# Carpoolio


MUHAMMAD TAHA SHAHID
**01-134201-063**

Supervisor: Sabira Feroz

**Bachelor of Science in Computer Science**

Department of Computer Science
Bahria University, Islamabad

October 2025

# Certificate

We accept the work contained in the report titled "TITLE OF THE REPORT", written by Mr. AUTHOR1 NAME AND Mr. AUTHOR2 NAME as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by . . . :

Supervisor:

_____

Internal Examiner:

_____

External Examiner:

_____

Project Coordinator:

_____

Head of the Department:

_____

# Abstract

Carpoolio is a cross platform mobile application with the aim of facilitating carpooling in the urban areas of Pakistan. The principal target group is students and office commuters who pass through these locales on a daily basis. Such users tend to follow fixed paths and suffer from the problems of high costs of transportation, traffic congestion, and the absence of alternate convenient travel options. Carpoolio tries to solve these problems by promoting sharing of rides in a simple and reliable manner. The application is based on Flutter which provides the ability to work on multiple platforms from a common code without any issues. Firebase is used at backend, which handles the authentication, database transactions, and real-time data handling. In addition, Google Maps services are also integrated to enable location tracking, route visualization, and geo-location functionality. The integration of these technologies together ensure optimum performance as well as smooth user experience.

Carpoolio provides a platform to enable drivers to provide rides, and passengers to find rides and join existing rides within the same city. The facilitation of shared rides reduces the transportation costs of the user and at the same time reduces the traffic congestion as a result of reducing the number of vehicles on the road. Consequently, the same leads to environmental benefits. The current report describes the general system design and technical architecture of the application, including the design of the user interface, the flow of navigation, and the database schema. The user interface is designed in such a way that it is simple and user-friendly which helps in user interaction with the application without any hindrance and the flow of navigation helps in smooth transition from one succeeding screen.

As far as the backend is concerned, the process of registering and logging in users is employed using the services of the Firebase Authentication. Cloud Firestore is the primary database to store information of user data, ride details, and booking details. Geo-location services are facilitated using Google's Geocoding and Maps APIs that give the precise location information and route visualization. A comparative analysis is included in the report as well, where Carpoolio is compared to other platforms like DriveLu - a Pakistan-focused carpooling application - and inDrive, a global ride-hailing service. The document also addresses limitations such as lack of in-app messaging functionality, reliance on cash-based mechanisms of payment, and lack of sophisticated administrative tools.

# Acknowledgments

I would like to thank my supervisor Ms. Sabira Feroz, Lecturer, Department of Computer Science, Bahria University for her invaluable guidance and encouragement throughout the course of this project. Due to her insightful suggestions and constructive feedback, I was able to keep up with the standards required by the university for this project. Her patience in explaining complex concepts, willingness in clearing doubts and giving timely advice whenever required made a tremendous difference in the progress of this research.

I would also like to thank our project co-ordinator, Dr. Asim Ali for setting such high standards for our final year project. It was due to his commitment to getting the best results out of all the students, we were able to produce our best work.

I would also like to thank Dr. Arshad Farhad for his invaluable suggestions and feedback. Due to his sincere and thorough evaluation of my project, I was able to see where my project was lacking and hence was able to finish my project at the required academic standards.

I would also like to thank our Head of Department, Dr. Muhammad Khurram Ehsan, for his leadership and specifically for making it easier for students to approach and discuss their academic issues, and his commitment in helping the students in any way he can.

Finally, I would like to acknowledge the support of my peers, friends and family, who have been a constant source of encouragement and motivation. Their understanding, patience and moral support through the difficult stages of this project kept me going. I am very grateful for the help of all those who contributed directly or indirectly to the completion of this work.

MUHAMMAD TAHA SHAHID
Bahria University
E-8, Islamabad, Pakistan

October 2025

# Contents

# List of Figures

# List of Tables

# Acronyms and Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| GPS | Global Positioning System |
| iOS | iPhone Operating System |
| SQL | Structured Query Language |
| NoSQL | Not Only SQL |
| UI | User Interface |
| UX | User Experience |
| CAGR | Compound Annual Growth Rate |
| IEA | International Energy Agency |
| CLI | Command Line Interface |
| FR | Functional Requirement |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| ERD | Entity-Relationship Diagram |
| GUI | Graphical User Interface |
| MVP | Minimum Viable Product |
| OOP | Object-Oriented Programming |
| CRUD | Create, Read, Update, Delete |
| TC | Test Case |
| PII | Personal Identifiable Information |
| GDPR | General Data Protection Regulation |

# Chapter 1

# Introduction

## 1.1 Project Background/Overview

Urban transportation systems are confronting many serious problems today all over the world. One of the primary causes of such problems is the rapid urbanization. Cities are expanding at a very rapid rate and immense number of people are going to urban areas for education, employment, and better living opportunities. Along with increase in population, use of private vehicles has also increased a lot. Most people prefer to use their own cars or bikes, rather than use public transport.

