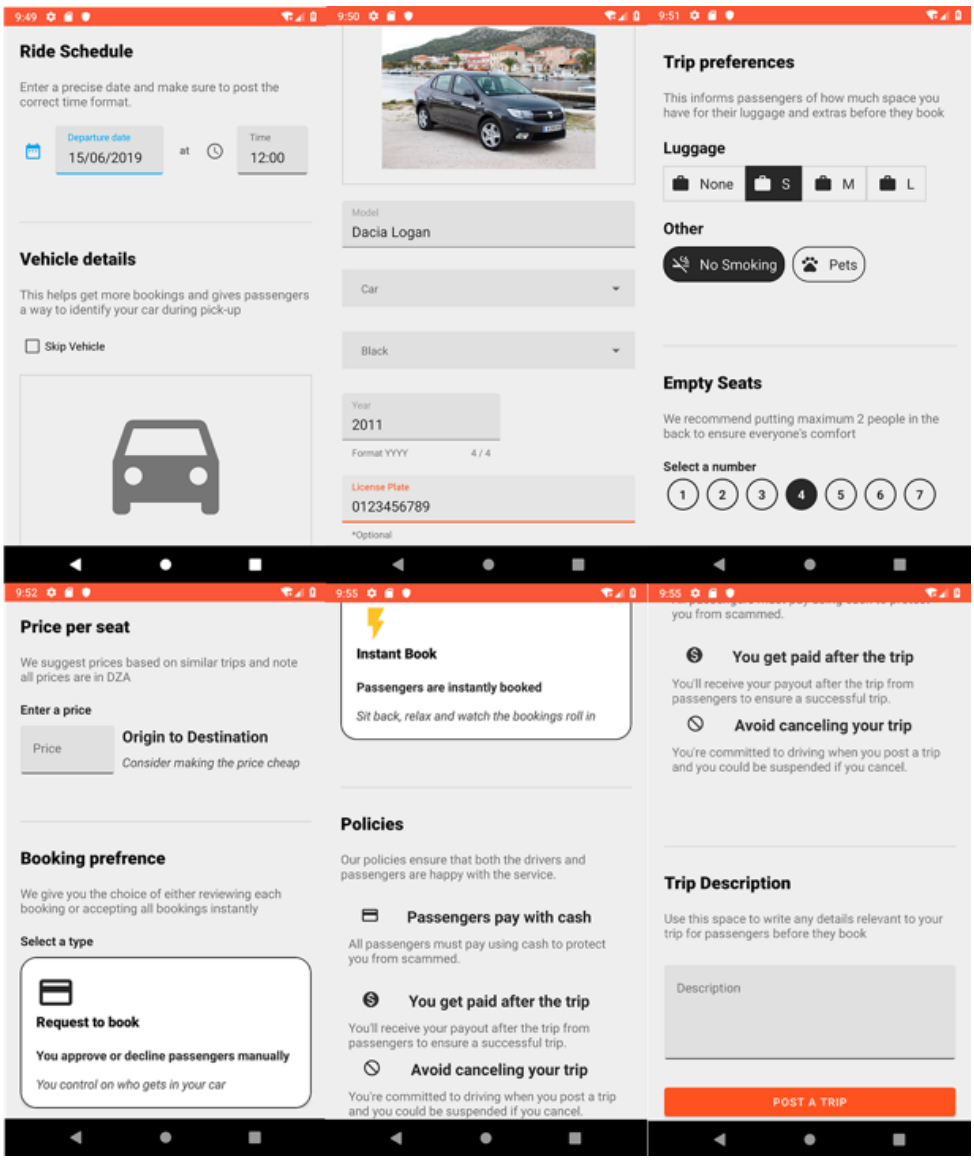

Figure 31 Post a trip sections Interface

After the user fills the required fields inside the form and clicks on “POST A TRIP” The system checks for input errors and if everything is correct the system will first upload the vehicle image if it’s present and then uploads all the data into the database, after all of this is done a Dialogue is shown to the user to let them know the operation is complete.

3.3.3. View Trip Interface

This interface is shown when the driver clicks the trip view inside the DriverFragment it opens a separate Activity showing a well-designed page that contains all the important information about the trip that the driver saved, it also contains a menu that gives the driver the possibility to Modify, Cancel or Report the trip in case of an error happening.

This activity and interface is the main activity and interface for viewing trips, it shows the date and time of the trip alongside with the sub cities and full addresses and stops if they exist, then it shows how many seats left (All Seats – People Booked) and shows the price and description, you also find a section which contains information about the driver, such as their name, profile picture and some ratings and stats, under that there’s the Vehicle Details section which contains information about the vehicle, this section is empty if the driver skips it while posting the trip. At the end of this activity you’ll find a MapView of the path between the origin and destination, this was put to help passengers know the path they’re going to take for more clear information.

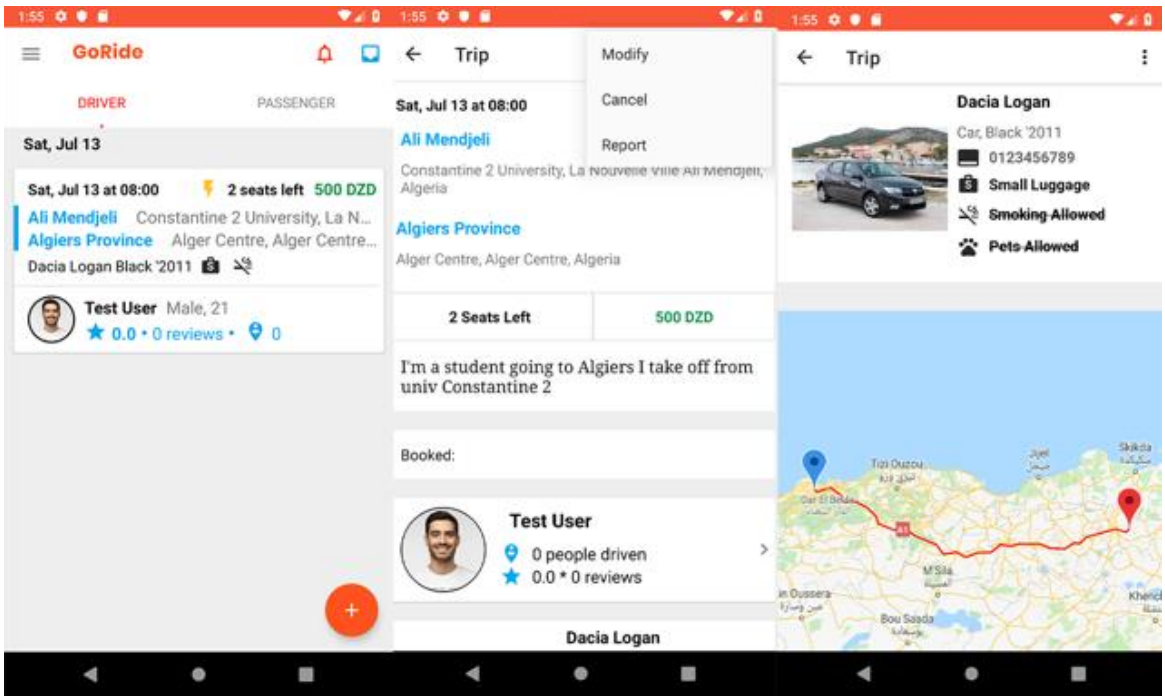

Figure 32 View Trip Interface for a driver.

3.3.4. Modify Trip Interface

This interface is used by the driver to modify a trip in case they missed something or in case they want to add something, when a driver picks the option to modify a trip he's taking into the post a trip interface but with the Origin and Destination EditText field disabled which means the driver can change every other information except the origin and destination, after this operation is done a notification is sent to each booked passenger.

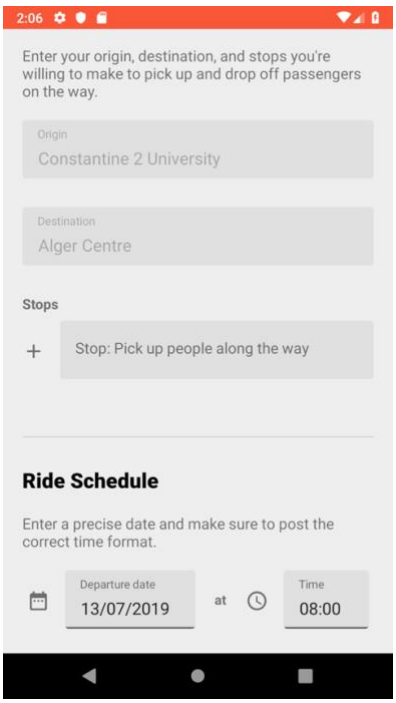

Figure 33 Modify a Trip Interface

3.3.5. Cancel Trip Interface

This interface is a Dialogue shown to the user with a warning asking if they really want to delete or cancel the trip ,the warning tells the driver that he might be suspended if he cancels too many trips, the reason for this is canceling trips may hurt passengers in a bad way, so we try to prevent that as much as possible, if the user decides to cancel anyways the trip is removed from the database and a notification is sent to all booked users letting them know their trip is canceled.

