UNIVERSITY CENTER - SALHI AHMED - DE NAAMA

INSTITUTE OF SCIENCES & TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

END OF STUDY THESIS

To obtain the diploma:

MASTER IN COMPUTER SCIENCE

Specialty: Information Systems

Theme

Design and production of an Android application for carpooling

Presented by :

- Tadj Bahaa Eddine

Supervised by :

- Mr : Khiati Nadri

University year : 2021-2022

Summary:

Carpooling is a mode of transport where an individual shares a journey with a group of people, in a not-profit manner. The benefits of that can range from personal like reducing costs and avoiding traffic, to environmental like reducing fossil fuel usage and the number of roads that need to be built. Therefore, it is essential for it to be adopted by as many people as possible.

To effectively carpool, thus encourage its usage, there needs to an online platform that can manage all the planning required in an efficient and easy-to-understand manner.

The purpose of this work is to create a native Android application for peer-to-peer online carpooling that fulfils that role.

Résumé :

Le covoiturage est un mode de transport dans lequel un individu partage un trajet avec un groupe de personnes, dans un but non lucratif. Ses avantages peuvent être d'ordre personnel, comme la réduction des coûts et l'évitement du trafic, ou d'ordre environnemental, comme la réduction de l'utilisation des combustibles fossiles et du nombre de routes à construire. Il est donc essentiel qu'il soit adopté par le plus grand nombre de personnes possible.

Pour simplifier le covoiturage, et par conséquent l'encourager, il faut une plateforme en ligne capable de gérer toute la planification requise de manière efficace et facile à comprendre.

L'objectif de ce travail est de créer une application Android native pour le covoiturage en ligne pair-à-pair qui remplit le rôle de cette platform.

ملخص:

مشاركة السيارات هي وسيلة نقل يشارك فيها الفرد رحلة مع مجموعة من الناس، لغرض غير للربح. يمكن أن تتراوح فوائد ذلك من الشخصية مثل تقليل التكاليف وتجنب حركة المرور، إلى البيئية مثل تقليل استخدام الوقود الأحفوري وعدد الطرق التي يجب بناؤها. لذلك، من الضروري أن يتم تبنيها من قبل أكبر عدد ممكن من الناس.

من أجل استعمال مشاركة السيارات بشكل فعال، وبالتالي تشجيع استخدامها، هناك حاجة إلى منصة عبر الإنترنت التي يمكنها إدارة كل التخطيط المطلوب بطريقة فعالة وسهلة الفهم.

الهدف من هذا العمل هو انشاء تطبيق فطري " Native " لمشاركة تطبيق مشاركة سيارات من "راسل الى مستقبل" عبر الانترنت لتأدية دور المنصة المذكورة سابقا.

Acknowledgments:

First of all, I would like to thank Mr. Khiati Nadri, for his support and guidance throughout the preparation of this dissertation. I thank him very much for his availability and his precious advice which allowed me to carry out my work.

The members of the jury, who did me the honour of attending the review of this modest work.

I would also like to thank the entire teaching team for this very rich scholar year.

Naama, June 20, 2021

Table of Contents

Table of figures:

General Introduction:

Carpooling is a great solution for solving a lot of issues that are growing in vehicle-based transportation like increasing costs, fuel emissions, traffic congestions and more. However, the effort and time spent planning for it can be very tiring.

So, how do we solve this issue?

Easy, we provide an online platform that facilitates the communication and planning necessary to carpool. And what better platform to use than an Android application, since Android has sited 2.8 billion users in 2020.[reference 1]

For this application to be a success, it needs to be functional, reliable and easy to use. To achieve this, this thesis will cover the following topics:

Chapter 1: Some definitions to carpooling.

Chapter 2: An introduction to Android and Android applications.

Chapter 3: The planning for the application.

Chapter 4: The technologies the methods used in the development of the app.

Chapter 1: ***Carpooling***

Introduction:

In this chapter, we will define carpooling and its various types as well as its benefits. This is necessary to understand the audience and their demands of this application.

1. Definition:

Carpooling, also known as ride-sharing, is when multiple people travel in the same vehicle, sharing the costs, for reasons other than profit. The travel details for carpooling are usually agreed upon in advance, those details may include: depart location and time, destination, individual cost, one or multiple stops, mid-way stops…

Carpool commuting is more popular for people who work in places with more jobs nearby, and who live in places with higher residential densities.

Carpools can be categorised using two criteria: their procedure, and their purpose.

2. Types of procedures:

2.1. Ad hoc carpooling:

Ad hoc carpooling is an informal procedure where to join a ride without a prior plan. As the name suggests, this type of carpool is very situational and is usually the result of unpredicted circumstances.

2.2. Periodic Carpooling:

A formal form of ride-sharing where the need to travel to a specific location is a reoccurring event, so the travel details are agreed upon once and are always the same.

2.3. Peer-to-peer carpooling:

A formal form of ride-sharing for on-off travels that are planned beforehand, usually conducted on an online platform (site or mobile app). It's called peer-to-peer because the travels are planned and managed by the travellers, with the platform acting as a facilitator.

2.4. Vanpools:

Similar to periodic carpooling, but with a much bigger vehicle like a van.

3. Types of purpose:

3.1. Home-to-work carpools:

This type of carpool is usually used by people who live in the same area (or on the path of the travel), leave for the occupation at the same time, and whose occupation is in the same general area.

3.2. Fam-pools:

The most common type of carpool, it's when a family travels together, whether to the same location or different locations.

3.3. Event carpools:

Event carpools are when a group that has a similar interest, plans to travel to a place that is related to that interest e.g., football fans travelling to watch their team.

3.4. Online carpools:

This type contains any carpooling that is planned online. The purpose of this type is purely to reduce travel costs.

4. Benefits:

Carpooling can have many benefits depending on its type, which may include:

- Reducing travel costs.
- Limiting fuel usage, thus helping the environment and the economy.
- Finding a more reliable travel means.
- Reducing traffic.
- Gaining access to High-occupancy vehicle lane.
- Avoiding travel stress.
- Meeting new people.

Conclusion:

Understanding everything so far, we come to the conclusion that the app developed in this thesis must be a peer-to-peer online carpooling app. That is because it's that type that can benefit from the mobility and flexibility of Android apps.

Chapter 2: Android Applications

Introduction:

In this chapter, we'll explore the topic of Android and its architecture. We'll also define Android applications, see their types and finally their architecture. All of that to see which type and configuration suits our needs.

1. The Android Operating System:

1.1. Definition:

Android is an Operating System for touch-based mobile devices owned by Google.

The OS was first developed by Android Inc (Andy Rubin, Rich Miner, Nick Sears, Chris White) and later by the Open Handset Alliance, a consortium of tech companies like Samsung and Motorola, as a new way of interacting with mobile devices.[reference 2]

1.2. Architecture:

Android is based on the Linux Kernel and is made up of proprietary software (e.g., play store) and open-source components. Android can be divided into the following layers:[reference 3]

- **Application framework**: The application framework is used most often by application developers. This layer is denoted by an API level that details its capabilities.
- **Binder IPC**: The Binder Inter-Process Communication allows the application framework to cross process boundaries and call into the Android system services code. This enables high level framework APIs to interact with Android system services without concerning themselves with how they work.
- **System services**: System services are self-contained modules that manage a portion of system functionality. Android includes two groups of services: system (such as Window Manager and Notification Manager) and media (services involved in playing and recording media).
- **Hardware abstraction layer (HAL)**: A HAL defines a standard interface for hardware vendors to implement, which enables Android to be work without caring about the lower-level driver implementations. HAL implementations are packaged into modules and loaded by the Android system at the appropriate time.
- **Linux kernel**: This layer contains the lower-level hardware drivers. Android uses a version of the Linux kernel with a few special additions such as Low Memory Killer (a memory management system that is more aggressive in preserving memory), wake locks (a Power Manager system service), the Binder IPC driver, and other features important for a mobile embedded platform. These additions are primarily for system functionality and do not affect driver development.

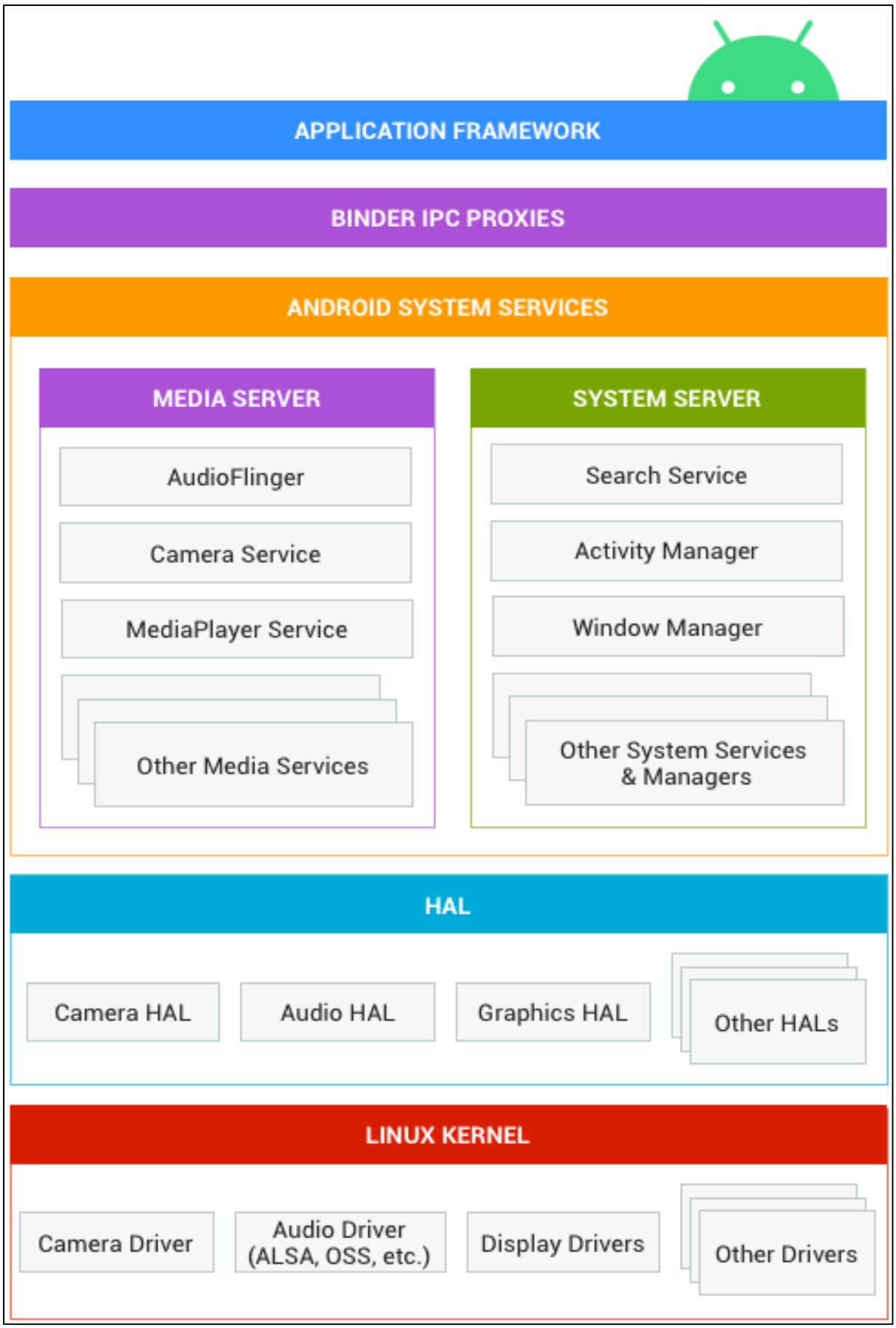

Figure 1: Android architecture

All Android processes have their own ID, properties and Virtual machine on which they are executed. Because all processes are executed independently in their own Virtual Machine , they cannot communicate directly with each other, so communication has to be done through the OS using the Binder IPC proxies. All applications have their own permissions and privileges which are either granted or revoked by the system (some permissions require the user's consent).

2. Android applications:

2.1. Definition:

Android apps (written in Java, Kotlin or C++) are built as a combination of components that can be invoked individually. Those components can be divided into 4 types (Activities, Services, Broadcast Receivers and Content Providers), and while an application will have only one main launcher, the rest of the components can be accessed directly by using intents and intent-filters. [reference 4]

2.2. Types:

There are 3 types of mobile applications:

- **Native:** These are app that are developed for a single platform. Native apps produce the best performance between the three and can use platform specific APIs.
- **Web-based:** Like the name suggests, Web-based applications are made using web technologies like HTML, CSS… and are cross-platform.
- **Hybrid:** Hybrid apps are a mix of the first two types and are developed using frameworks like Apache Cordova, Flutter, Xamarin, React Native and Sencha Touch. these apps use a single Codebase that works in multiple platforms.

2.3. Components:

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter the app. Some components depend on others.