Because of this, traffic jams have become an issue in nearly every big city. Roads are congested in peak hours and people spend a lot of time stuck in traffic. This not only wastes time, but also causes frustration to daily commuters. Another major problem is the rise in the costs of fuel. Fuel prices continue to rise, making it expensive to travel daily, especially for students and office workers.

Environmental issues are also becoming more serious due to the increased use of vehicles. A large number of vehicles on the road leads to an increase in greenhouse gas emission and air pollution. This impacts the air quality in the cities and causes health issues for people. Smog, noise pollution and poor air quality are now common in many urban areas. As cities continue to expand these transportation related problems are becoming more difficult to manage.

Transportation solutions that are economical, ecologically friendly, and efficient are desperately needed in light of all these issues. The government is now taking interest in finding an environmentally clean mode of transportation that would be able to minimize the pollution and traffic rates and also serve the daily travel requirements of people.

One such solution has been carpooling. When multiple people travel in the same car in the same direction, it's known as carpooling. This straightforward idea can contribute to lowering the overall number of cars on the road. Carbon emissions, fuel consumption, and traffic congestion all decrease when there are fewer cars on the road.

People who travel frequently, like students and office employees, can benefit greatly from carpooling. Traveling becomes more affordable and less expensive when people share rides. Additionally, it encourages users to collaborate and engage socially. Carpooling has not become very common or widely adopted despite these advantages.

The absence of suitable digital platforms is one of the main reasons carpooling has not become popular. Manual ride coordination is difficult for many people. There are concerns associated with trust, safety and reliability in sharing rides with unknown people. Existing solutions are often complicated, not user-friendly or not easily accessible to everyone.

This is where Carpoolio comes in. Carpoolio is a mobile application designed to address these exact problems. It aims to provide a simple, secure and easy-to-use platform on which drivers and passengers can connect with each other for shared rides. The application is therefore limited to making carpooling easy for everyday users.

Carpoolio makes use of state-of-the-art mobile development technologies and cloud-based infrastructure to ensure a smooth performance and reliability. The application is built with a huge user experience focus with ease of user to find, offer and join rides. Special attention is paid to establishing trust in the community through clear role of users and transparent ride details.

By keeping the focus on simplicity, reliability and trust by the community, Carpoolio's goal is therefore to make carpooling a practical choice for daily commuting. The goal is not only to develop an application, but to foster a transport culture that is shared and that will not only works towards a cleaner environment but contribute to reducing traffic issues, and saving money.

## 1.2 Problem Description

Traditional transport networks within urban centres are gradually getting inefficient and unsustainable over time. With the expansion of cities and growth of population, the available transport infrastructure fails to accommodate the daily transportation needs. The heavy usage of single-occupancy vehicles is one of the largest contributors to this problem. The majority of the population is used to moving individually in their own cars or bikes, thus this makes the amount of cars on the roads very high.

The prevalence of single occupancy vehicles results in serious traffic jams particularly during rush hours. Roads are over-loaded, travelling time is prolonged and it is stressful and exhausting to commute daily. This situation promotes air and noise pollution besides the congestion of traffic. Many of the cars emit toxic gases to the environment, which has adverse effects on the quality of air and health. Traffic noise also contributes to the discomfort of the urban dwellers.

Along with negative environmental outcomes, the use of a personal vehicle poses a high financial burden on a person. The price of gasoline is ever-increasing thus commuting

is quite costly. There is increased financial pressure in the form of vehicle maintenance expenses, including repairs and servicing. The cost of parking in cities is also expensive. All these make the use of private transport an exorbitant and inefficient option to a large number of people.

Despite the fact that most cities have transport systems, they are not always satisfactory to the users. Public transport is typically on predetermined schedules that cannot necessarily fit all users in their day to day activities. The coverage is usually not extensive, and thus, most of the regions are not well connected. Another predominant problem is the issue of overcrowding particularly during rush hours, which makes traveling inconvenient and uncomfortable. Public transport is also deficient in last-mile connectivity in most instances, so the user is compelled to use alternative forms of transport to get to their ultimate destination.

The limitations encourage more individuals not to use the public transport and resort to personal vehicles, thus exacerbating the situation. This scenario creates the necessity to find alternative transportation solutions which are flexible, affordable and easy to use.