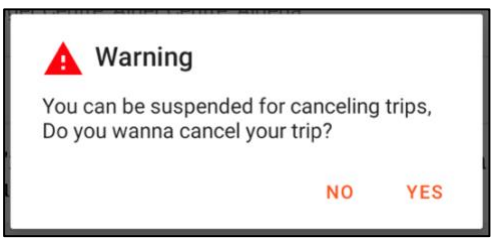

Figure 34 Cancel Trip Interface

3.3.6. Search And Accept Trip Requests Interfaces

- **Search Trip Request Interface**

Using this interface the driver can search for trips but cannot join other driver's trips however he can also see trip requests from passengers and accept them , the interface contains an origin and destination field and a date field, this interface can accessed by clicking the search FloatingActionButton, it also contains a MapView to show the path between origin and the destination. A red line is drawn once the path is determined. This search uses an efficient and optimized algorithm that is going to be well explained in the search interface for passengers.

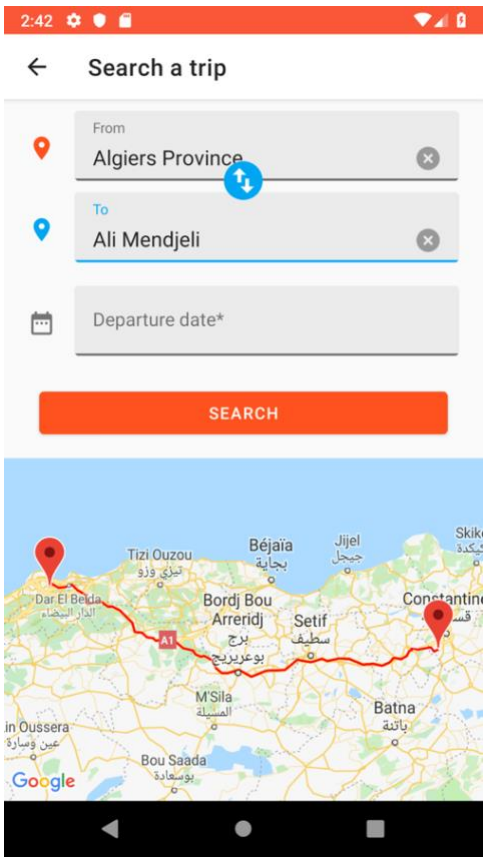

Figure 35 Search a Trip or Trip Request Interface

After the user enters this information a Trip Search Data object is sent to the next Activity containing extra information such as City, sub city, date, full address, origin code etc. The user is then taken to a ResultsActivty which contains 2 Fragments the first being the normal Trips list posted by drivers, in this case the driver can view them but cannot book as a driver, however the second fragment is a Trip Requests fragment which contains Trip Requests objects.

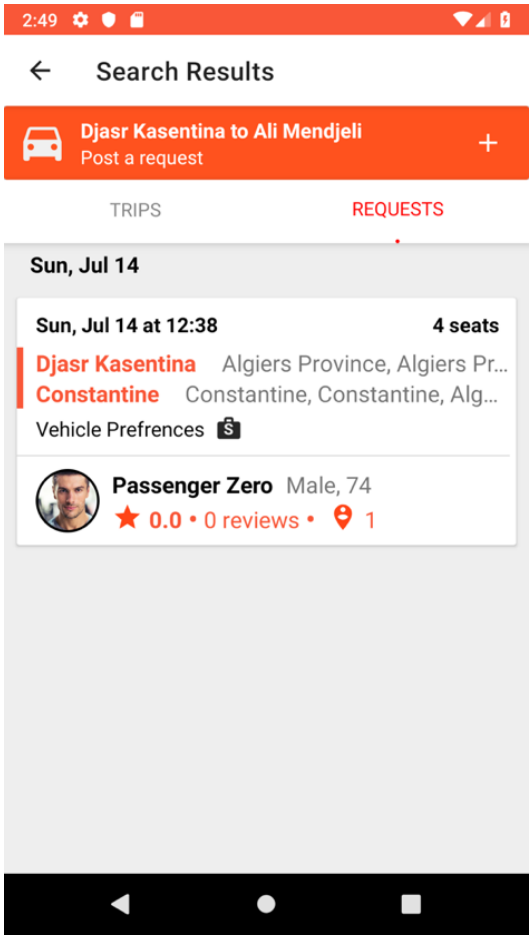

Figure 36 Search Results Interface.

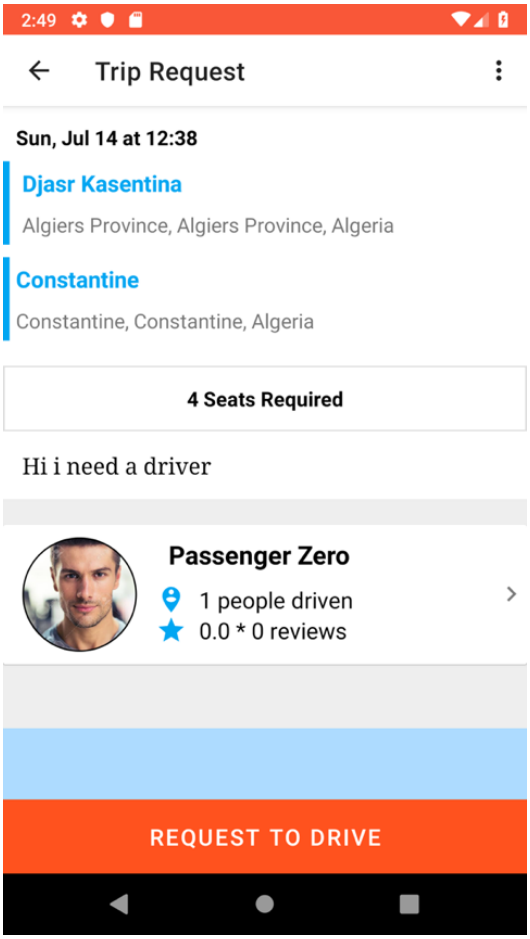

Figure 37 Trip Request Interface.

- **Accepting Drive Request**

the driver can request to drive these Requests by sending a request to drive notification to the passenger. The requests stays pending until the passengers accepts it.

If the passenger accepts the request, a notification is sent to the Driver asking him to post a trip with the same specifications as the Trip Request and after the drivers fills the post trip form the owner of the request will be automatically booked into the trip.

The Trip Request is then removed from the database and the new Trip is saved into the database instead.

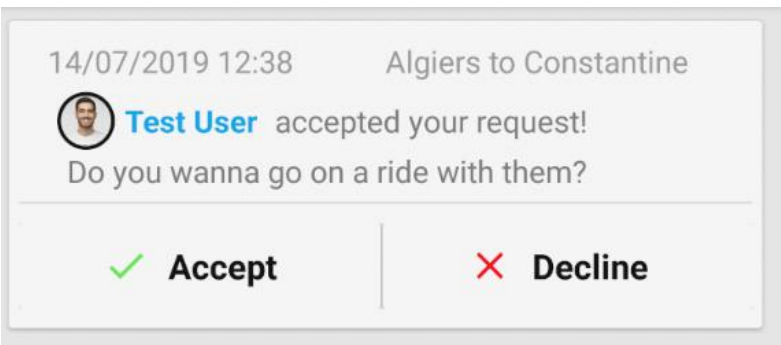

Figure 38 Passenger Drive Request (Passenger View).

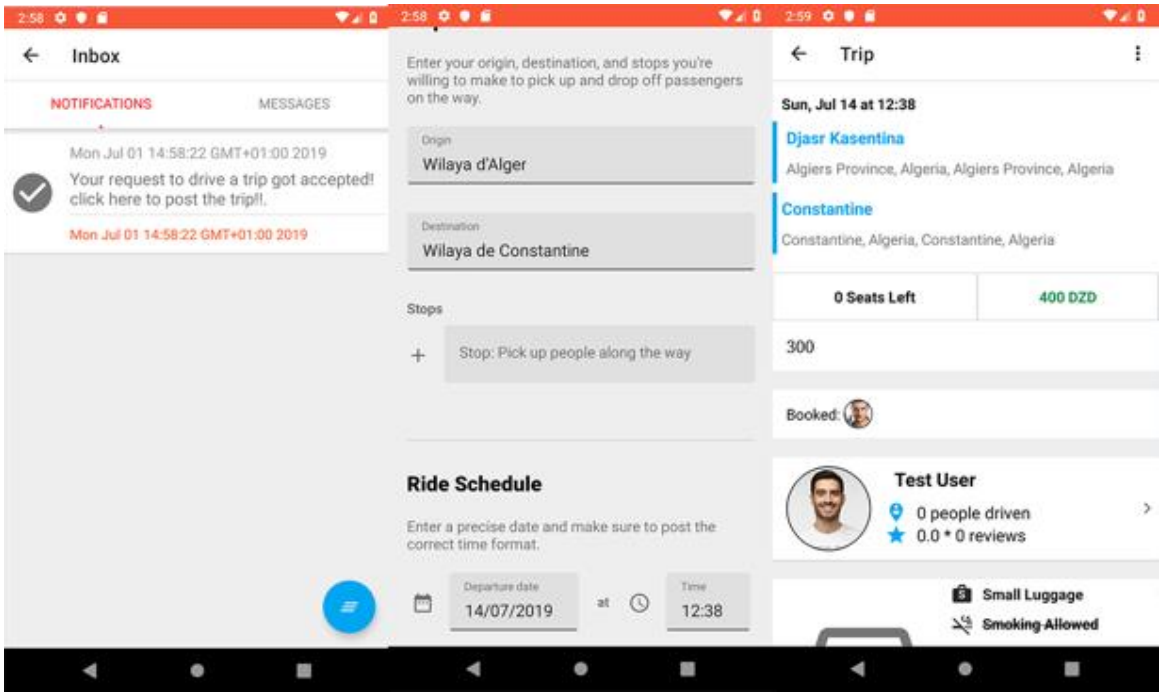

Figure 39 Accept Trip Request And Post it as a Trip Interfaces

3.3.7. Booking Requests Interface

In this Interface the driver decides to accept or decline users when they book into his trip, he must first enable this while posting the trip, this interface exists in the notification interface and has 2 buttons “Accept” and “Decline” if the driver accepts or declines the user a notification is sent to the passenger to let them know, this notification request also contains the path and date of the trip so the driver knows which trip the request corresponds to and it also contains the date of which the passenger requested to book. If the user is accepted he’s automatically booked into the trip.