There are four different types of app components:

- **Activities:** The main building blocks of the application, an activity represents a single screen with a user interface defined in resources and a class that defines how to respond to input events. Activities can also host fragments which are modular components that define their own UI and can handle their own input events.
- **Services:** services are processes that execute in the background. They don't have a UI and can be managed by either the Job Scheduler or Work Manager.
- **Content Providers:** Content Providers present a way for apps to share their data with other application by assigning said data with public URIs, so when another application requests the data assigned to a URI the system can fetch it if permission is granted.

- **Broadcast Receivers:** A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. So, with a Broadcast Receiver an app can schedule an event, stop running and receive an alarm when the event has happened.

An application will also contain a number of resources that help define its look and behaviour.

2.4. Architecture:

Google recommends using a small number of activities (one if possible) that represent top level destinations, which host several fragments that executes a specific functionality. The goal behind that is to have each activity accomplish its job independently of other components.

Navigation between fragments can be done using the relative fragment manager to do transactions, like shown in the figure below:[reference 5]

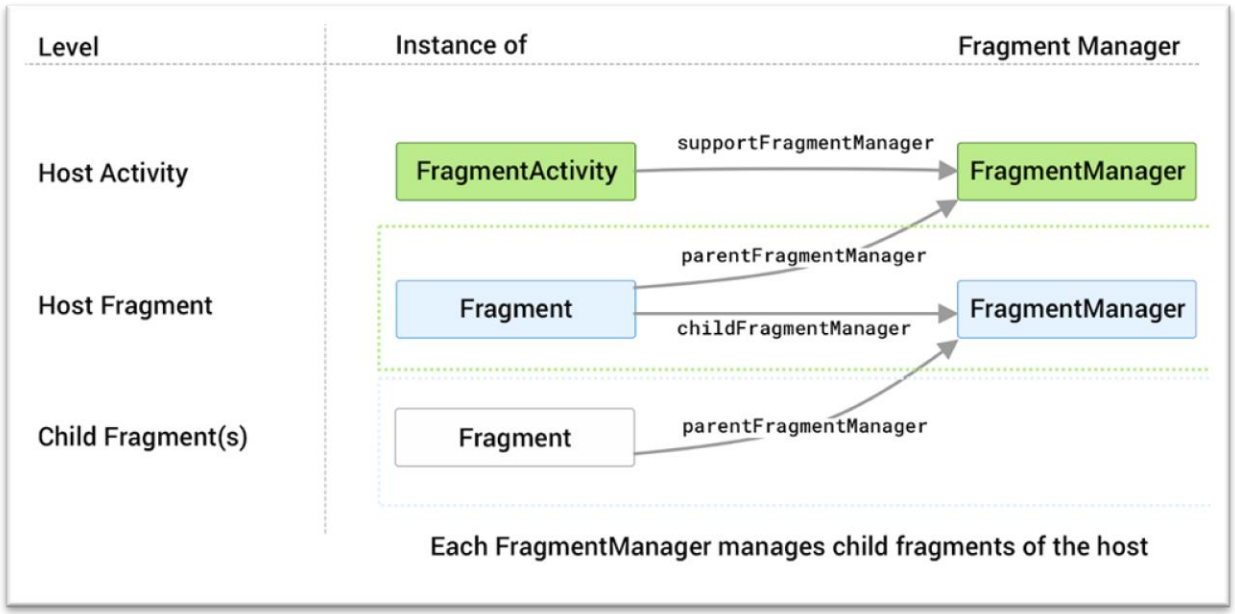

Figure 2: Fragment managers hierarchy

All activities have a lifecycle which is detailed in the figure below:[reference 6]

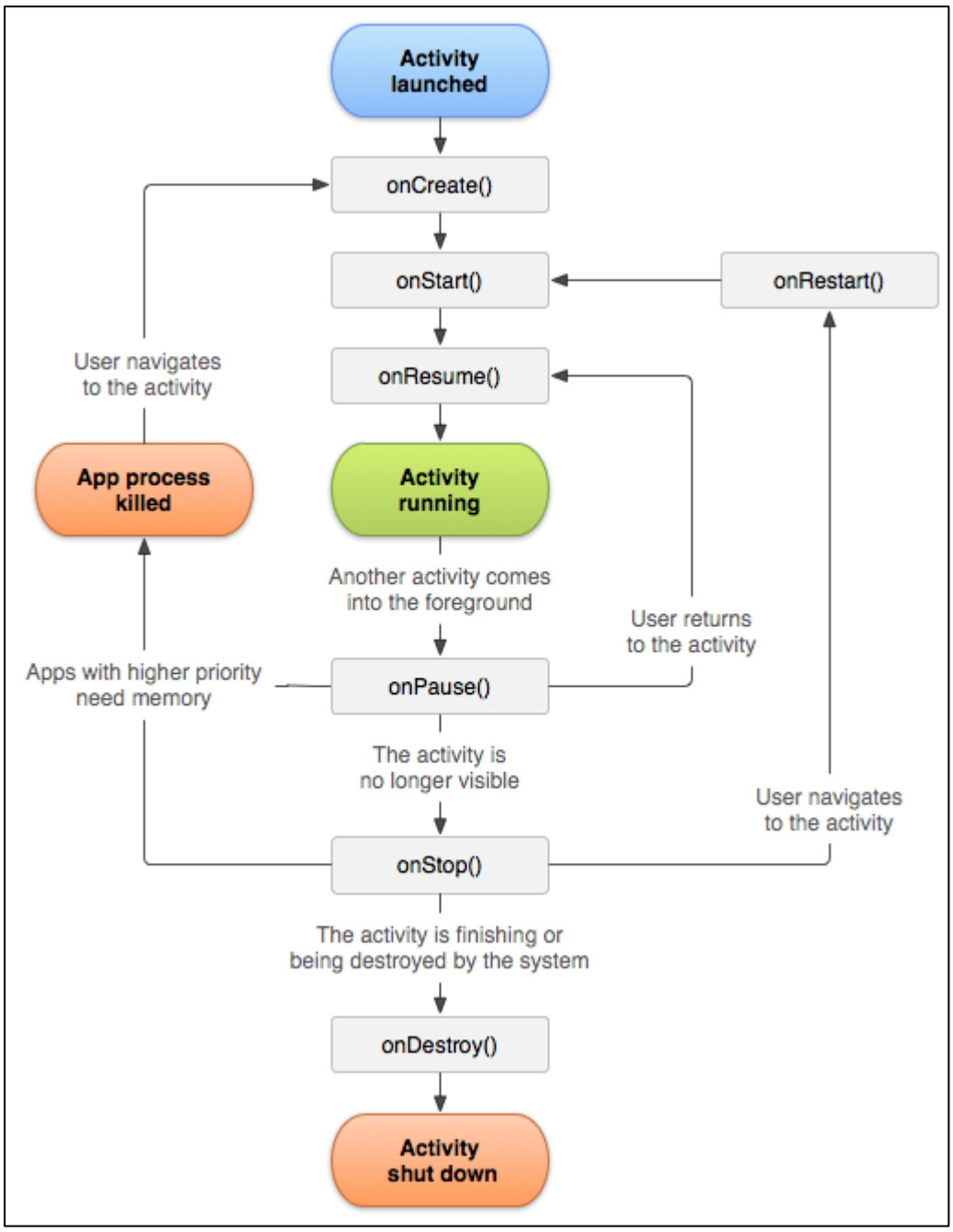

Figure 3: Activity life cycle

When an activity or fragment is destroyed, all data that it contained will be lost. To prevent this the app can either save it in another object called a ViewModel during execution or save it to a persistent container like a file, SQLite database or an external database. ViewModels use a class called a Mutable Live Data, which a type of observable data container that can hold any type of object and is life-cycle aware, meaning it only updates the views when an activity or fragment is active.

Conclusion:

In conclusion, an Android application must have a type, purpose and respect Google's design guidelines. In our case, we chose a native Java app with two activities: one for authentication and the other for the carpooling functionalities.

Chapter 3: *Conception*

Introduction:

As shown in the previous two chapters, the purpose of this work is to create a ***native Android application*** for ***peer-to-peer online carpooling***. Since the main objective of this app is to keep track of travel plans, all capabilities of this app require internet connectivity.

1. Success criteria:

The carpooling application will need to fulfil several requirements to be effective, which include:

- Uniquely identifying each user.
- Save their personal information (email, profile picture, name…).
- Letting verified users create travel records which contain info like depart, destination, time, driver…
- Searching for said records and requesting to join.
- Accepting or denying said requests.
- Keeping track of relevant travel info like requests, passengers, joined travels…
- A way of communication between travellers.
- Viewing and modifying personal information.

To do this, several components need to planed and implemented. This chapter will cover the planning phase.

2. Application use case diagram:

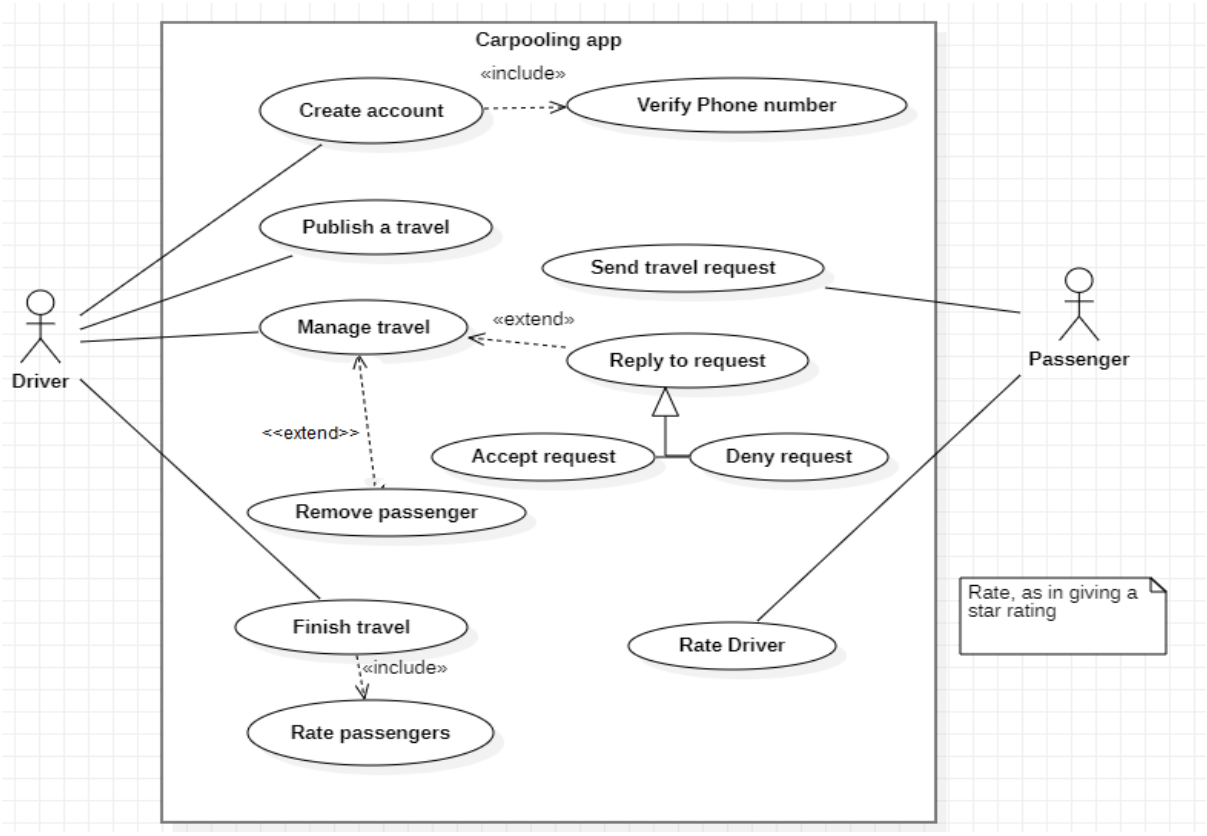

Figure 4: Application use case diagram

3. Application breakdown:

The app will contain two activities “Sign in Activity” and “Main Activity”, the first will handle the authentication and account creation process and is the launcher and the later will handle the apps five core functionalities (“Book”, “Publish”, “My Rides”, “Messenger”, “Account”).

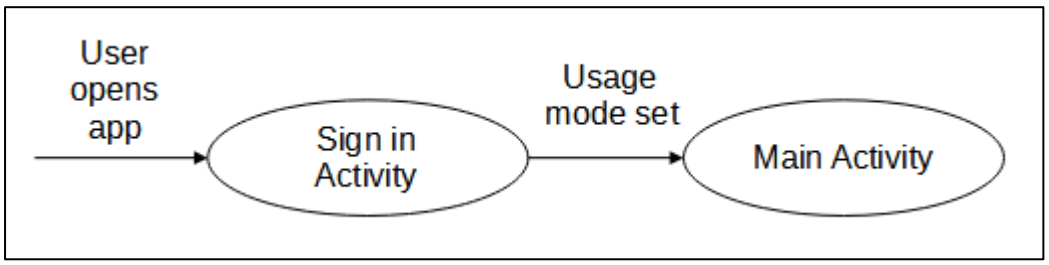

Figure 5: activities launch sequence

3.1. Sign in Activity:

Sign in activity has three main functionalities:

- Fetch user data (if it exists) from database.
- Authenticate new users.
- Create an account if the user wants to.
- Set usage mode (anonymous or known user).