Carpoolio intends to overcome these deficits by offering a mobile-based service which promotes shared transportation. The application provides user-friendly nature since people can easily search for available rides or offer rides to other persons on the same route. Carpoolio assists in minimizing the number of cars on the road by encouraging people to use a car to share rides and allowing the current transportation means to be utilized more efficiently.

The app highly emphasizes verification of the users and security to maintain trust among the users. Authenticated profiles and transparent ride information make users more comfortable in sharing rides. Besides that, Carpoolio is oriented on simple and smooth user interface, which enables the user to easily navigate the application without any misunderstanding.

Carpoolio is not only a solution to transportation logistics by taking both technical and social issues into account. It also addresses the safety issues and trust problems that usually makes people discouraged to join carpooling. In this way, Carpoolio will make shared transportation a feasible, safe, and reliable alternative form of transportation on a daily commute.

## 1.3 Project Objectives

The Project Objectives are as follows:

- Provide a platform for users to offer and request carpool rides through dedicated driver and passenger interfaces.

- Enable seamless user authentication and account management using Firebase Authentication.
- Allow drivers to add rides with relevant details.
- Facilitate ride requests and matching between drivers and passengers.
- Offer role-specific features for drivers and passengers.
- Integrate Firebase for backend services such as authentication and data storage.
- Deliver a user-friendly and intuitive interface for all core functionalities.
- Provide real-time location tracking and mapping using Google Maps.
- Promote eco-friendly transportation and cost-sharing among users.
- Enable ride status management including start, in-progress, and completion states.
- Display ride history and trip details for both drivers and passengers.

## 1.4 Project Scope

### 1.4.1 In-Scope Functionality

- **User Account Management:** The system permits different user accounts for passengers and drivers with separate sign up and login using email and password. It includes strong verification of user type against Firestore collections ("users" vs "Drivers") to ensure role specific access and functionalities.
- **User Profile Management:** Both passenger and driver profiles are editable so that users can manage essential information such as email and phone number. Drivers are given some extra fields for car information such as model, seating capacity, and AC availability, which are filled in during sign-up.
- **Ride Offering (Drivers):** Ride offerings can be created and posted by drivers in an intuitive way by selecting start and destination points on an embedded map, date and time and amenities such as air conditioning. The application automatically computes and presents an estimated fare depending on the distance of the trip.
- **Ride Requesting (Passengers):** Passengers can make ride requests by specifying their desired start and end locations through map interaction or current location, and choosing a departure time. The system then goes on to query existing driver-offered rides and to present matching options within a predefined time and location proximity.

- **Booking Management:** Passengers can book a seat in a matched ride, which is recorded in a "bookedride" Firestore collection, which includes both driver and passenger details.
- **Ride Lifecycle Management:** The home screen gives an overview of all the upcoming rides of both drivers and passengers. Drivers have the ability to log rides to the "started" and "ended" status where their rides are tracked in the Firestore database.
- **Notifications and Feedback:** Through SnackBar messages, the app gives users instant in-app feedback for successful or unsuccessful actions.
- **Cross-Platform Compatibility:** Carpoolio is created with Flutter, which uses Material Design principles to provide a consistent user experience across desktop, iOS, and Android platforms from a single codebase.
- **Cloud Integration:** Cloud Firestore serves as a NoSQL document database for all application data, and Firebase Auth is used for user authentication. The backend infrastructure is entirely built on Google's Firebase platform. Address conversion and location handling are integrated with Google Maps and geocoding APIs.
- **Basic Security:** Firebase Auth is used to manage user authentication, and basic input validation is used to guarantee data integrity.

### 1.4.2 Out-of-Scope Functionality

- **In-App Communication:** Direct calling or integrated chat between drivers and passengers are not included in the current scope. Users are expected to co-ordinate outside the application through phone numbers provided.
- **Online Payment Integration:** It is assumed that all the fares transactions are handled offline through cash payments. The application does not have mobile wallets, credit/debit card processing or in-app payment gateway of any kind.
- **Advanced Administrative Controls:** The prototype does not have a separate administrator dashboard for moderating rides, managing users, or support requests. Backend management is currently only possible through manual operations using the Firebase Console.
- **Complex Matching Algorithms:** The ride matching logic is based on static thresholds on time and location. Advanced matching mechanisms, such as those using the AI or machine learning capabilities to suggest personalized trips, are beyond the scope.