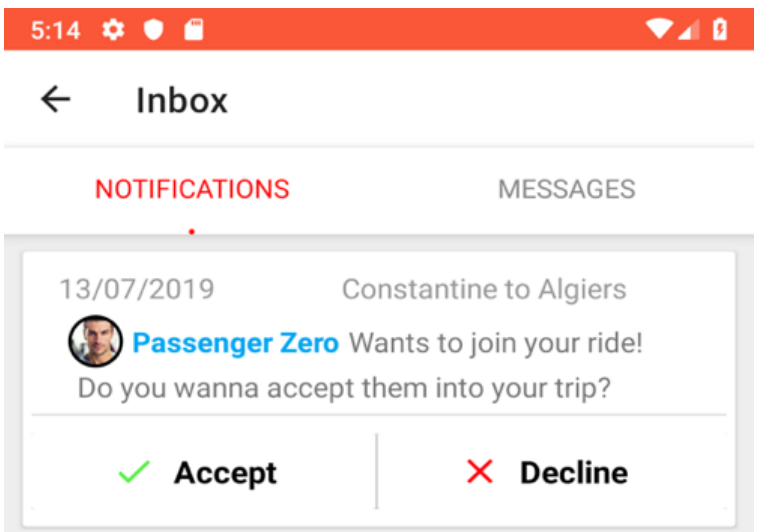

Figure 40 Booking Request View inside Notification Interface.

3.4. Passenger Interfaces

3.4.1. Main Passenger Interfaces

This is the main Passenger's interface after the user is logged, this is what the user see's once they open the application, this interface is a Fragment of MainActivity alongside with the DriverFragment, the passenger can use the driver's fragment and options they can even post trips, since a passenger is required to be 60+ of age, as long as they can drive they're allowed to. This fragment contains a RecyclerView which contains trip objects that represent the trips that this passenger is signed into, whenever a passenger books into a trip it's shown in this fragment

The main interface also contains a Search Trip FloatingActionButton, a notification and a messages button in the AppBar and a DrawerLayout which contains Profile, Settings, Logout buttons. Just like the driver's MainActivity.

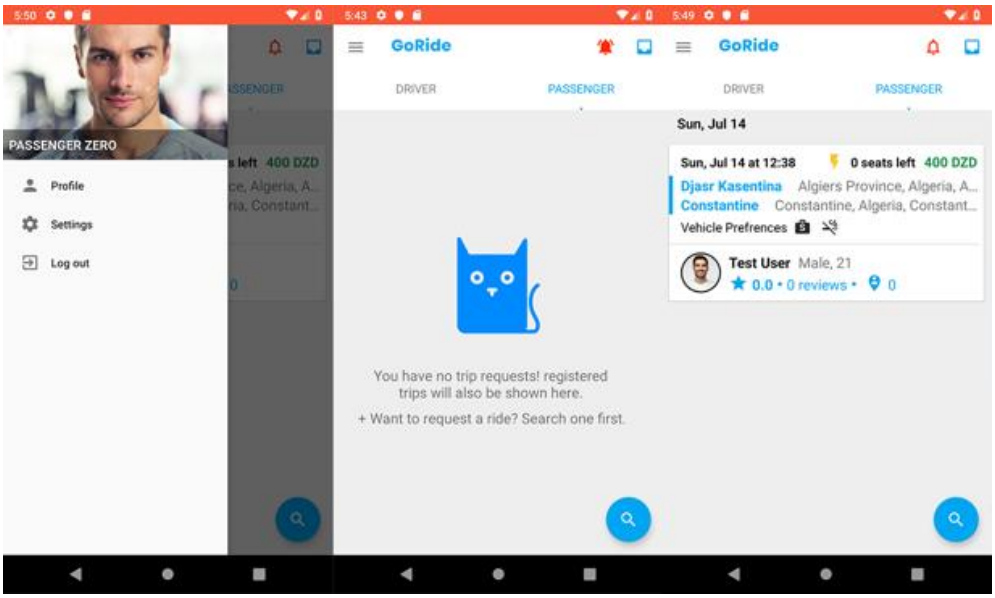

Figure 41 Main Passenger Interfaces

3.4.2. Search a Trip Interfaces

- **Main Search Interface**

In this Interface the passenger fills the search data which is consisted of the Origin and Destination and Trip Date, There's a Places AutoComplete Field for both origin and destination to optimize search, When a user inputs an Origin/Destination the

autocomplete system saves the main city and sub city, and sends it to the next activity for the algorithm to use in the search.

This interface also contains a MapView of the path between the origin and destination and also a switch button. (Search Interface is the same in Figure 35)

- **Search Algorithm and Results Interface**

The Search algorithm works as seen below in Figure 42 after the search data is returned the data is filtered by date if the user picked a specific date.

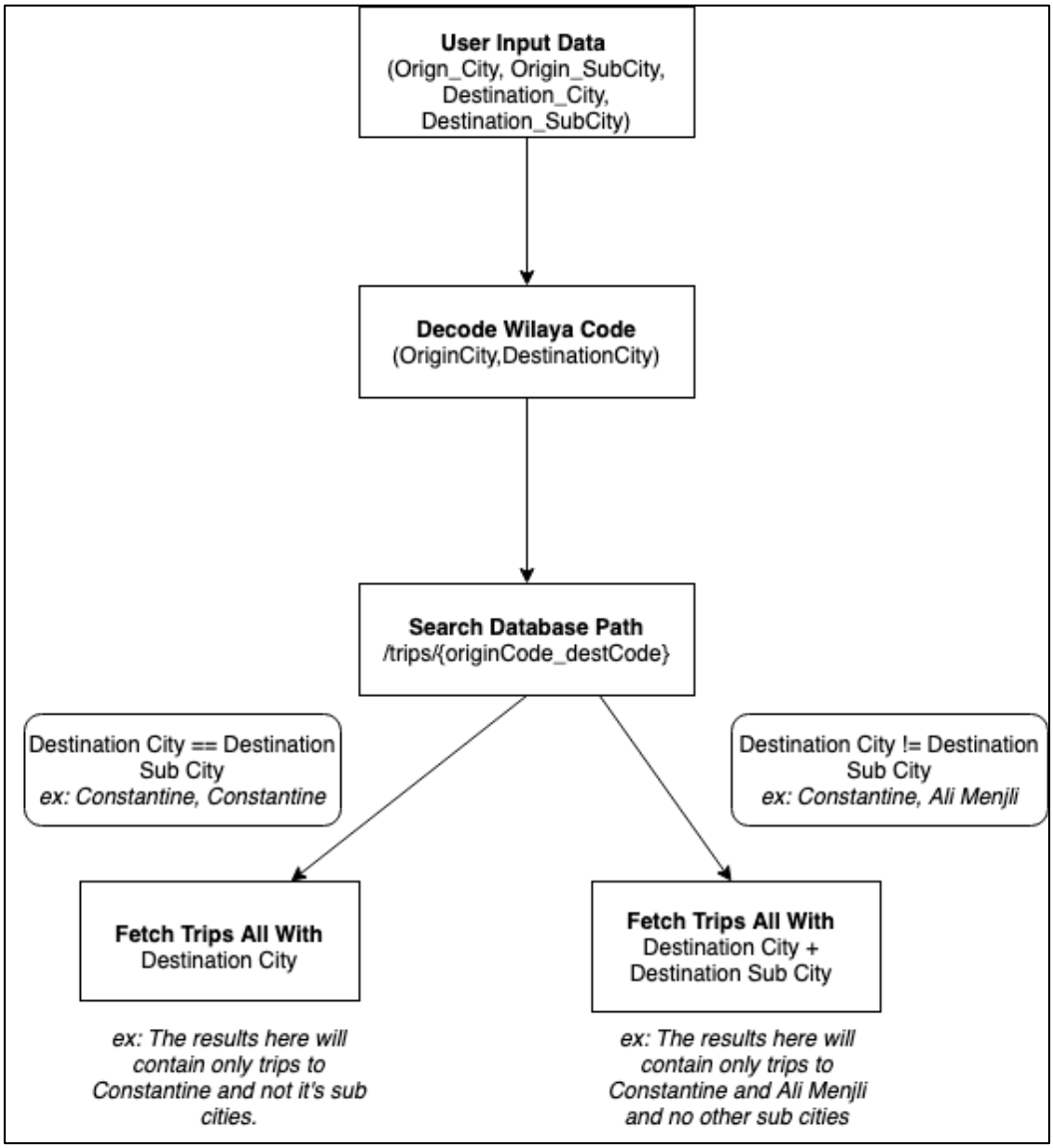

Figure 42 Search algorithm working steps

After the search data is fetched it's made into a list of Trip objects which then get sent into the results RecyclerView which gives as a clean and an optimized interface with beautiful trip views that contain all the information for users to pick their best trip, information such as Origin, Destination, Date, User Profile Information etc.