All of which will be done in the following sequence:

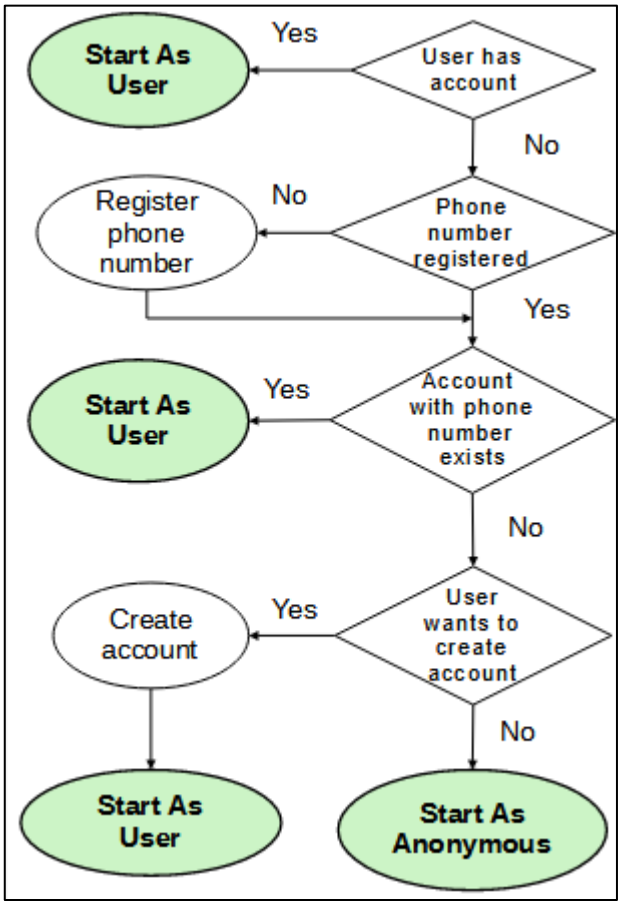

Figure 6: Sign in activity flowchart

An account is indexed by the user’s ID which is generated when they authenticate their phone number, and will contain their personal info like their name, email, phone number, star rating…

After the sequence, the app will open the Main Activity with the usage mode set to either “Anonymous” if the user is authenticated using only the phone number, or “User” if the user has an account in the database. The usage mode will set what the user can and cannot do.

Sign in Activity Diagram:

In this diagram, a red end point means signing in as “Anonymous”, and a blue end point means signing in as “User”:

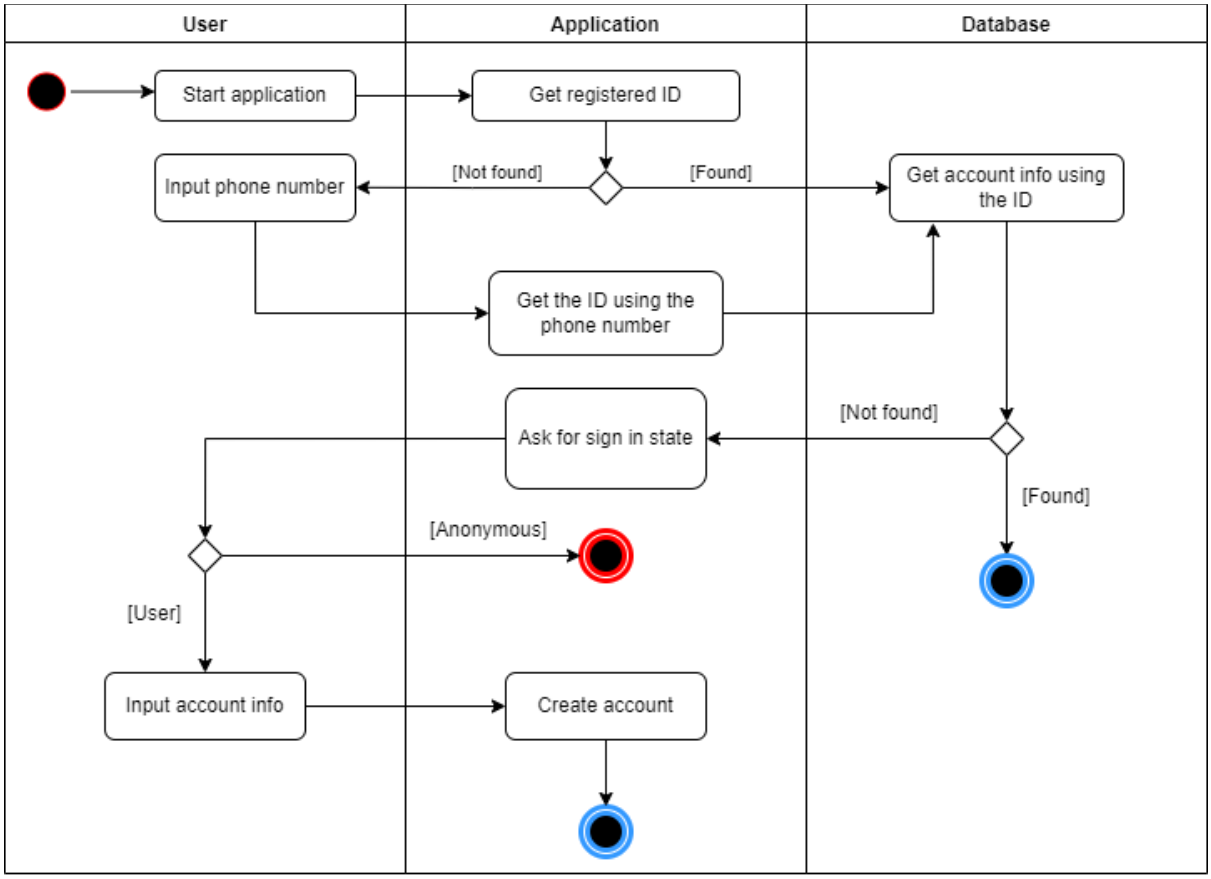

Figure 7: Sign in Activity diagram

3.2. Usage modes:

The functionalities that the user can access in Main Activity can change depending on the state passed in the previous activity.

3.2.1. User Usage mode:

In this mode the user has access to all five functionalities which are:

- Publishing a travel record (Publish).
- Searching for, and requesting to join a voyage (Book).
- Keeping track of, and modifying relevant travel info (My rides).
- Sending and receiving messages (Messenger).
- Viewing and modifying account info (Account).

3.2.2. Anonymous usage mode:

In this mode the user has no account, only an ID and a phone number. An anonymous user has access to the full “Book” functionality, the “Joined rides” and “Ride requests” from “My rides” and the ability to either create an account or use another phone number.

3.3. Main Activity:

3.3.1. Publish functionality:

To publish a travel, several details are needed to inform potential passengers about which are:

- The name of the depart location.
- The name of the destination location.
- The date and time of taking off.
- The number of seats available.
- The driver's contacts.

When all details are provided a new travel record is added to the database with a unique ID to search for it.

Publish Activity diagram:

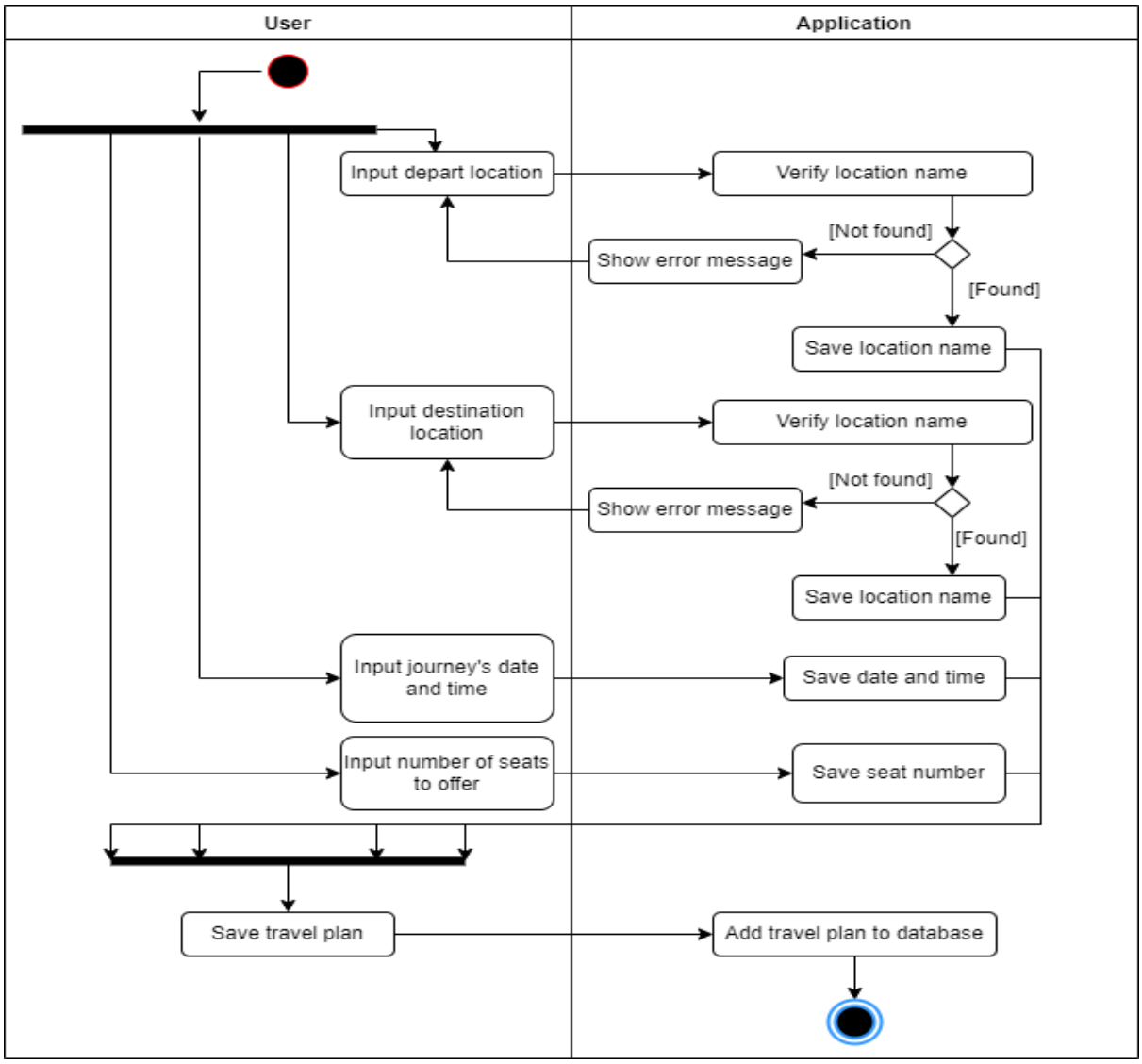

Figure 8: publish activity diagram

3.3.2. Book functionality:

The Book component has two usages: finding matching travels and sending requests to them. It requires the user's travel demands (depart, destination, date, seat to request) to do the first job.

When the details are entered, the app will show a list of matching rides with the name and star rating of the driver. The star rating will show the impression of their previous passengers, so the higher the better.

When the user selects a travel record, they will be prompted to send a travel request which the driver can either accept or deny.