- **Offline Functionality (beyond basic caching):** Full functionality of application is dependent on network connectivity. While Firestore provides some offline caching, it is not one of the core design considerations for core functionality.
- **Comprehensive Localization:** The application is English only at this point with future localization to regional languages such as Urdu is a potential enhancement.
- **Formal Automated Testing:** The project mainly depended on the manually performed unit and integration testing. Automated test suites (e.g. widget tests, integration tests) were not developed within the scope of the project.
- **Monetization Features:** Any features related to commission, subscription mod-els, or advertising are beyond the scope of this academic project.

## 1.5 Methodology

The Agile methodology has been applied to this project, as it is an iterative development with a high level of feedback and an adaptive plan to the project [1]. This strategy was adopted to accommodate evolving requirements and ensure close alignment with user needs in the dynamic urban transport landscape of Pakistan. The development was split into five iterations, each of which had a set of deliverables that needed to be completed and resulted in a working version of the application.

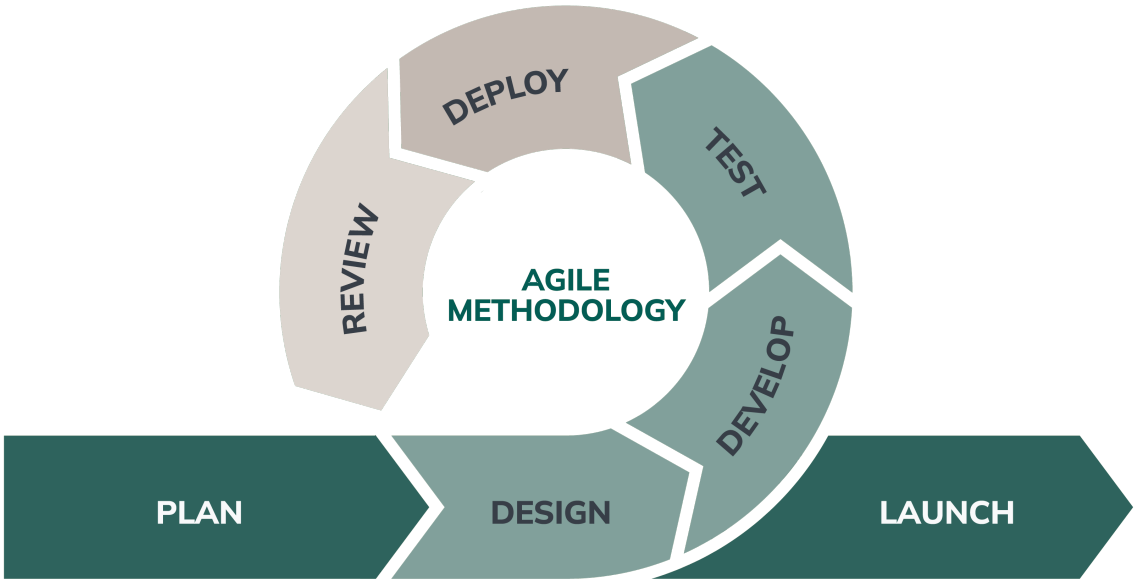


Figure 1.1: Agile Methodology Life Cycle Diagram

### 1.5.1 Iteration 1: Authentication

The first iteration was for setting up the base authentication system for Carpoolio. This phase introduced the basic user management capabilities required in order to achieve multi-role access control.

- **User Registration:** Implemented sign up functionality by using Firebase Authentication where users can create their accounts with the help of email and password verification [2].
- **Login System:** Created secure login system along with role-based access control, differentiating the account of passenger and driver.
- **Role Selection:** Created user role assignment system during registration, which enabled different interfaces and functionalities of the system based on the type of user.
- **Basic Firebase Setup:** Setup of a Firebase project structure of the and creation of an initial database collections for user management.

This iteration provided a working authentication slice upon which all the rest of the features are based.

### 1.5.2 Iteration 2: Driver Flow

The second iteration was to work on the driver side functionality, allowing drivers to create and manage rides.

- **Ride Creation:** Implemented ride posting functionality with automatic calculation of fare on the basis of estimation of distance.
- **Ride Management:** Created the interface for drivers to access their current active rides, passenger bookings and the ride status.
- **Location Services:** Used geolocator API so that the drivers can track their location in real-time.

This iteration provided full driver flow that allows ride creation and management.

### 1.5.3 Iteration 3: Passenger Flow

The goal of the third iteration was to develop the entire passenger experience, including booking and ride-finding. The fundamental passenger-related features were implemented during this phase.

- **Ride Search:** Using Firestore queries, ride searching functionality with time and location proximity filtering was developed.
- **Map Integration:** To choose a location and see a route, a Google Maps widget was integrated [3].
- **Booking System:** A ride booking workflow was implemented, enabling passengers to reserve seats and create booking records.
- **Booking Management:** Developed an interface that allows travellers to see their current and previous reservations along with pertinent information.