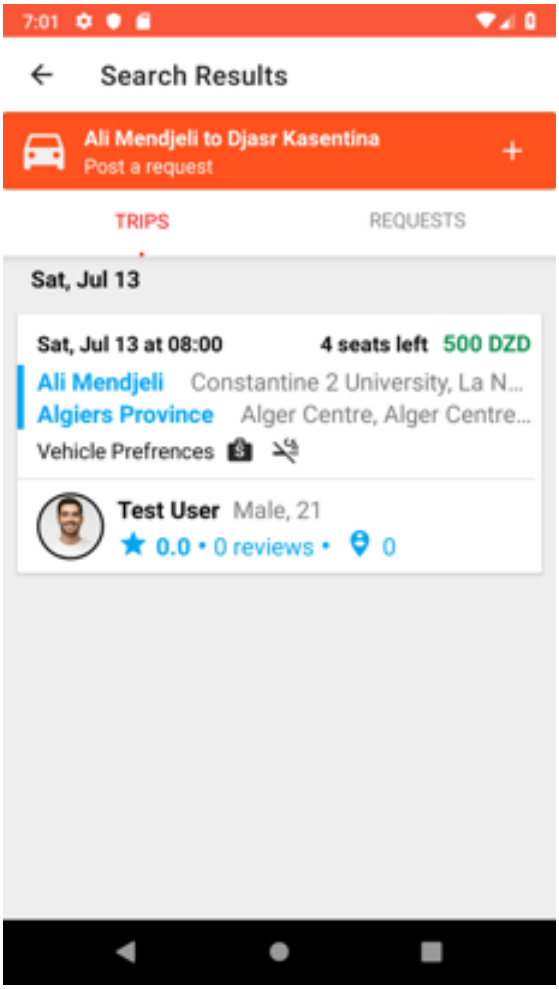

Figure 43 Results Interface

3.4.3. Trip Interface

This interface is accessed by clicking a trip from the trips list either by searching trips or finding them in the main interface, this interface contains full information posted by the driver, it's well designed and well organized , at the top it contains information about the trip, below that there's a layout that contains all booked users with their profile pictures and a link to their profiles, below that there's a section which contains driver information such as name, profile picture, ratings etc. after this section there's the optional Vehicle Preferences section which contains information about the driver's vehicle, this interface also contains a MapView of the path between the origin and destination, for passengers this interface contains a button at the very button for passenger's to register if there's empty seats, if the driver chose "Instant Booking" as a booking option the booking button will book the passenger instantly in the trip and if the driver chose "Request

Booking" the button will send a request to the driver and informs the passenger that his booking is pending, if the trip is full the button informs passengers that it's full.

A Booked Passenger has the option to cancel or report the trip at any time, if the user cancels he is un-booked from the trip, if he reports it he gets a dialog with an input field that gets sent to the administrator.

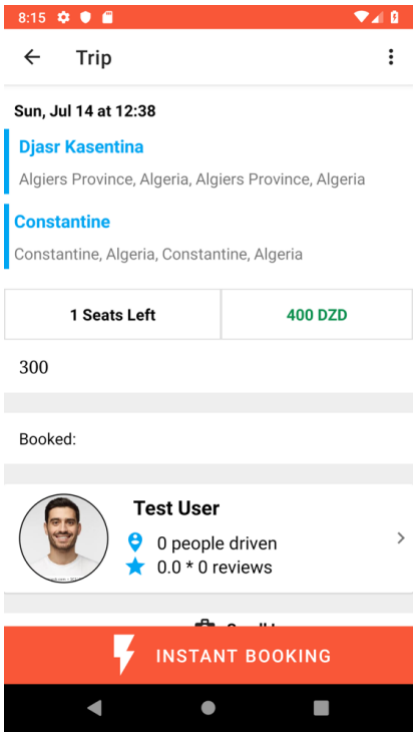

Figure 44 Trip Interface before booking

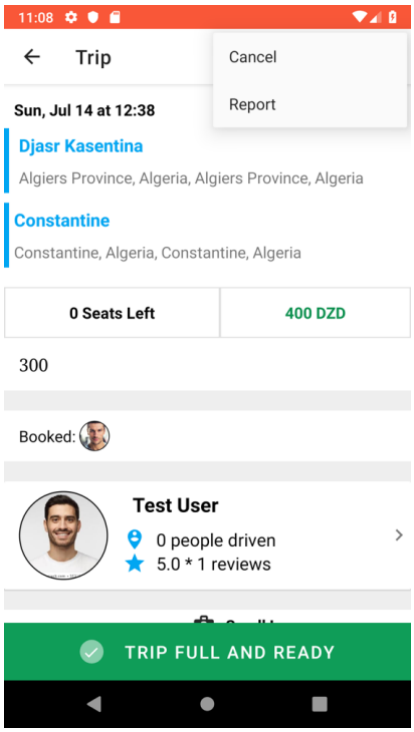

Figure 45 Trip Interface after booking

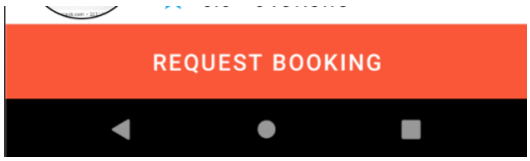

Figure 46 Request Booking Option Button

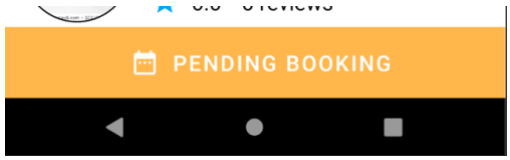

Figure 47 Pending Booking State Button

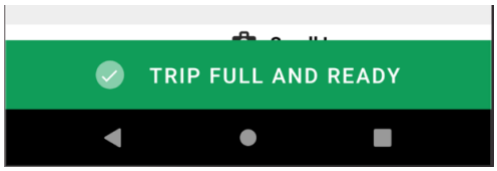

Figure 48 Registered and Full Trip State Button

3.4.4. Driver Ratings and Reviews Interfaces

This interface is triggered when a passenger's trip ends (after the trips date is passed) the passenger gets a Dialog on their Main Interface asking them if they made it our trip, if they say yes a Review Dialog is shown asking the passenger if they want to review the driver, if the passenger chooses to review the driver he's taken to the main Review Interface where he can give the driver a rating of 5 stars, and an optional review text input with 500 max characters, the passenger can also give extra rating of Timeliness (arrived on time or not), Safety (risky or safe driving) and Communications, after the passenger submits the review it's saved on the driver's database and shown on his profile. A review button will be available on the driver's profile for passengers who went on a trip with that specific driver for them to review later or modify their review.