Book Activity diagram:

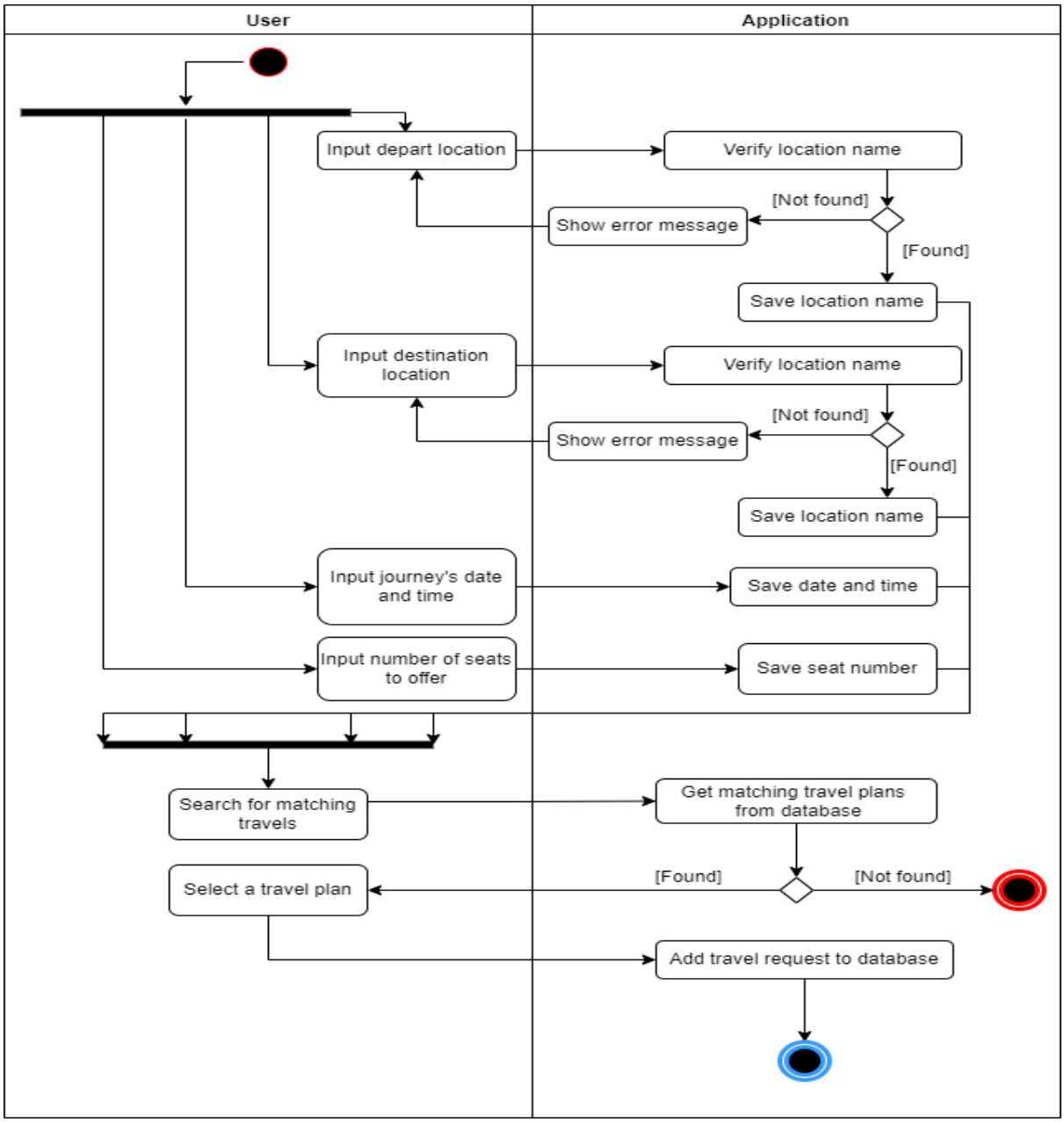

Figure 9: book activity diagram

3.3.3. My rides functionality:

The main job of this component is to keep track of the user's travel info, which belong to three categories:

- **Started rides:** includes all the travel records that the user created in the publish functionality.
- **Joined rides:** includes all travel records that the user is a passenger in.
- **Ride requests:** includes all the travel requests that the user sent in the Book Functionality.

3.3.3.1. Started rides:

Selecting a started ride will show two lists:

One for all the requests for the selected travel, and selecting one will prompt the user to either accept or reject said request.

The other will show all the passengers of the travel, and selecting one will prompt the user to remove said participant (cannot be done 2 hours before a travel).

When the travel is finished, the driver is prompted to give a star rating all passengers before removing the travel record from the database.

3.3.3.2. Joined rides:

A joined ride will appear after a request has been accepted by the driver, and will contain all the travel details. Selecting a joined ride will show the driver's contacts; like their phone number and email.

When the travel is finished, the user will be prompted to give a star rating to the driver before the joined ride disappears.

3.3.3.3. Ride requests:

A ride request will contain the requested travel's details along with a state. The state when the request is first sent is "Waiting" since the user is waiting for the driver to respond.

If a request is accepted by the driver, then it will disappear and a joined ride will appear in its place, and if it is denied then the its state is changed to "Denied".

If a request is selected, then the user will be prompted if they want to cancel their request.

My rides Activity diagram:

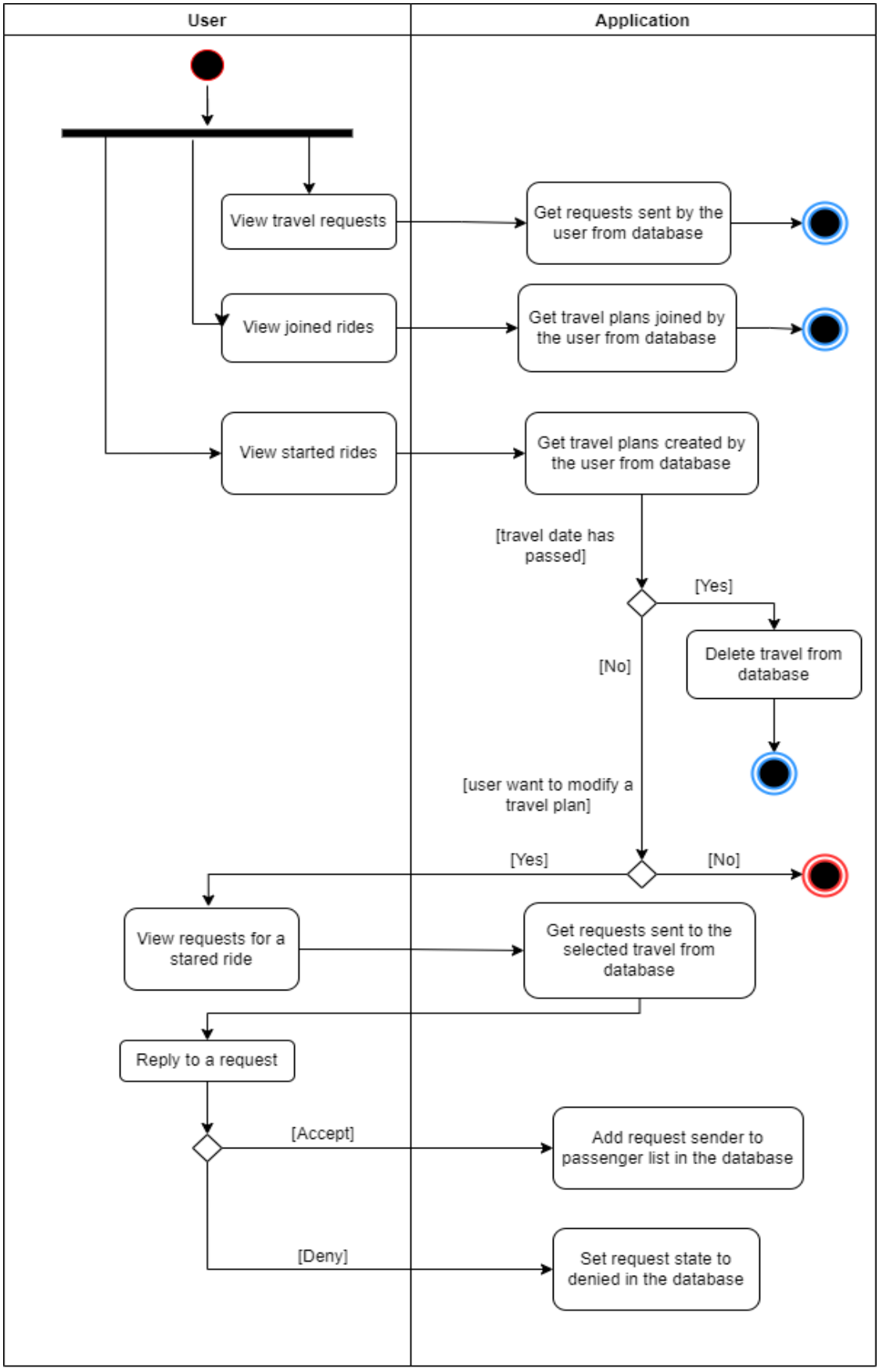

Figure 10: my rides activity diagram

3.3.4. Messenger functionality:

The messenger component will show a list of all the passengers/drivers that the user is traveling with. When the user selects one, they will start a “Chat room” where they can exchange messages.

Messenger Activity diagram:

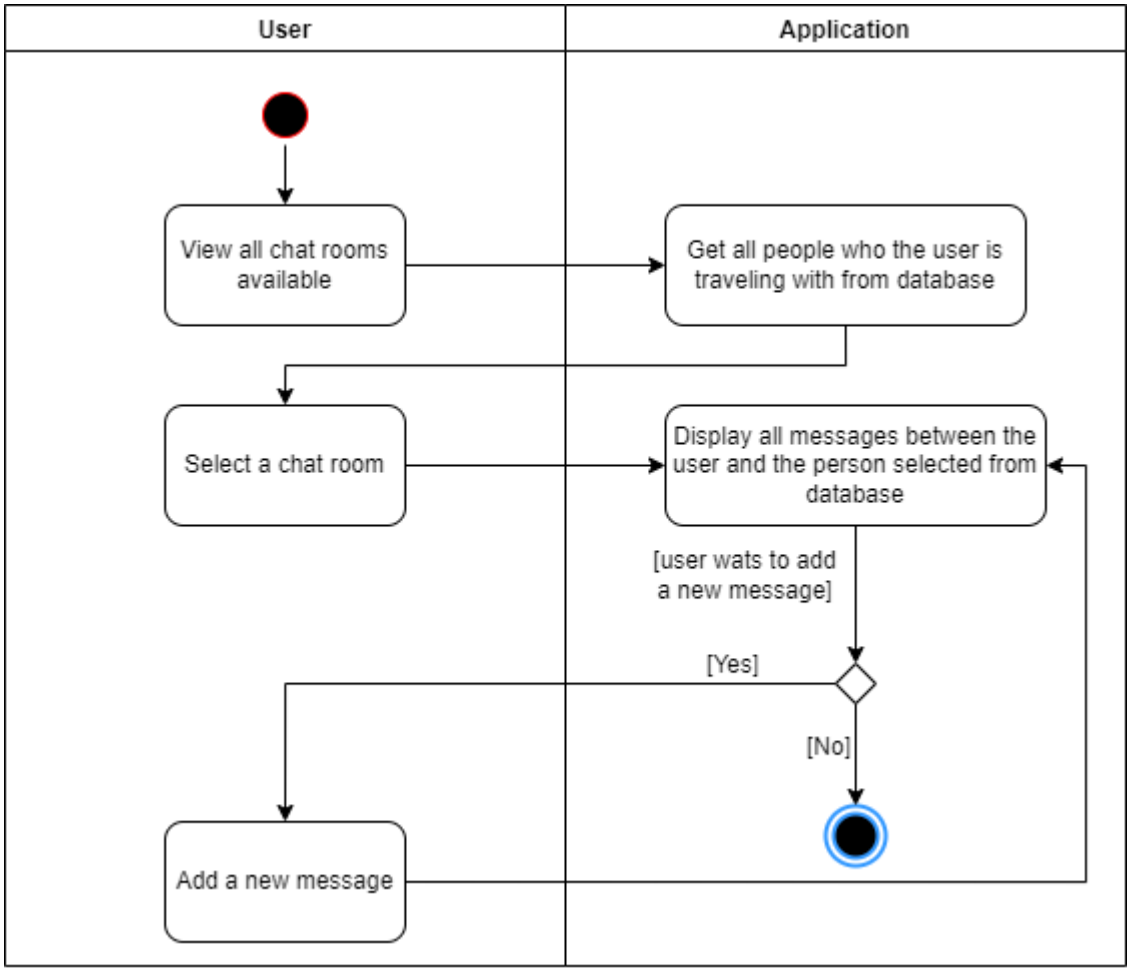

Figure 11: messenger activity diagram

3.3.5. Account functionality:

The account component allows the user to view and modify their account info, except for their phone number, star rating, number of travels and ID.