This iteration provided a complete passenger flow that makes it easier for users to locate and reserve rides.

### 1.5.4 Iteration 4: Profiles and Bookings

In the fourth iteration, user management and bookings improvements were done.

- **User Profiles:** This feature allows the user to view ride history and update personal information.
- **Booking Refinement:** Improved user feedback and error handling in the booking process.
- **UI Improvements:** To improve usability and visual consistency, user interface elements were improved based on testing feedback.
- **Data Management:** Improved database functionality for user data synchronization and booking records.

This iteration improved the booking experience and user management features.

### 1.5.5 Iteration 5: Polish and Real-time Features

The final iteration focused on improving the application, adding real-time functionality, and ensuring system stability.

- **Real-time Location Updates:** During active rides, live location streaming of drivers and passengers was implemented.
- **Route Visualization:** Included dynamic route visualization that shows the locations of drivers and passengers on the map.
- **Error Handling:** Appropriate user feedback messages are used to improve error handling throughout the application.

- **Performance Optimization:** When applications are optimized, their responsiveness and performance are determined by testing results.

This iteration provided the last polished application with real-time capabilities and improved stability.

### 1.5.6 Testing and Refinement

Throughout the development process, continuous testing and refinement of the application was carried out in order to ensure the quality of the application and user satisfaction.

- **Continuous Testing:** Each iteration involved extensive testing of new functionality and regression testing of the existing functionality.
- **User Feedback:** Obtained feedback from target users (students and office workers) in order to spot usability problems.
- **Iterative Refinement:** Used feedback and testing insights to make changes to features and enhance user experience in future iterations.
- **Cross-platform Validation:** Tested the functionality of the application with different platforms to ensure consistency of functionality.

This iterative Agile methodology provided for the continual adaptation and improvement of the solution to ensure that Carpoolio developed into a robust and user-centric solution. Each iteration produced a working slice of functionality, allowing for constant testing and improvement in the development process.

# Chapter 2

# Literature Review

## 2.1 Background and Relevant Platforms

Carpooling and ride-sharing have attracted a large attention as sustainable transport solutions globally [4]. These initiatives have the primary objectives of reducing single-occupancy trips by vehicle, reducing carbon emissions, and supplementing public transit by improving vehicle occupant. Modern ride-sharing trends have a very important role to play in promoting sustainable mobility, although successful adoption of these trends is highly dependent on addressing user trust and convenience issues, as well as surmounting technological, economic, and regulatory barriers.

The global carpooling market is witnessing phenomenal growth and is expected to reach USD 10 billion in 2025 and USD 70–75 billion in 2035 at a growth rate of 20–22% compounded annually for this period [5]. This explosive growth is due to the increasing urbanization, rising cost of ownership of vehicles and a growing environmental consciousness among the commuters all over the world. Transportation is currently responsible for about 20–25% of global $CO_2$ emissions from fuel burning, and therefore carpooling is a key strategy for mitigating environmental impact and offers economic benefits to those who use it [6].

### 2.1.1 Carpoolio: Technological Innovation in Dynamic Matching

Carpoolio is a new vision of carpooling technology, designed as an academic effort and specifically targeting the technical limitations that have been seen in existing applications for carpooling in Pakistan. Unlike conventional platforms that are based on static and pre-planned matching systems, Carpoolio brings dynamic mid-ride pickup capabilities, where the passengers may join a ride already underway. This technical advancement solves the problem of spontaneous commuting patterns widely prevalent in Pakistani urban centers in which the traditional system of advance booking transportation does not fulfill the real-time transportation needs [7].

### 2.1.2 Ridemate: Operational Carpooling Platform of Pakistan

Ridemate is the currently active carpooling application performing as a basic ride-sharing application in the major urban centers of Pakistan. The platform uses conventional static matching systems in which users must specify their start locations, destinations and preferred travel times for advance booking. While providing true cost-sharing advantages compared to multi-passenger taxi providers, Ridemate's technological shortcomings preclude dynamic mid-ride taxi pickups that limit the service's capacity for serving spontaneous commuting demands.

The platform's performance in the market has shown low adoption rate despite the operational status, which is mostly due to the technological constraint, which does not cater to the need for flexible, real-time transportation options of the Pakistani commuters.

### 2.1.3 Savari: Market Misrepresentation Case Study

The case study of Savari helped to clarify the widespread usage of the various terms for the transportation app market in Pakistan. Savari says it provides carpooling; however, Savari is actually a multi-passenger taxi type service and users each pay for a full fare as opposed to sharing the cost amongst all passengers. When booking through Savari, the user can select the number of passengers they wish to ride with and the prices are always based on traditional ride-hailing pricing schemes. Therefore, Savari uses the carpooling terminology to market its service but does not offer these services.