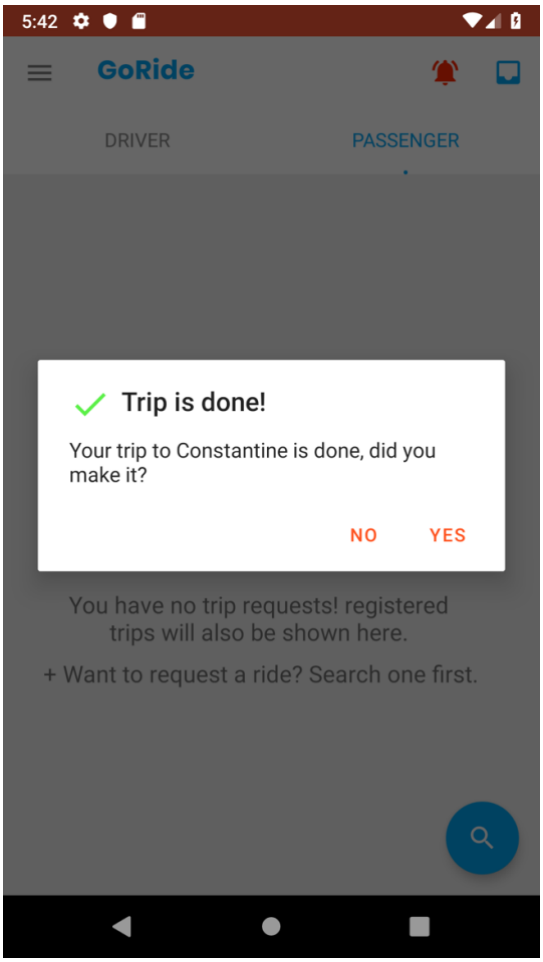

Figure 49 Trip End Confirmation Dialog

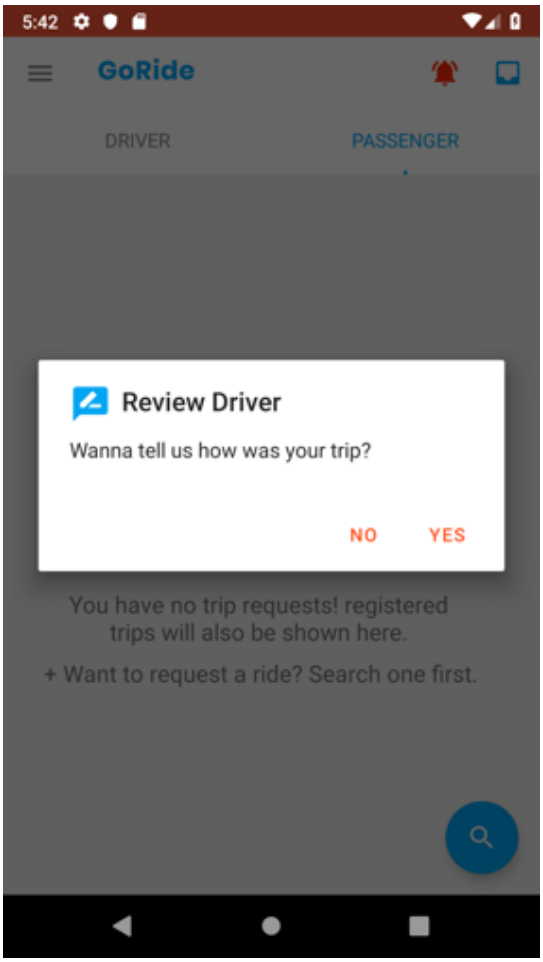

Figure 50 Review Option Dialog

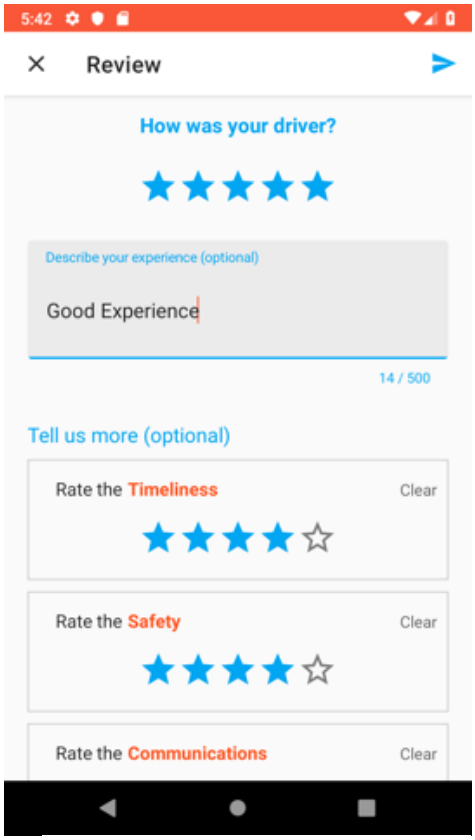

Figure 51 Driver Review Interface

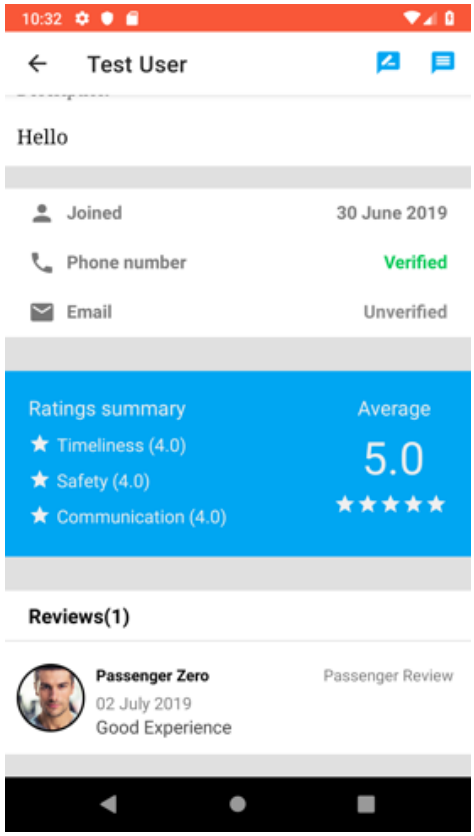

Figure 52 Ratings and Reviews on Driver's Profile

3.4.5. Trip Request Interfaces

This interface can be accessed if the passenger searches a trip or if the passenger clicks the Add FloatingActionButton on the driver's fragment it contains a form similar to the driver's post a trip form for posting trip requests by the passenger, the form contains basic trip information, once it's posted and saved into the database, it's shown on the passenger's main interface, and once a driver requests to drive a notification is sent to the passenger, if the passenger choses to accept another notification is sent to the driver asking them to post a trip with the preferences as the passenger's request and with the passenger booked in it. A passenger can always modify or cancel his trip request and this works the same way as a the driver trip modification interface.

3.5. Administrator Interfaces

For administrator features (database editing / management) we can use Firebase's web interface which lets you make changes to the database or alter users, view stats etc. Firebase also contains a Firebase Admin SDK which works in web, for our implementation we are going to turn a normal account into an admin account by altering a variable in the database from the Firebase Console,

for example we want to make a user with a specific email into an admin, we first search the User UID from the Auth Firebase Console, and then we look for it in the Database, then add a child called “superuser” and set its value to 1.

In the application once a user is logged in the system checks if the “superuser” child with the value of 1 exists, if it does the user is given admin permissions (remove trips, remove users, receive reports), we have used an Integer value instead of a Boolean because in case we want to give different levels of admin permissions in the future we wouldn’t have to change the method, for example a superuser of the value can only view reports etc. The administrator contains 3 main permissions, delete trip, delete user, receive reports.

3.5.1. Main Admin Interface

This interface is the same as the driver/passenger interface except that it contains a yellow text after the user’s name in the DrawerLayout saying “(ADMIN)” to let the user know he’s using an admin account, the interface also contains a “face” button on the top bar which contains list of user reports.

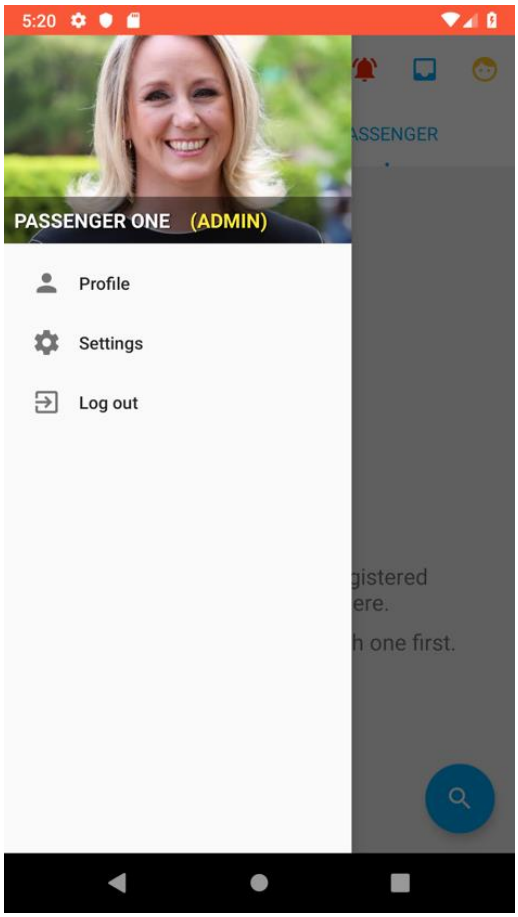

Figure 53 Main Admin Interface

3.5.2. Reports List Interface

This interface can be accessed by clicking the face button in the AppBar in the main activity it takes the admin into an activity with a list of Report Objects, which are like notifications but contain reports from users, the report objects contain the Trip Id, Reporter Id, the Report and the date of the report, the reason we included id's instead of names is for the admin to find the accounts and trips easier by looking through the Firebase Console. This interface also contains a clear button to clear the reports list.

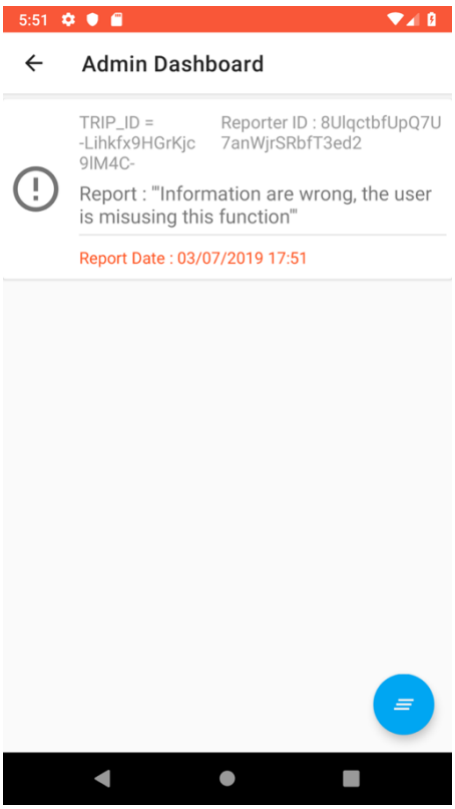

Figure 54 Reports list Interface

3.5.3. Remove Trips Interface

In this Interface the admin account user can delete trips while viewing them by clicking the menu in the AppBar then choosing the option of "Delete (superuser)", this means when the admin decides to delete this trip it's going to be removed from the database and all the booked users will be un-booked, this is so no errors happen when deleting the trip.