Account Activity diagram:

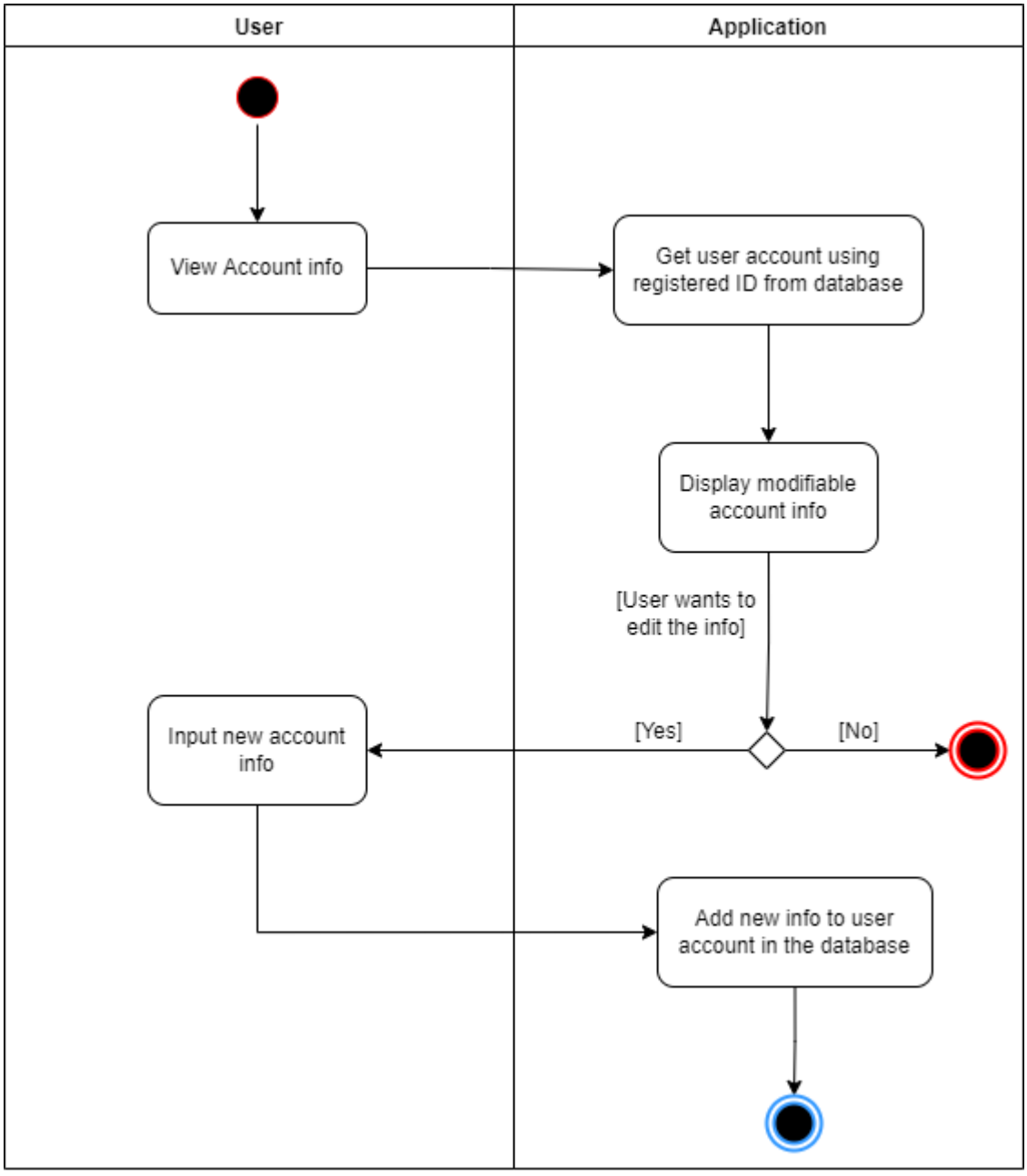

Figure 12: account activity diagram

4. Application class diagram:

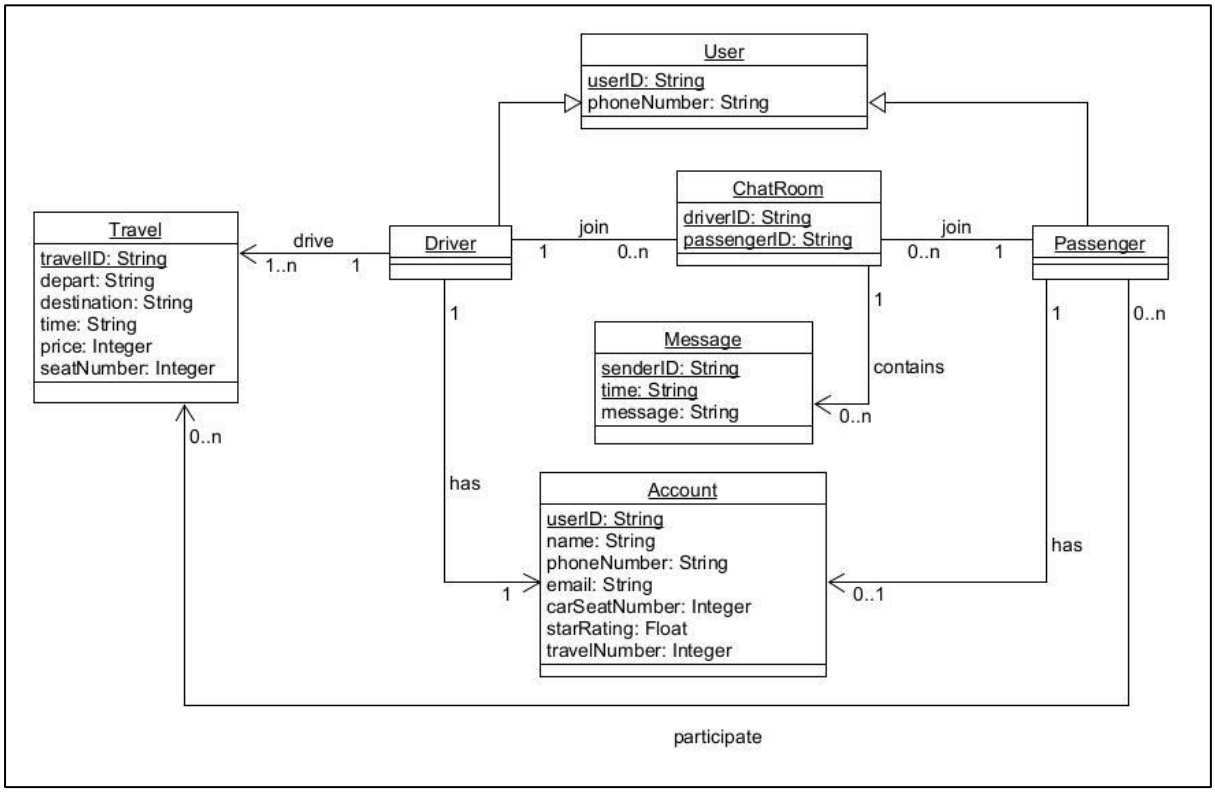

Figure 13: application class diagram

Conclusion:

In conclusion, the implementation of this app will require several tools including:

- Database.
- Storage.
- Authentication.
- searching for real life locations.
- Ways to display user and travel info.
- Navigating between app functionalities.

All of which are covered in the next chapter.

Chapter 4: Implementation

Introduction:

In this chapter, we will review the technologies used in the app, the two usage modes and the implementation of all functionalities discussed in the previous chapter.

1. Technologies used:

The application was developed in java using the latest version of Android Studio. The theme used is extended from the theme "Theme.Material3.Light"[reference 7], where some small changes to some views have been made.

Several other tools were used which include:

- **Firebase:** a platform that provides backend services for developing mobile and web applications.[reference 8]
- **Android jetpack:** a suite of libraries that simplify several aspects of android development.[reference 9]
- **Google Maps SDK:** software development kit for using Google map's services. [reference 10]
- **Dialog fragments:** A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.[reference 11][reference 12]

1.1. Firebase:

A backend-as-a-service (BAAS) platform that offers various backend services like databases and storage, provided by Google for web and mobile apps. Available in two tiers: a free "Spark" plan where the user has limited access to some of the functionalities, and a pay-as-you-go "Blaze" plan where the user has full access to all functionalities and are charged based on their usage.[reference 13] This app was developed using the free plan.

To use Firebase, a backend project needs to be linked to the android project and the dependencies to the relevant Firebase functionalities need to be added. During the linking process a JSON file (JavaScript Object Notation) with the Firebase Project properties is generated and must be added to the android project, so that the developer can access the Firebase functionalities by calling the static method getInstance() from the relevant functionality class e.g., FirebaseDatabase.getInstance().[reference 14]

This app uses three Firebase functionalities, which are:

- Firebase Authentication.
- Firebase Realtime Database.
- Firebase Storage.

1.1.1. Firebase Authentication:

Firebase Authentication provides a secure way to authenticate users using industry standards like OAuth 2.0 and OpenID Connect. A user can be authenticated using several identifiers like

a phone number, email, Google, Facebook…[reference 15] this app uses the phone number authentication. When a user is authenticated, an ID is generated and is saved alongside the authenticator (phone number in this case) in the Firebase Auth Users database.

1.1.2. Firebase Realtime Database:

The Firebase Realtime Database is a cloud-hosted, NoSQL database where data is synchronised in real time. The NoSQL (Not Only SQL) part means that the database structure is different from the one used in relational database, in this case the data is represented by key-value pairs.

From the point of view of the developer, the Realtime database is essentially two JSON files: one to contain the data, and the other to contain usage rules.

The rules file is necessary to keep the integrity of the data and to limit who can or cannot read/write to the database.

Due to the nature of JSON files, data will be stored in a tree structure, where a path from the root must be provided to either read or write the values. To write lists of values, the method push() which will generate a new key to store an element of the list, can be used.

Reading from the database can be done in a one-off manner, or in synchronous way where any change to the database is reflected in the app.[reference 16]

1.1.3. Firebase Storage:

Firebase Storage is built for app developers who need to store and serve user-generated content, such as photos or videos, without worrying about things like network quality.

Files can be uploaded and downloaded from the Firebase Storage using a path from the storage's root, where the file has to be converted to an array of bytes and the app has to provide Firebase Storage with the name and extension of the file.[reference 17]

1.2. Android Jetpack:

Android Jetpack is a Library suite for implementing various app components in a stable and uniform way. This app uses the following components:

- **Appcompat:** allows access to new APIs on older API versions of the platform, added as replacement for support libraries.
- **Fragment:** modular UI controllers that are hosted on an activity.
- **Navigation:** framework to navigate between destination in the app using a navigation controller and navigation graphs.
- **Work Manager:** an API used to schedule deferrable, asynchronous tasks (workers) that must be run reliably, added as a replacement for the Job Scheduler.
- **Card View:** a widget for displaying data in a rounded corner layout along with a specific elevation.
- **Constraint layout:** a layout that positions view relative to their surrounding using constraints, added to replace nested layouts.

- **Recycler View:** Creates dynamic lists with custom layouts. Can be used by defining a layout for the rows and implementing an adapter to populate the rows.
- **View Pager:** enables swiping between views or fragments.

Jetpack can be used by adding the relevant dependencies.

1.3. Google Maps SDK:

Google Maps can be added to the android project by following three steps:

- Creating a Google Cloud project.
- Getting the Maps API key from the Cloud project and adding it to the android manifest file.
- Enabling Google Play Services in the project and adding the relevant dependency.

The SDK can then be used to implement the OnMapReadyCallback interface which can be used to populate a Support Map Fragment with an actual map.

2. Sign in Activity implementation:

The Sign in Activity acts as a gatekeeper between the user and the Main Activity, where the user has to define their state (“User”, “Anonymous”) before the Main Activity is launched using an intent. This activity hosts three fragments which are:

- Banner fragment.
- Sign in choice fragment.
- Create account fragment.

2.1. Banner fragment:

The Banner fragment displays the apps logo and has no component that the user can interact with.

Figure 14: banner fragment screenshot

This fragment’s behaviour changes depending on two sequential questions:

- Is there a phone number registered in the app?
- Is there an account in the database that is associated with said phone number?

Which leaves us with four scenarios detailed in the table below, where the columns are the answers to question 1 and the rows are the answers to question 2:

	Yes	No
Yes	Main Activity is launched directly with the state “User”.	The user is asked to registered **the** phone number, then Main Activity is launched with the state “User”.
No	The app navigates to the Sign in choice fragment.	The user is asked to registered **a** phone number, then the app navigates to the Sign in choice fragment.

Figure 15: sign in scenario table

The app registers the phone number using a FirebaseUI activity, a pre-built authentication UI flow that uses the Firebase Authentication SDK.[reference 18] FirebaseUI conforms to various security standards like SafeNet and OAuth2.[reference 19] [reference 20] When a phone number is registered, an ID is generated to be used as the user’s ID.

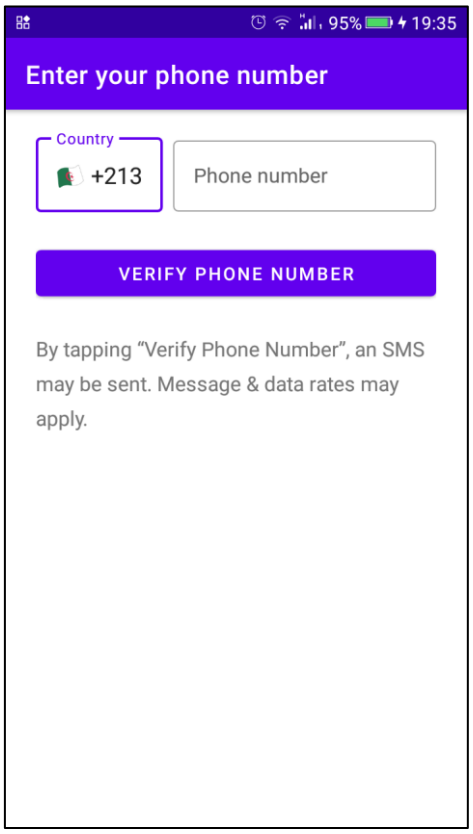

Figure 16: FirebaseUI activity screenshot

2.2. Sign in choice fragment:

This fragment contains two buttons for the user to choose from:

- **Continue as guest:** This launches the Main Activity with the state “Anonymous”.
- **Create account:** This navigates to the Create account fragment.