Market research indicates customer base for the app, which can be attributed to the manner in which the app misleads users. The Savari case proves the need for implementation of real and legitimate carpooling as well as the establishment of true and clear cost-sharing methods in order to create confidence between users and create a stronger belief in, and adoption of carpooling in emerging markets [8].

### 2.1.4 DriveLu: Lessons Learned from Commercial Failure

DriveLu was a carpooling application that was available in the Pakistani market from 2019 to 2024 [9]. DriveLu failed primarily due to the time required to build up a substantial amount of users on the platform, as well as the dynamic matching capability which is essential for carpooling. Essentially, without the ability to find another drive near you with a simple button push the service would not provide the user the needed information in order be able to use the platform immediately.

DriveLu's failures gives a lesson into the sustainability issues that comes with dealing with carpooling platforms in Pakistan. It has shown that a simple carpooling app without dynamic matching is not enough to be viable in the market, especially because the existing ride hailing services have instant booking capabilities. This failure points to the need for

advanced technical innovation and sustainable business models in the implementation of carpooling in Pakistan.

### 2.1.5 inDrive: Ride Hailing Market Dominance

inDrive is a ride-hailing platform based on price negotiation that has achieved strong market presence in Pakistan and has become one of the dominant competitors in urban transportation [10]. Unlike carpooling, inDrive provides point-to-point individual rides. The success of this platform shows that Pakistani users prefer instant ride booking and flexible pricing instead of advance booking systems that are commonly used in traditional carpooling services.

Even though inDrive is not a carpooling platform, its strong market position helps to better understand the difficulties of adopting carpooling in Pakistan. inDrive's success in gaining a large market share shows the weaknesses of static carpooling models and points out the need for dynamic pricing and real-time matching features for carpooling systems to work successfully [11].

### 2.1.6 Yango: International Ride-Hailing Competition

Yango is a ride-hailing platform based in Dubai that has entered the Pakistani market and is slowly increasing its market share, which makes competition in urban transportation more strong. Similar to inDrive, Yango mainly focuses on individual ride-hailing services and does not offer cost-sharing or carpooling features. The expansion of Yango into Pakistan shows that international companies see the potential of Pakistan's urban transport market, while also highlighting the problems faced by local carpooling projects.

The launch of Yango in Pakistan shows the competitive advantage of global ride-hailing companies compared to traditional carpooling applications, especially because of better technology, stronger business models, and immediate service availability. This situation shows that technical innovation and service differentiation are very important for the successful implementation of carpooling systems in Pakistan.

## 2.2 Pakistan Transportation App Ecosystem Analysis

### 2.2.1 Market Evolution and Classification

Pakistan transport app system has changed a lot after 2019. It first started with commercial carpooling app called DriveLu, and later many other types of city transport apps came. The market can be put into three groups. First is real carpooling apps like Carpoolio, Ridemate and DriveLu. Second is apps that say they are carpooling but are not really carpooling, like

Savari. Third is big ride-hailing apps such as inDrive and Yango. This grouping helps to see how the transport market works in cities of Pakistan.

This change shows that people do not like fixed and static carpooling systems and they prefer fast and flexible transport options. DriveLu stopping its service and very few people using Ridemate is opposite to the strong success of ride-hailing apps. This shows that good technology and strong business models are needed for long-term urban transport solutions in Pakistan. [7].

### 2.2.2 Technical Architecture Comparison

Table 2.1: Comprehensive App Comparison

| **App** | **Service Type** | **Dynamic Matching** | **Cost Sharing** | **Market Performance** |
|---|---|---|---|---|
| Carpoolio | True Carpooling | ✓ | ✓ | Academic research solution |
| Ridemate | True Carpooling | × | ✓ | Limited market adoption |
| DriveLu | True Carpooling | × | ✓ | Discontinued (2019-2024) |
| Savari | Fake Carpooling | × | × | No active user base |
| inDrive | Ride-Hailing | × | × | Market leader |
| Yango | Ride-Hailing | × | × | Growing market share |

The technical study shows a clear connection between dynamic matching features and success in the market. Ride-hailing apps are more successful because they allow instant booking. Static carpooling systems face problems and do not get many users. Among carpooling apps, only Carpoolio tries to solve this technology problem by using dynamic matching systems based on academic research.