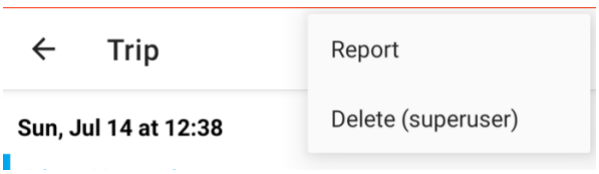

Figure 55 Delete Trip as an Admin Interface

3.5.4. Disable User Accounts Interface

This interface exists as an “Trash Bin” AppBar icon inside user profiles, only administrators can see this option, when it is clicked a Confirmation Dialog is shown asking if the admin really wants to delete this profile, if the answer is Yes, the account and all its related trips are deleted.

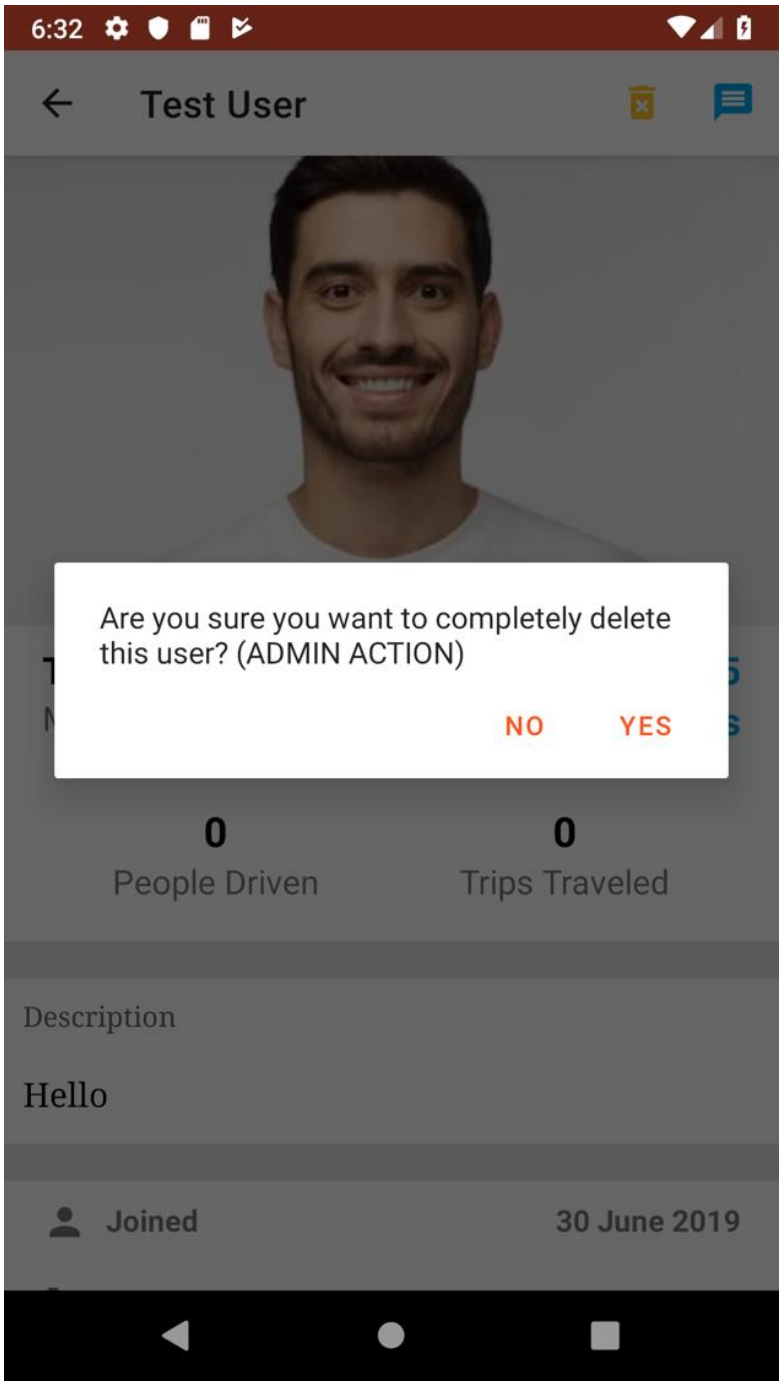

Figure 56 Delete User as an Admin Interface

3.6. Global Interfaces

3.6.1. User Profile Interface

This is the interface that contains users information it can be accessed if a user clicks their own profile or another person's profile, if it's his own profile an option to change the profile picture and modify the profile is shown to the user, if it's not their profile they have the option to message the user or review the user if they went on a trip together before. This interface contains all necessary information such as Full name, age, gender, ratings, reviews by other users, account creation date, and phone and email verifications status. The profile also shows user statistics such as Trips Traveled and People driven which gives good information to users who want to be careful about who they choose.

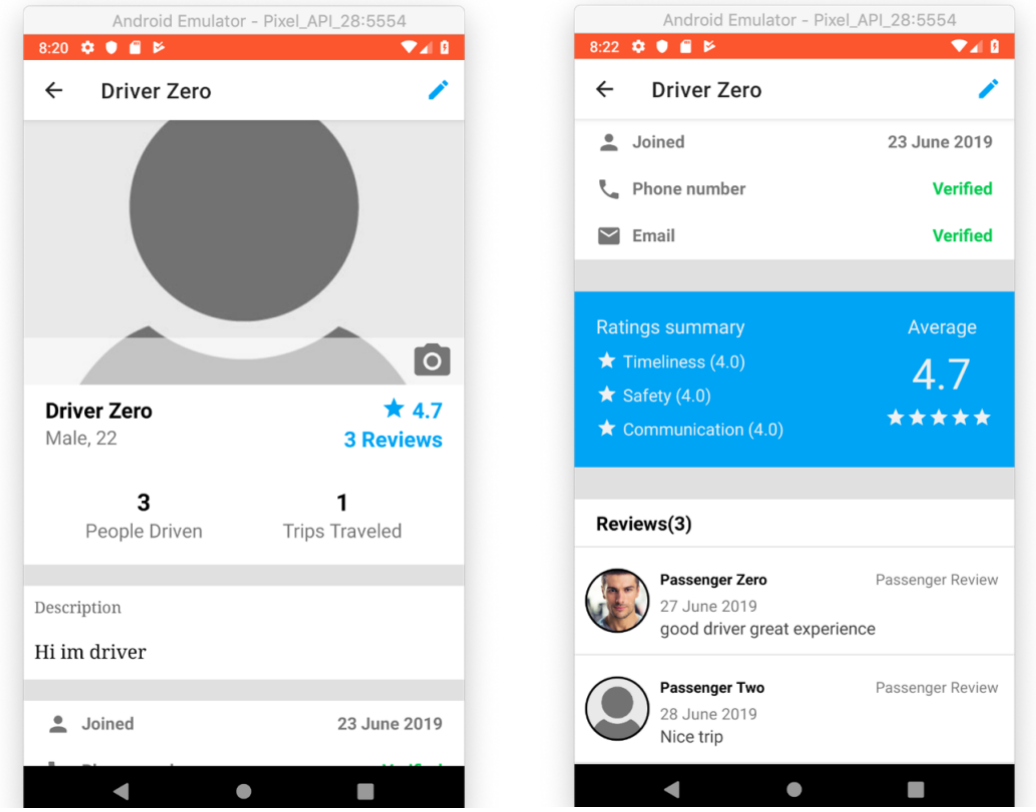

Figure 57 User Profile Interfaces (Own Profile)

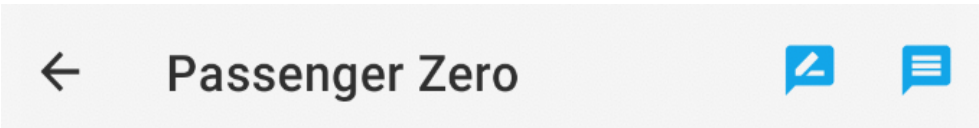

Figure 58 User Profile AppBar (Not Own Profile)

3.6.2. Notifications and Chat List Interfaces

- **Notifications**

This interface is accessed by clicking the Notification icon in the main activity's AppBar it contains all kind of notification views and driving or booking requests. The notification categories are :

For Passengers :

- Driver cancels a trip
- Driver modify a trip
- Driver rejects booking request
- Driver accepts booking request
- Driver sends a driving request for a Trip Request

For Drivers:

- Passenger books in a trip
- Passenger un-books from a trip
- Passenger accepts drive request
- Passenger rejects drive request
- Passenger sends a booking request

Example of Kotlin implementation of the notification category system:

```
when {
   notify[position].type == "unbookedUsers" -> {
      val ds1 = "Someone just unbooked! Check your trip feed to see who's left."
      holder.desc1.text = ds1
      holder.notifIcon.setImageResource(R.drawable.ic_error_outline_gray_50dp)
   }
   notify[position].type == "canceledTrips" -> {
      val desStr = notify[position].otd.substring(notify[position].otd.length - 2)
      val destination = wilayaArrayEN[desStr.toInt() - 1]
      val ds1 = "Your trip to $destination has been canceled. Sorry about that. You can
report the driver"
      holder.desc1.text = ds1
      holder.notifIcon.setImageResource(R.drawable.ic_cancel_gray_50dp)

   }
   notify[position].type == "bookedUsers" -> {
      val ds1 = "Someone just booked! Check your trip feed to see more info."
      holder.desc1.text = ds1
      holder.notifIcon.setImageResource(R.drawable.ic_person_add_gray_50dp)
   }
}
```

This interface contains a Clear FloatingActionButton to clear all notifications, and the notification icon on the main activity changes weather your notifications list is full or not.