2.3. Create account fragment:

This fragment contains several text fields, a Circular image view[reference 21] and a save button. The fields require information about the user like email, name, car seat number…

Tapping the circular image view will ask the OS to launch an image browsing app using an intent filter, and when an image is selected (under 1 megabyte) it's loaded into the circular image view using the Glide library.[reference 22]

When all fields are full and the user presses the save button, the fields are used to create a User object and saved to the Firebase Realtime Database using the phone number's generated ID (referred to as the user ID from now on) as the index. If an image was selected when tapping the circular image view when the user tapped the save button, said image is uploaded to the Firebase Storage as a jpg with the user ID as its name.

The path to a user's account data is "/users/{User-ID}".

When the account is created, the Main Activity is launched with the state "User".

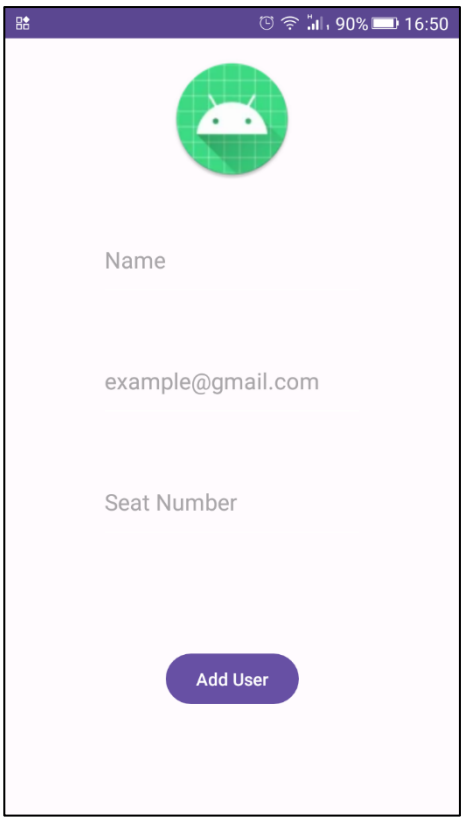

Figure 17: create account fragment screenshot

3. Main Activity implementation:

When the activity is first launched, it gets the user's account info from the Firebase Realtime Database and saves it to the account ViewModel and loads the user's uploaded picture into a Circular image view in the toolbar. The activity will host five collections of fragments that fulfil the five main functionalities, each starting with a fragment of the same name as the collection, and these collections are:

- Book fragments.
- Publish fragments.
- My rides fragments.
- Messenger fragments.
- Account fragments.

The navigation between the fragments is done using a nav controller with a navigation graph as it's blueprint.

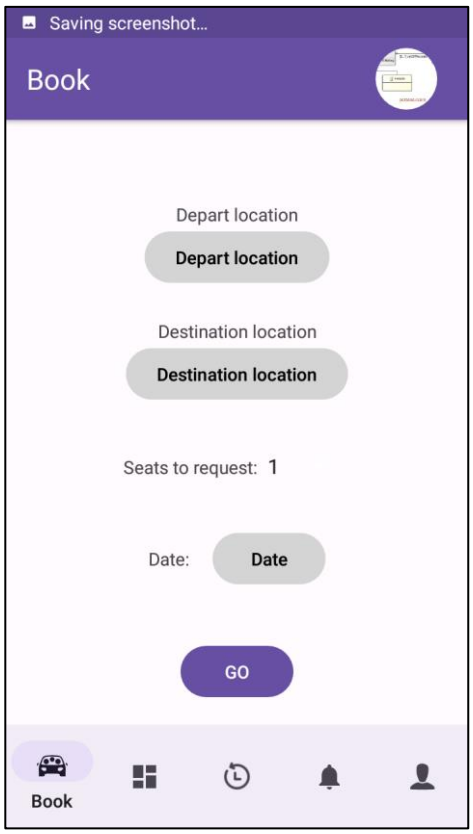

Figure 18: main activity screenshot

The graph contains five sub-graphs (“Book-navigation”, ”Publish-navigation”, ”My rides-navigation”, ”Messenger-navigation”, ”Account-navigation”) each with their own destinations. The user can access these sub-graphs through a “Bottom Navigation View”.

The app will also send and receive notification when certain events happen.

3.1. Book fragments:

There are three destination fragments in this collection: Book, Map search, and Ride search.

3.1.1. Book fragment:

This fragment contains three information buttons, a spinner and a search button.

Tapping either location button (depart or destination) will navigate to the Map Search fragment where the user can search for a location and save it to the Book ViewModel.

Tapping the date button will open the Date Dialog where the user can set the depart date.

Selecting the spinner will open a drop-down list where the user can set the number of seats to requests.

Tapping the search button when all travel info (depart, destination, date and seats to request) have been saved to the Book ViewModel will navigate to the Ride Search fragment.

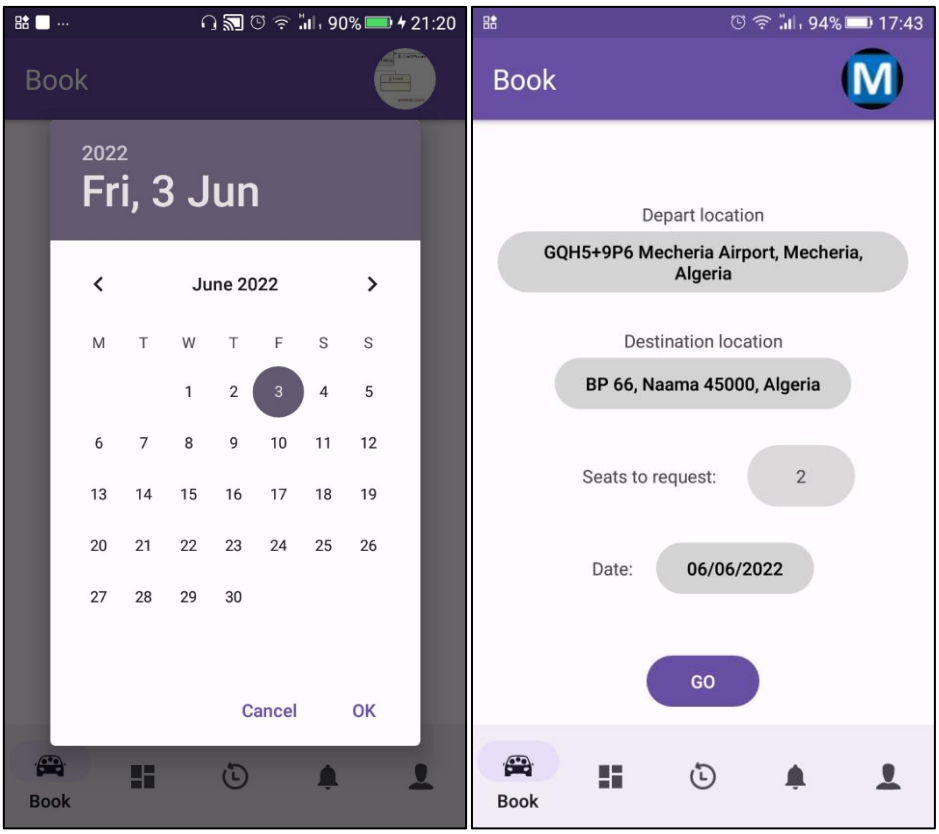

Figure 19: book fragment screenshot

3.1.2. Map Search fragment:

This fragment contains a search view, support map fragment and a floating action button. The code used in this fragment is a modified version of a Geeks-for-geeks tutorial page source code.[reference 23]

Map Search fragment uses the Google Maps SDK to load a map into the support map fragment (a fragment that turns the data provided by Google maps into a visible map). To find a location the user has to type the location name and press enter, the name is then passed to Google map's geo coder which returns a list of addresses that might correspond to the name passed. The fragment then takes the first address (closest possible match) and passes it to the support map fragment which puts a marker at the location specified.

Tapping on the floating action button when a location is specified will save the address to the Book ViewModel and return to the parent fragment.

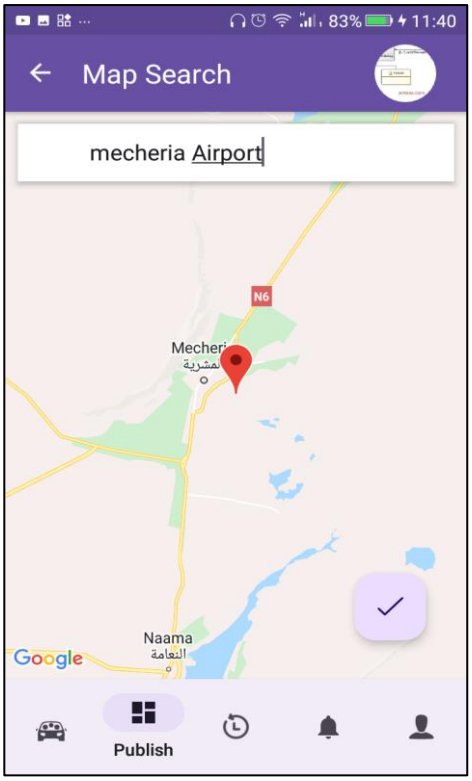

Figure 20: map search fragment screenshot

3.1.3. Ride Search fragment:

When this fragment is opened, it searches the Realtime Database for all travel records that match the data in the Book ViewModel then it displays them in a Recycler view alongside their driver's info (profile picture, name, star rating).

When the user taps one of the records, a dialog will ask them if they want to send a travel request to the travel record's driver.

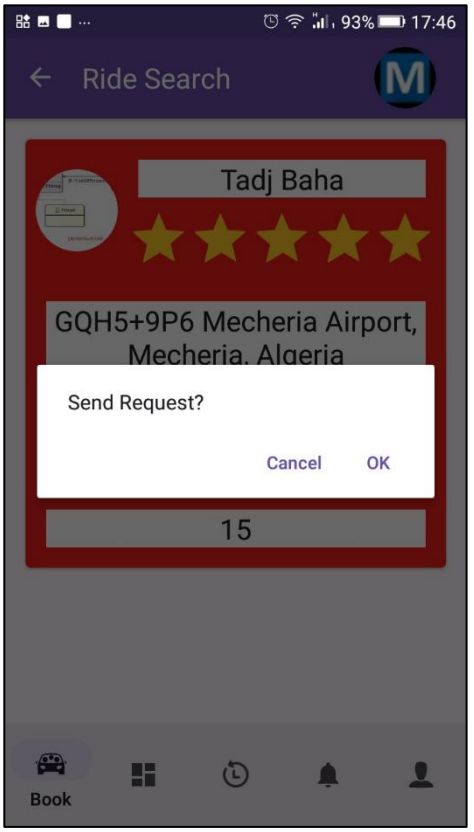

Figure 21: ride search fragment screenshot

3.2. Publish fragments:

This collection contains two destination fragments: publish and Map Search.

The Map Search in this collection is the same as the one in the Book fragments collection.

The procedure for filling the publish fragment is similar to the Book fragment, except that the date dialog will be followed by a time dialog and the data is saved in the Publish ViewModel.

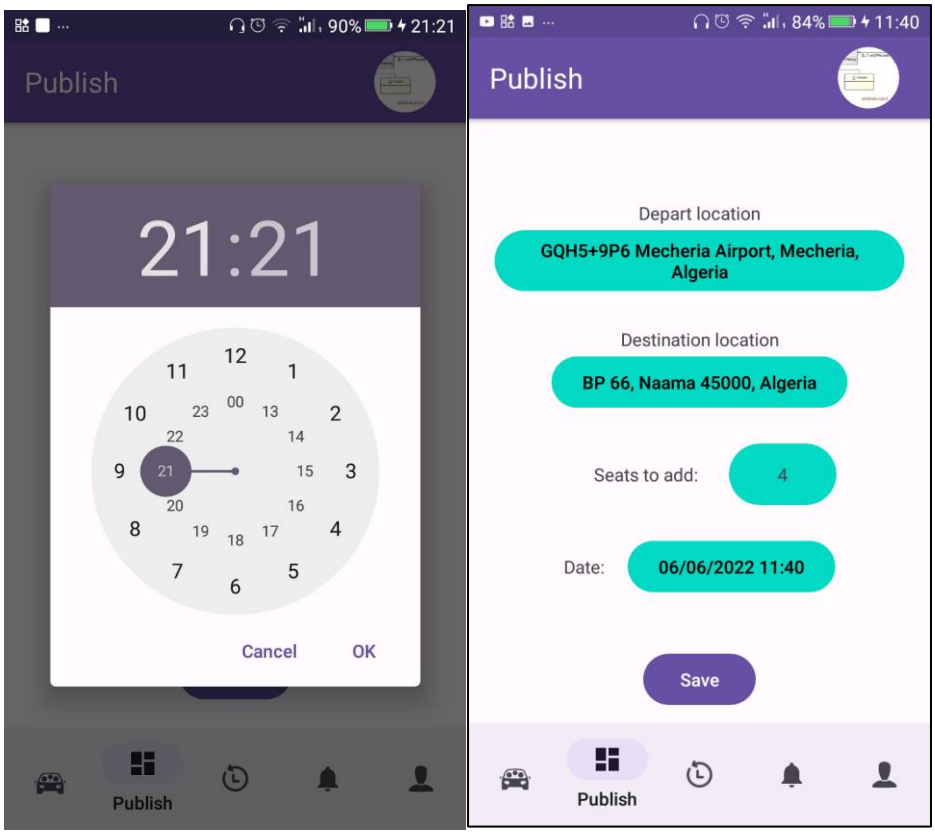

Figure 22: publish fragment screenshot

When the user taps the save button, the data from the Publish ViewModel is used to create a travel record and save it to the Realtime Database. The path for the travel records in the database is "/travelRecords/{driver-ID}/{travel-ID}".