### 2.2.3 Technical Feature Analysis

Table 2.2: Technical Feature Comparison

| Feature | Carpoolio | Ridemate | DriveLu | Savari | inDrive |
|---|---|---|---|---|---|
| Real-time Location | ✓ | × | × | × | ✓ |
| Route Optimization | ✓ | × | × | × | × |
| Multi-passenger | ✓ | ✓ | ✓ | × | × |
| Price Negotiation | × | × | × | × | ✓ |

Study of technical features shows Carpoolio uses full approach for carpooling. It has real-time location tracking, route planning, and can take many passengers. Because of this, Carpoolio is only app that tries to do all important things needed for carpooling to work in Pakistan.

## 2.3 Market Failure Analysis

### 2.3.1 Shortcomings of Static Matching

DriveLu and Ridemate failed because static matching systems do not work well for carpooling in Pakistan. It does not fit with people who travel spontaneously in big cities of Pakistan. Because of this, these apps cannot compete with ride-hailing services that allow instant booking. Market research shows people in Pakistan need transport that is flexible and works in real-time. Static carpooling systems cannot give this. People cannot join rides that already started, so these apps are not very useful and few people use them. This shows dynamic matching is very important for carpooling to work in Pakistan.

### 2.3.2 Confusion Regarding Market Terminology

Savari says it is carpooling but it is not. This is big problem and makes people do not trust real carpooling apps. Savari uses carpooling words but really gives normal taxi service. People get confused about cost-sharing and real carpooling. Because of this, not many people use Savari even with marketing. This case shows that confusion in words can stop carpooling from working. It is very important to make clear and real carpooling to get trust from users. This failure shows Pakistan needs to separate carpooling and multi-passenger taxi services.

## 2.4 Identification of Research Gap

### 2.4.1 Technical Innovation Gap

Study of transport apps in Pakistan shows a tech innovation gap in the market. There is no carpooling app in the market that uses dynamic matching. While DriveLu provided the basic functionality of route-based matching for pre-planned journeys, DriveLu was unable to cater for the real-time ride requests during the active journey. Ride-hailing platform such as inDrive and Yango offer instant booking but without the cost sharing benefits, while Ridemate is still restricted to advance booking requirements. This technological gap is especially pronounced in the urban centers of Pakistan where commuting patterns are spontaneous and require flexible transportation solutions. Static and pre-planned systems such as DriveLu are no match to the instant availability offered by ride hailing platforms, and the real carpooling platforms such as Ridemate are limited by the need for advance booking. Carpoolio counters this lack by having a dynamic mid-ride pickup system, where passengers can join rides already underway and reserve parts of routes during active rides.

### 2.4.2 Market Education Gap

The Savari experience shows a large education gap in the markets on how to truly implement carpooling and about the cost-sharing benefits. Consumer confusion between true carpooling and multi-passenger taxi services hurts the adoption of real carpooling. This gap needs to be addressed through a transparent implementation and a clear market positioning.

## 2.5 Carpoolio Positioning and Innovation

### 2.5.1 Academic Advantage Framework

Carpoolio's academic context offers great benefits over commercial implementations, without the immediate pressure for profit and hence the ability to concentrate on technical innovation. This framework allows for the experimental implementation of dynamic matching abilities that provide the ability for passengers to join rides already underway. The academic approach focuses more on research and technical improvements, not on getting many users. This way allows new ideas that commercial apps cannot do.

### 2.5.2 Technical Superiority

Carpoolio uses dynamic mid-ride pickup, which is a big technical improvement for carpooling in Pakistan. It has real-time location tracking, route planning, and flexible matching.

These fix big problems that commercial apps have. Because of this, Carpoolio is only app that can give fast and spontaneous transport for Pakistani commuters.

### 2.5.3 Market Differentiation Strategy

Carpoolio is different because it gives real carpooling with clear cost-sharing. This stops confusion made by apps like Savari. Real carpooling benefits and good technical features make Carpoolio different from static carpooling and ride-hailing apps.

## 2.6 Conclusions

Study of transport apps in Pakistan shows important patterns. Static matching carpooling apps cannot get many users. Ride-hailing apps are strong because they have instant booking. Confusion in words makes people not trust real carpooling. Carpoolio fixes these problems with research, technical innovation, and real carpooling. Its dynamic matching is big achievement and helps people travel spontaneously, which other apps cannot do.

This research helps to understand why carpooling is hard in Pakistan. It shows how research and innovation can fix big gaps in commercial transport apps. The findings give useful ideas for future urban mobility solutions in Pakistan and similar countries.