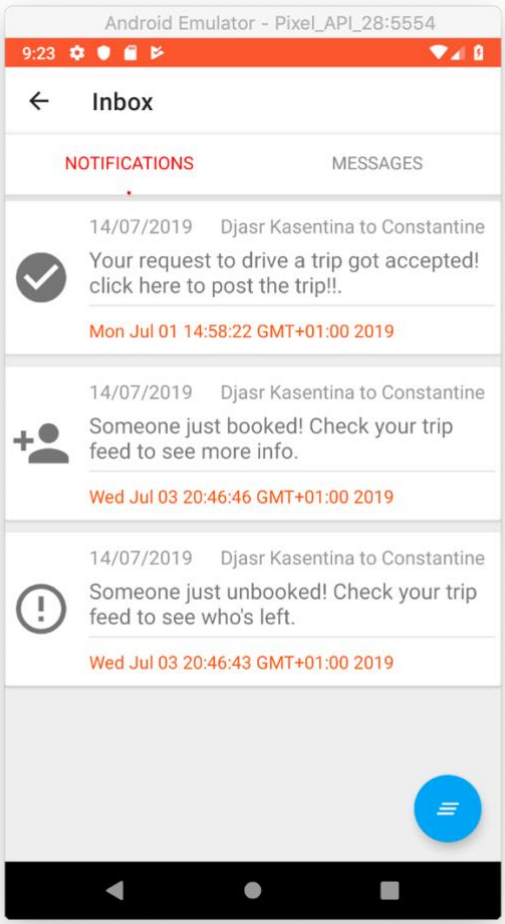

Figure 59 Filled Notification List Example Interface

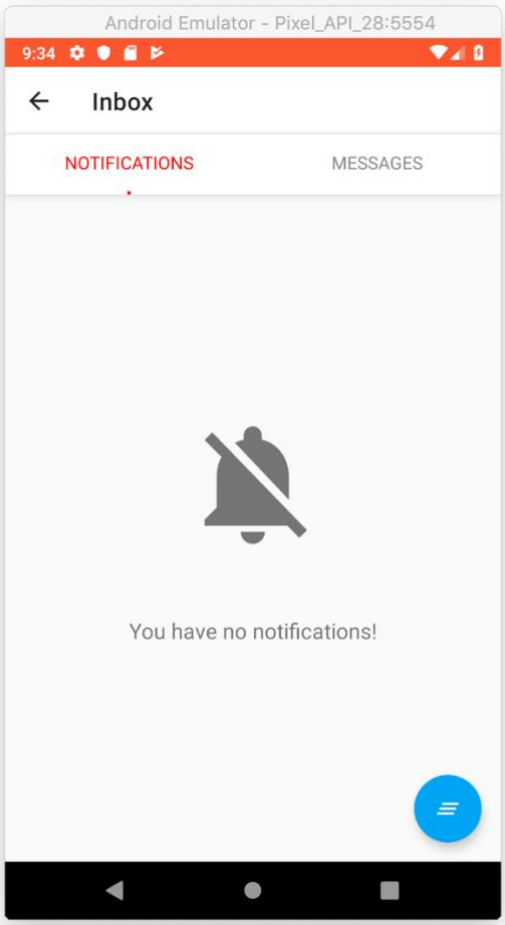

Figure 60 Empty Notification List Interface

Figure 61 Notification icon while empty

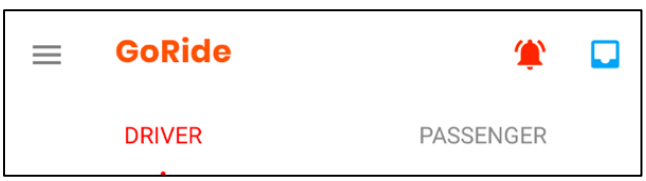

Figure 62 Notification icon while filled

- **Chats List**

This Interface is a accessed by the inbox icon in the main activity, it shows a list of conversations you had with other users, it contains meta-data such as other user's names, last message and message date.

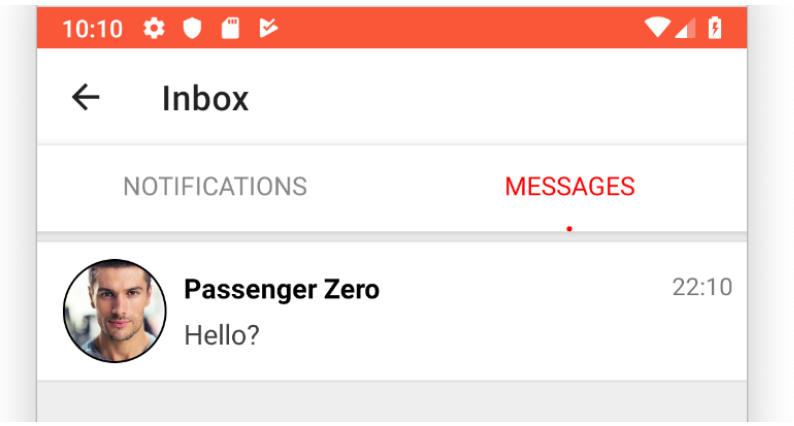

Figure 63 Chat list Interface with one conversation

3.6.3. Messages and Chat

This Interface is used by users to message each other's, to go into the Messaging Interface for the first time the user has to go to a user's profile then clicks the messaging icon, after that the conversation will be saved into the chat list interface, in the Messaging Interface users can send instant messages in real-time to each other's, they will also receive push notifications when that happens.

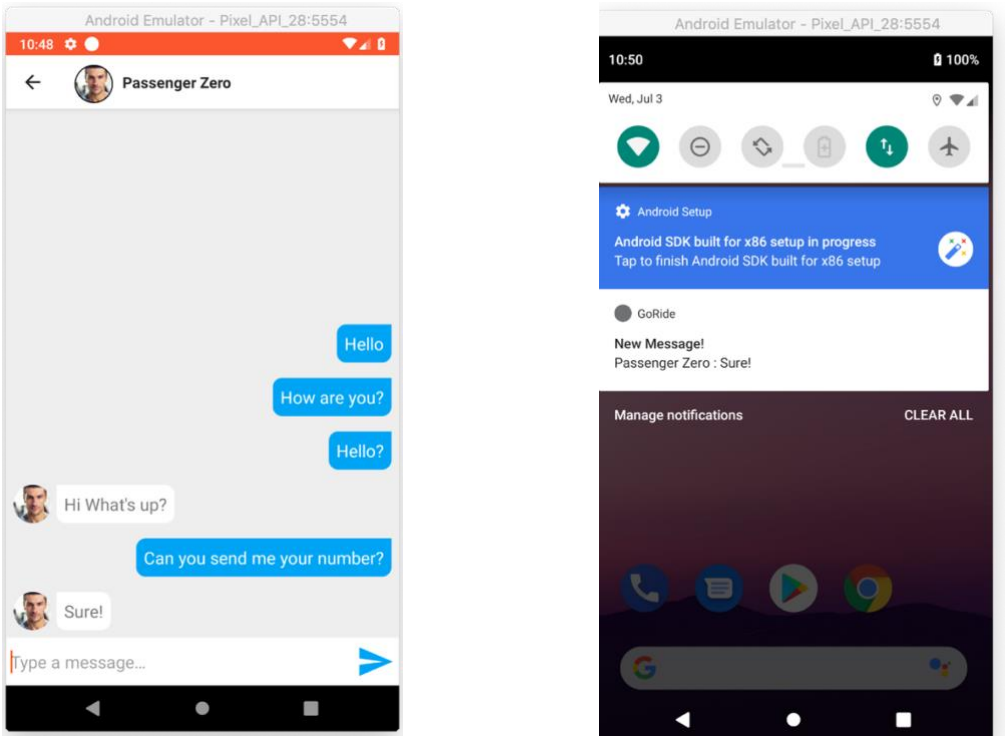

Figure 64 Instant Messages Interface and Messages Notifications

3.6.4. Settings Interface

This interface can be accessed within the DrawerLayout in the main interface and it contains various settings and important information and sections, this interface has a Modify Profile Setting, Enable/Disable push notifications setting and Terms Of Service, Privacy, About information sections and finally it has a logout button at the very bottom.