3.3. My rides fragments:

This fragment contains five destination fragments: My rides, Started rides, Started rides choice, Started rides requests, Started rides participants, Joined rides and Ride requests.

3.3.1. My rides fragment:

The My rides fragments hosts three fragments: Started rides, Joined rides and Ride requests. This is done by using a View Pager, Tab layout and a Tab Adapter.

The Tab layout is a container that can contain a variable number of fragments which can be accessed by selecting tabs.

The Tab Adapter is an adapter that is defined by the developer, whose job is to set which fragments populate the Tab layout.

The View Pager allows for swiping between the fragments defined in the Tab Adapter.

3.3.2. Started rides fragment:

This fragment shows all the travels that the user has published by using a Recycler view.

When the travel date has passed, a dialog will prompt the user to give star ratings to all the participants and then deleting the travel record.

Tapping on a travel will navigate to Started rides choice fragment, with the travel ID passed to it.

Figure 23: started rides fragment screenshot

3.3.3. Started rides choice fragment:

This fragment contains two buttons: the check requests one will navigate to the Started rides requests fragment, and the view participants one will navigate to the Started rides participants.

3.3.4. Started rides requests fragment:

This fragment contains a Recycler view that lists all the requests sent to the travel record previously selected.

Tapping on one will show a dialog where the user is prompted to either accept or deny the request.

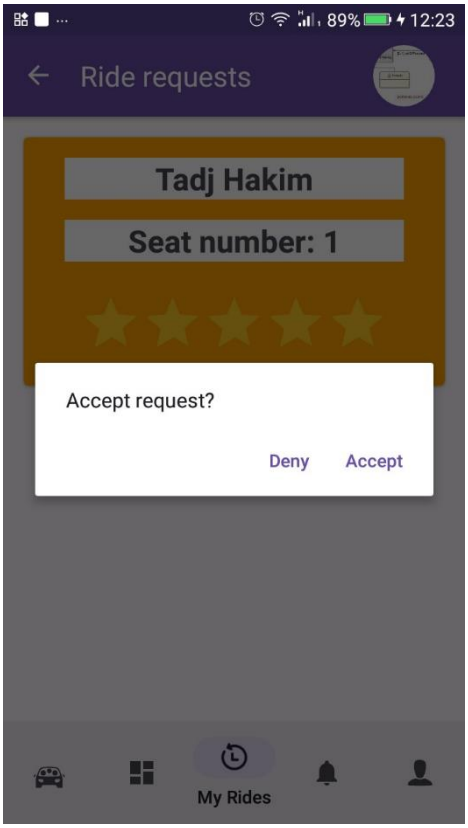

Figure 24: ride requests fragment screenshot

3.3.5. Started rides participant's fragment:

This fragment contains a Recycler view that lists all the participants in the travel record previously selected.

Tapping on one will show a dialog which asks if the user wants to remove the selected participant.

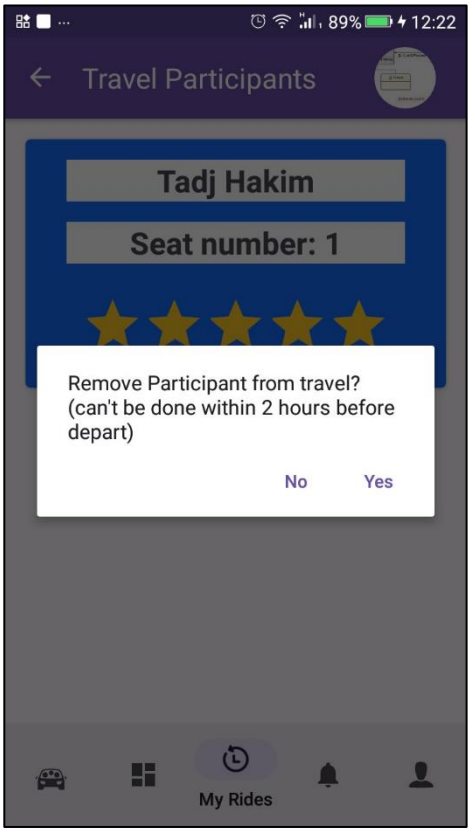

Figure 25: travel participants fragment screenshot

3.3.6. Ride requests fragment:

This fragment contains a Recycler view that lists all the request sent by the user. Tapping a request will show a dialog that asks the user if they want to cancel their request.

If the request is yet to be answered, then the Card view's colour is set to grey.

If the request is denied, then the Card view's colour is set to red.

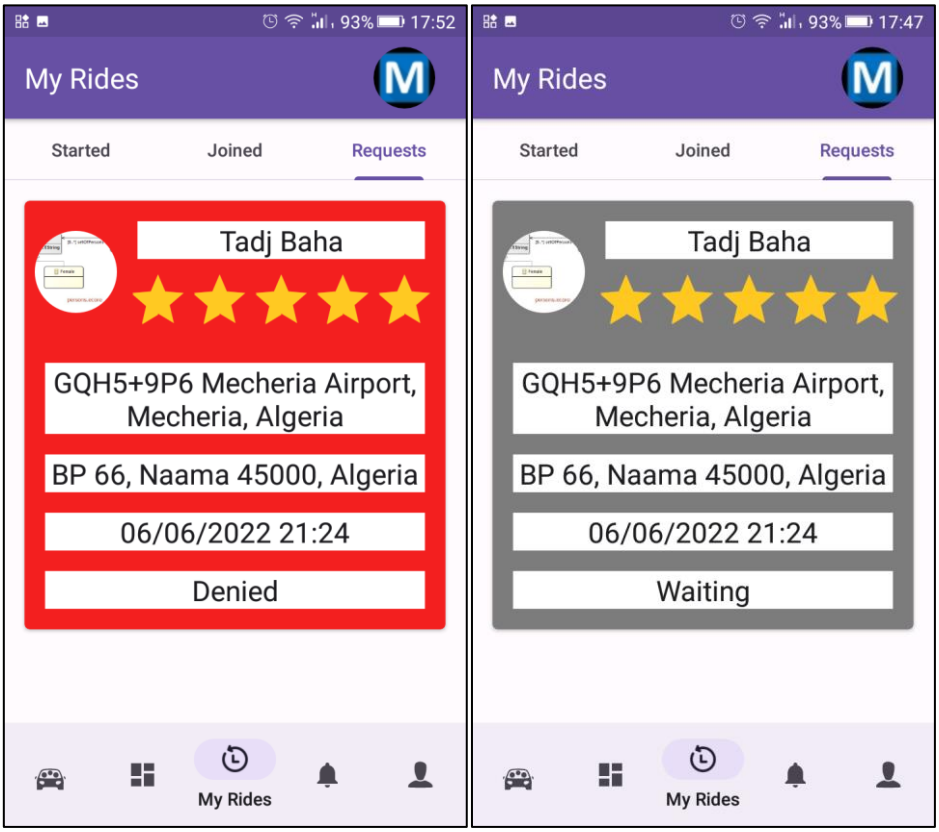

Figure 26: requests fragment screenshot

If the request is accepted, then the request is removed from this fragment and the travel is added to the Joined rides fragment's list.

3.3.7. Joined rides fragment:

This fragment contains a Recycler view that lists all the travels that the user is participating in. Tapping a travel will show the drivers contacts.

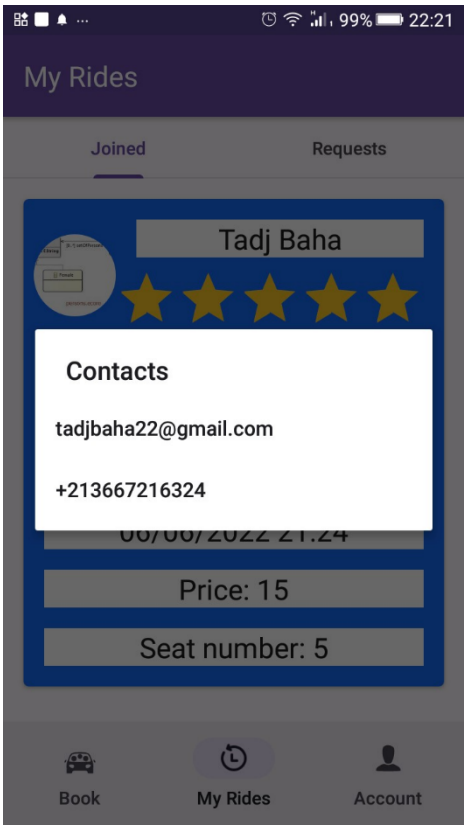

Figure 28: joined rides fragment screenshot

3.4. Messenger fragments:

This collection contains two destination fragments: Messenger and Chatroom.

3.4.1. Messenger fragment:

This fragment lists all the people that the user is travelling with. When a travel request is accepted, both the driver and the passenger add the other to their chat room list in the database, which is then used by this fragment to indicate who is the driver and who is the passenger in a particular chat room.

Tapping an element will navigate to Chatroom fragment with the selected participant's ID passed as argument.

Figure 27: messenger fragment screenshot

3.4.2. Chatroom fragment:

This fragment displays the profile picture and name of the selected participant from the previous fragment at the top, followed by a Recycler view, and finally a text field and a send button at the bottom.

The Recycler view displays messages in order of release, with the user's messages on the right and the other participant's on the left. The messages contain the message text, followed by the release date and time. The fragment uses a change listener to detect changes in the Realtime Database and add new messages in real-time.

The text field is where the user can input messages, and send them by tapping the send button. The app uses the ID of the travel driver followed by the specific participant's ID to index the messages in the database. The path of the messages in the database is "/chats/{driver-ID}/{participant-ID}/".

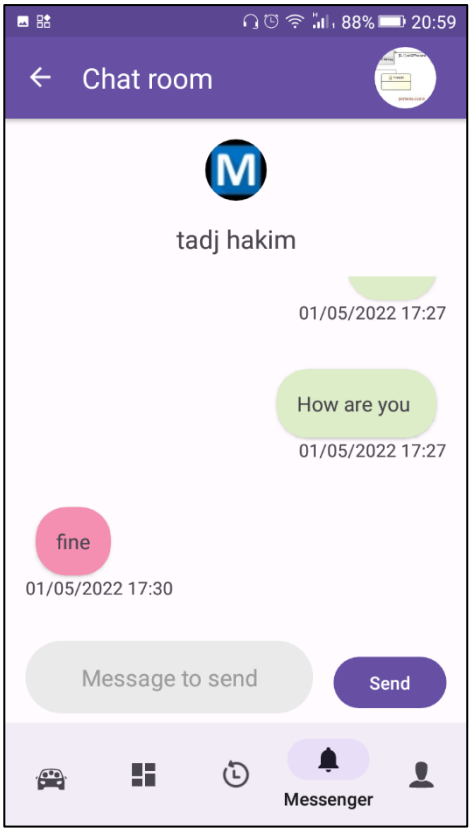

Figure 29: chat room fragment screenshot

3.5. Account fragments:

This collection contains three destination fragments: account, view account, edit account.

3.5.1. Account fragment:

This fragment contains two buttons: a log out button that logs out of the current Firebase Authentication session and launches the Sign in Activity, and a view account button that navigates to the View account fragment.

3.5.2. View account fragment:

This fragment displays the account info, alongside the profile picture, and an edit button.

The edit button navigates to the Edit account fragment.