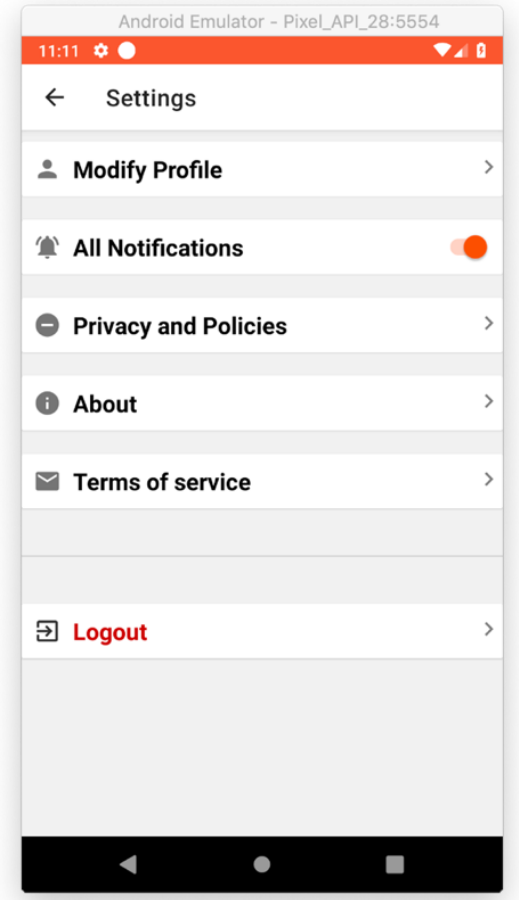

Figure 66 Settings Interface

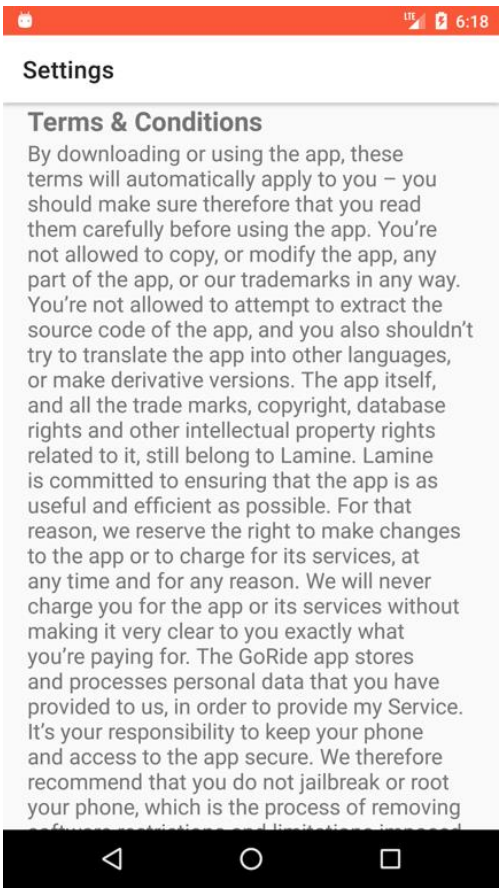

Figure 65 Terms setting Interface

3.6.5. Modify Profile Interface

This interface lets the user change some of their profile information such as Full name, Age, Description, Account type and email and password. The user can't switch to a passenger unless they're elders. The changes are immediately saved into the database after the user confirms the information in the form. This interface can be accessed either from the Setting Interface or own profile modify button on the AppBar in the User Profile Interface.

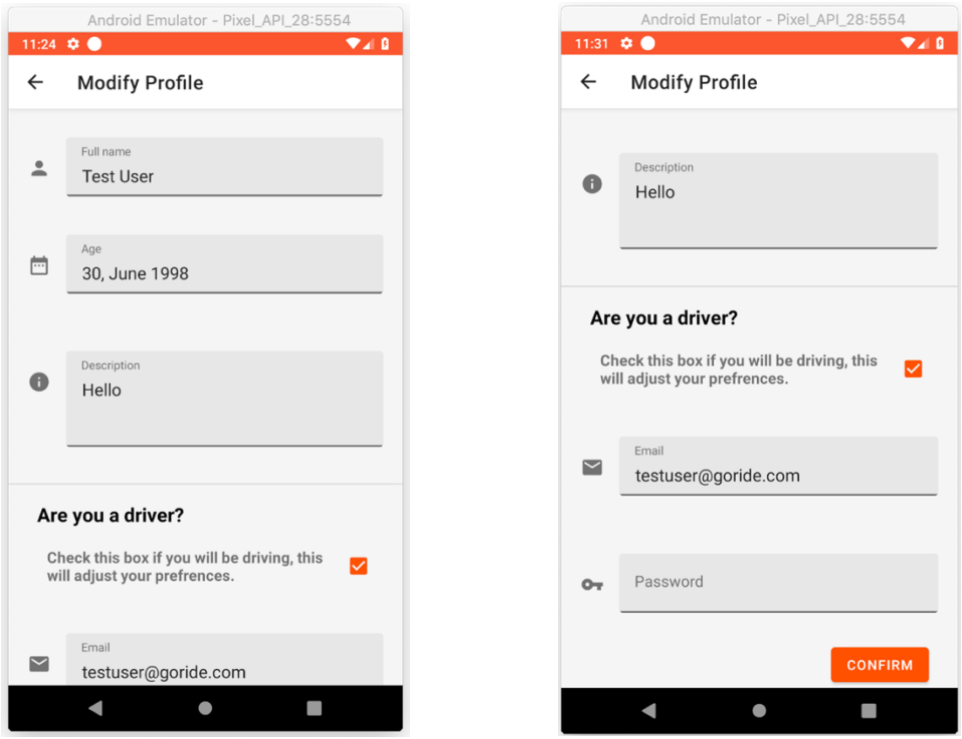

Figure 67 Modify Profile Interface

3.6.6. Splash Screen Interface

This Interface is called whenever the user opens the application, it's a white screen with the application's logo on it, in the background this interface checks if the user is logged or not, if they are, they get sent into the main interface, if not they get sent to the login activity, this interface is optional but makes the application looks more professional.

The Kotlin code for this interface works like this:

```
class SplashScreenActivity:AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        if(FirebaseAuth.getInstance().currentUser != null){
            val intent = Intent(this@SplashScreenActivity, MainActivity::class.java)
            startActivity(intent)
            this@SplashScreenActivity.finish()
        } else {
            val intent = Intent(this@SplashScreenActivity, LoginActivity::class.java)
            startActivity(intent)
            this@SplashScreenActivity.finish()
        }
    }
```

4. Conclusion

The implementation from a concept to a real application was extremely challenging but it was very effective and full of experience, the final application turned out to be very professional and organized, it contained all necessary features to make both the Driver and the Passenger feel comfortable using it from design to backend features everything is checked. After a lot of testing and bug fixes the application finally reached a final state with no known bugs.

We have implemented all functional needs and non-functional needs and some of the optional needs, some additional features were added to make the application more usable but the core concept stayed the same since it offers all the mentioned features and needs in the project description it also respects the optimization needs such as good code design style, small application size (9mb) and targeting as many Android Devices as possible.

by using Kotlin which is the new official programming language for Android we have made sure that the code was well structured as well as optimized, since the Kotlin code is inherently safer than Java code, we didn't waste so much time debugging and checking errors like we would have if we used Java. By using Kotlin with Android Studio we were able to make a Native Application which means we had access to so many Android Libraries and Native Functions which was very helpful in this project.

As for the backend my best choice was Firebase because it's very powerful and simple to use especially for beginners, it also contains a free usage tier with good performance. The application does its best to optimize the backend for both database usage and user experience. Firebase provided all necessary backend functions which made making a professional application in time possible.

By using all the tools and libraries available and by using all knowledge of Data Structures and Object-oriented programming we were able to make a well-designed application that could be used with no problems and didn't lack any important features, this implementation was definitely helpful to my career since implementing a concept into a real Application gives good knowledge of the development environment.

Final Conclusion

By the time the application was finished we can look back and tell how much we have learned from this project. So many things that should have been done differently in terms of implementation and design concept, so many additional features that an application like this has to have to be useful and competitive.

When the application was first being designed there were so many concerns that were not considered and by the time the testing happened and after all that time developing it we have realized that we learned so many things in terms of concept design and code optimizations and because of that we modified so many things since we started working on the implementation just to make the application more efficient and optimized. This proves that working on this project made us learn so many things about software development in general, not only that but we also learned how to make better conceptual design and learned how to turn an idea into a real application.

Building the application from the ground up was a great experience, the most important thing that we learned and after doing so much work is, in this type of software/product it is difficult to reach a final state so it is very important to deliver and get a first raw version and then fast and optimize and develop side features later. Some decisions were even based on this premise, having in mind that some choices are for the short-medium term rather than the long term are just faster to deliver that way, but finally we have ended up with a satisfying application that checks the main qualities of an application, these qualities being, good design, secure and efficient backend, and finally a clean and optimized coding design style.

Webography

[1]:https://magora-systems.com/mobile-application-development-architecture/

[2]:https://dzone.com/articles/everything-you-need-to-know-about-mobile-application-archi

[3]:https://blog.heliossolutions.in/mobile-application-architecture-vital-mobile-application-development/

[4]:https://www.techopedia.com/definition/2953/mobile-application-mobile-application

[5]:https://manifesto.co.uk/history-mobile-application-development/

[6]:https://legibra.com/mobile-application/

[7]:https://en.wikipedia.org/wiki/Mobile_architecture

[8]:https://en.wikipedia.org/wiki/IOS

[9]:https://en.wikipedia.org/wiki/Windows_Phone

[10]:https://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

[11]:https://en.wikipedia.org/wiki/Carpool

[12]:https://www.cleveroad.com/blog/see-how-to-start-a-rideshare-business-and-make-a-rideshare-application-from-scratch