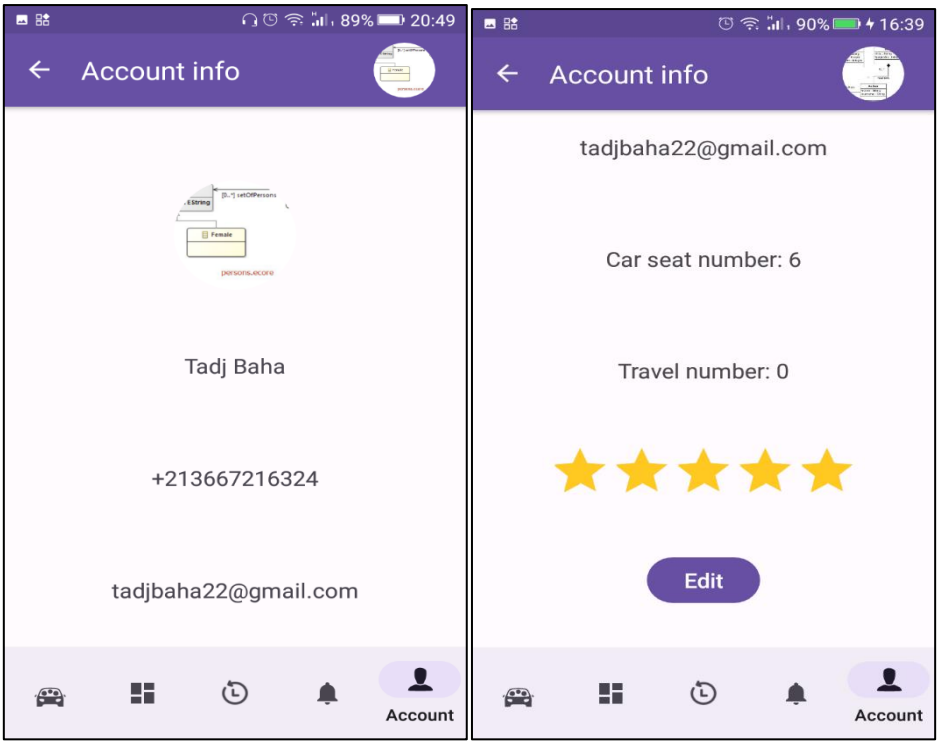

Figure 30: view account fragment screenshot

3.5.3. Edit account fragment:

This fragment contains a circular image view, several text field and an update button.

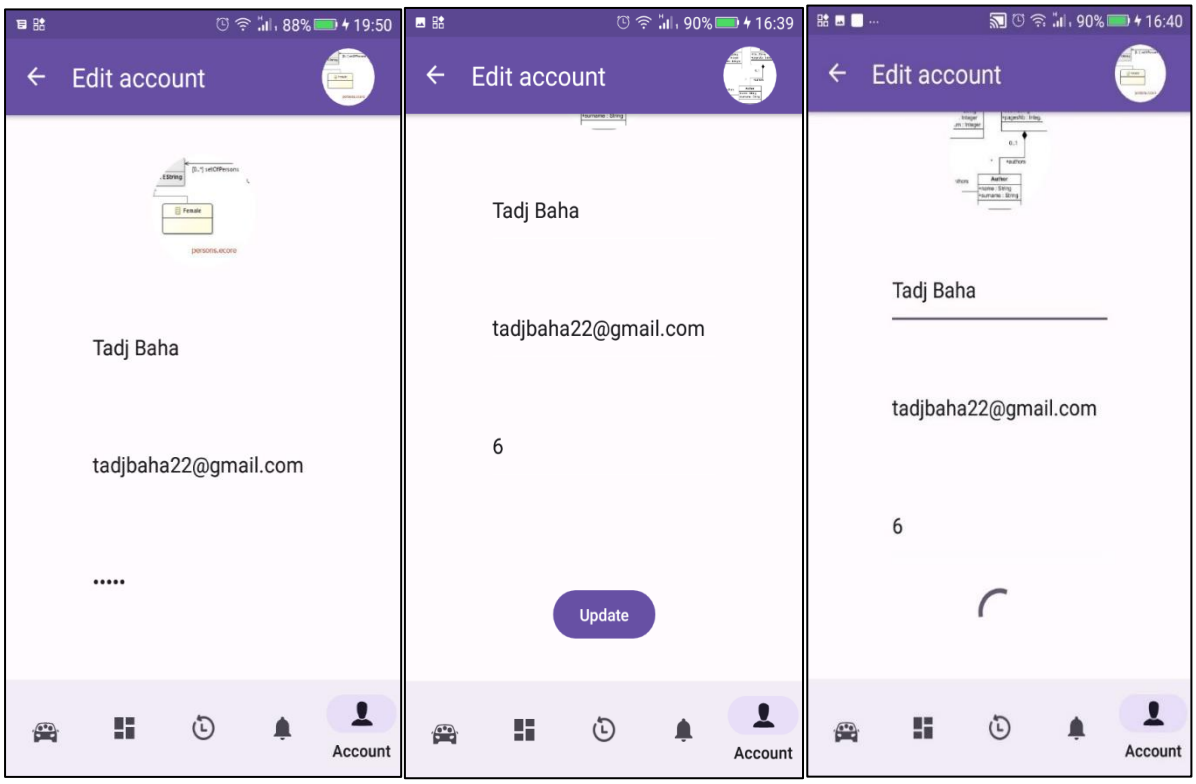

Figure 31: edit account fragment screenshots

The field are all the account info that can be modified (the star rating, phone number and the number of travels done are unmodifiable).

Tapping on the circular image view allows changing the image like in the Create account fragment.

Tapping the update button will save the new account info, and if the user changed the image, the app will prevent the user from interacting with it until the image is uploaded.

3.6. Notifications:

The app will need to show notifications when certain events happen (e.g., a travel request accepted). In the “Blaze” plan, this can be done by using Firebase Functions to send messages through Firebase Cloud Messaging when certain changes happen in the Realtime Database.

This app, however, will use a more manual method:

1. When the notification events happen, the app will also add the notification info to the Realtime Database in the path ”/notifications/{receiver-ID}/”.
2. The app will launch a worker named Notifications Worker every time the user uses the bottom navigation view.
3. The Notifications Worker checks the user’s notification node in the database and creates the notifications using the data in the node. The worker will then delete all the notification nodes in the database that have already been made into actual notifications.

A worker is a deferrable and asynchronous service that runs in the background. Workers are scheduled and run using the Work Manager.

4. Anonymous Main Activity implementation:

The functionality provided by the Main Activity is decided by the state passed to it by the intent. If the state is “Anonymous” then the activity will contain only three sub-graphs: Book fragments, My rides fragments and a “Not logged in” fragment.

The Book fragments are the same as the one in the “User” state, except that the travel requests that are sent are given the state “Anonymous”.

Since the anonymous user can’t publish travels, the Started rides tab is not visible.

The “Not logged in” fragment contains two buttons: a log out button that logs out of the Firebase Authentication session and launches the Sign in Activity (leading to the FirebaseUI Activity), and a create account button that directly launches the Sign in Activity (leading to the Create account fragment).

The anonymous user must Communication with a joined ride driver using external ways like calling their phone number or emailing them. The anonymous user can access those contacts by tapping the travel record the Joined rides tab.

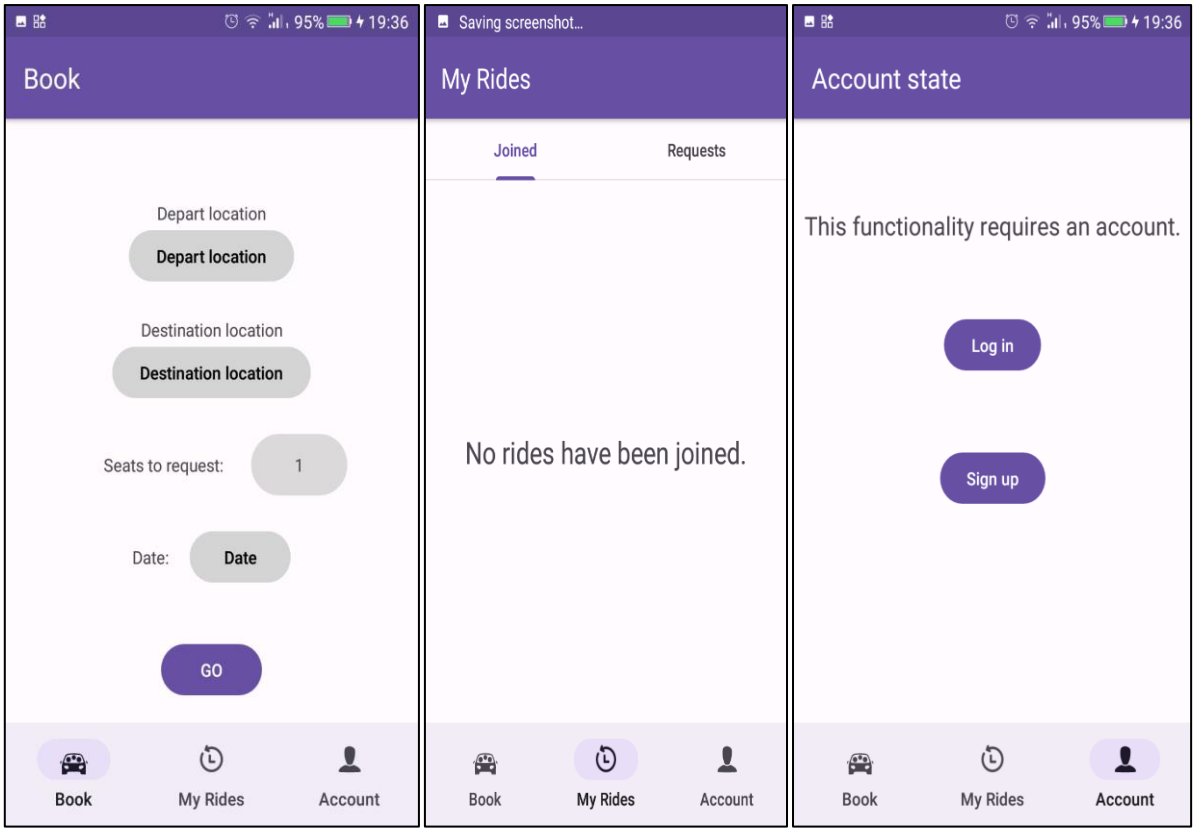

Figure 32: anonymous main activity screenshots

Conclusion:

At the end of the implementation phase, we end up with an that lets the users start, manage and finish travel plans. Users can also choose who to travel with, using their star rating. Communication can be done using the in-app messenger for users with accounts, or using external means if they're anonymous users.

Since it has to keep the travel data up to-date at all times, the app requires internet connectivity to start working.

General Conclusion:

In this work we have documented the development of an Android Carpooling app. The reason for this being that the future of this mode of transportation is clearly online, especially if its simplified by an application that can be used by anyone with an Android phone, one of the most common devices around.

And carpooling isn’t going away, especially because of climate change and rising living costs, preserving the environment and saving costs will only get more important.

Resources:

- Book - Head first Java – by Bert Bates and Kathy Sierra.

- Book – Java notes for professionals - https://goalkicker.com/JavaBook/JavaNotesForProfessionals.pdf

- Book - Head First Android Development: A Brain-Friendly Guide – by David Griffiths and Dawn Griffiths.

- Guides site – Official Android developer guides - https://developer.android.com/guide

- Documentation site – Official Android API reference - https://developer.android.com/reference

- Guides site – Official Firebase guides - https://firebase.google.com/docs/guides

- Guides site – Code Path Android Cliff notes - https://guides.codepath.com/android

- Forum site – Stack Overflow android questions - https://stackoverflow.com/questions/tagged/android

- Guides site – Android developers articles on Medium - https://medium.com/androiddevelopers

References:

1) Android statistics 2022 - https://www.businessofapps.com/data/android-statistics/

2) Android definition - https://en.wikipedia.org/wiki/Android_(operating_system)

3) Android architecture - https://source.android.com/devices/architecture

4) Android app components and definition - https://developer.android.com/guide/components/fundamentals

5) Fragment manager - https://developer.android.com/guide/fragments/fragmentmanager

6) Activity life cycle - https://developer.android.com/guide/components/activities/activity-lifecycle

7) Material theme - https://github.com/material-components/material-components-android/blob/master/lib/java/com/google/android/material/theme/res/values/themes.xml

8) Firebase - https://firebase.google.com/

9) Android jetpack - https://developer.android.com/jetpack

10) Google maps SDK - https://developers.google.com/maps/documentation/android-sdk/overview

11) Dialogs - https://developer.android.com/guide/topics/ui/dialogs

12) Dialog fragments - https://developer.android.com/guide/fragments/dialogs

13) Firebase pricing - https://firebase.google.com/pricing

14) Firebase setup - https://firebase.google.com/docs/android/setup

15) Firebase authentication - https://firebase.google.com/docs/auth/android/start

16) Firebase database - https://firebase.google.com/docs/database/android/start

17) Firebase storage - https://firebase.google.com/docs/storage/android/start

18) Firebase UI - https://github.com/firebase/FirebaseUI-Android

19) Safe net - https://cpl.thalesgroup.com/access-management/safenet-trusted-access

20) OAuth2 - https://oauth.net/2/

21) Circled image view- https://github.com/hdodenhof/CircleImageView

22) Glide library - https://github.com/bumptech/glide

23) Geeks for geeks map search - https://www.geeksforgeeks.org/how-to-add-searchview-in-google-maps-in-android/