# Chapter 3

# Requirement Specifications

## 3.1 Key Features

The main features of Carpoolio are aimed at creating a smooth carpooling experience for both drivers and passengers:

- User account management (sign-up, login, editing of their profile).
- Drivers are able to create and provide rides with Automated Fare calculation.
- Passengers can search and request available rides according to location and time.
- Booking management of confirmed rides.
- Showing user home screen of upcoming rides for both users.
- Real-time feedback and notifications through SnackBars.

## 3.2 Stakeholder Analysis

Effective development requires understanding the needs and expectations of all of the stakeholders involved with the Carpoolio application.

### 3.2.1 Primary Users

- **Students:** Daily commuters who want cheap and convenient way for intra-city travel.
- **Office Commuters:** Professionals who seek affordable and effective means of getting to and fro from work.

### 3.2.2 Drivers

People who have a willingness to give up some of their vehicle space in order to save on the cost of commutes or to earn additional income.

### 3.2.3 Project Team

The development team who is responsible for designing, implementing and testing the application.

### 3.2.4 Academic Supervisors/Mentors

Give guidance, feedback and assess the progress and outcome of the project.

## 3.3 Language

The Carpoolio application is currently developed for English (UK) only. While the design permits for localization in the future, all user interfaces, prompts and content within the application are in English.

## 3.4 Requirement Specifications

The system requirements are divided into functional, non-functional, software, and hardware requirements. This section describes the specific needs that Carpoolio has been designed to meet, in order to make its development clear and precise.

### 3.4.1 Functional Requirements

These requirements define the core functionalities that the system must perform.

Table 3.1: Functional Requirements: User Account Management

| Req. No. | Functional Requirements |
|---|---|
| FR01-01 | Users (passengers/drivers) can sign up with email and password. |
| FR01-02 | The system must verify the selected user type during sign-up. |
| FR01-03 | Users can log in with email and password. |
| FR01-04 | The system must verify the user's role (Passenger/Driver) upon successful login. |
| FR01-05 | Users can edit their profile (name, phone, email - view only). |
| FR01-06 | Drivers must be able to add/edit car details (model, seating, AC) during sign-up or profile update. |
| FR01-07 | Users can securely log out of the application. |

Table 3.2: Functional Requirements: Ride Management (Driver)

| **Req. No.** | **Functional Requirements** |
|---|---|
| FR02-01 | Drivers can create an offered ride by selecting start/destination on map. |
| FR02-02 | Drivers can view passenger details and contact information for their booked rides. |
| FR02-03 | The app must compute an estimated fare based on distance for offered rides. |
| FR02-04 | Drivers can view all their upcoming offered/booked rides. |
| FR02-05 | Drivers can mark a ride as "started" and "ended". |

Table 3.3: Functional Requirements: Ride Management (Passenger)

| **Req. No.** | **Functional Requirements** |
|---|---|
| FR03-01 | Passengers can specify start/end location and desired departure time for a ride request. |
| FR03-02 | The app must query existing driver-offered rides to find matches. |
| FR03-03 | Passengers can view a list of matching rides with details (driver, time, fare). |
| FR03-04 | Passengers can book a seat in a chosen matching ride. |
| FR03-05 | Passengers can view all their upcoming booked rides. |

Table 3.4: Functional Requirements: General Features

| **Req. No.** | **Functional Requirements** |
|---|---|
| FR04-01 | The app must display user-friendly success and error messages via SnackBars. |
| FR04-02 | The app must allow location selection via map taps or current GPS. |
| FR04-03 | The app must perform reverse geocoding to display human-readable addresses. |
| FR04-04 | The app must store and retrieve all user and ride data from Cloud Firestore. |

### 3.4.2 Non-Functional Requirements

These requirements are those that define the quality attributes of the system.

- **Cross-platform Support:** The application should be able to run flawlessly on Android, iOS, and desktop platforms, all from a single codebase of Flutter.
- **Usability:** The application must include intuitive navigation flow and user-friendly interface which should follow the principles of Material Design, presence of clear prompts and large interactive elements suitable for students and office workers.
- **Performance:** The app must work fast. Map must respond in real-time. Getting and updating data from Firebase must be quick.
- **Reliability and Scalability:** The app must use Firebase cloud backend. This will ensure data safety. Also, the app will be able to handle more users easily without needing many extra servers.
- **Security:** User authentication is required to be handled securely using Firebase Auth, which is implemented with basic input validation to avoid common vulnerabilities.
- **Maintainability:** The codebase needs to be split into logical directories including screens for full pages, models for data structures, and components for reusable UI elements. Configuration and theming need to be split for easier maintenance.

### 3.4.3 Software Requirements

This section describes some of the most important software components and tools needed for Carpoolio's development and operation.

- **Frontend Development:** The SDK (Dart) of Flutter for developing the cross-platform mobile application.
- **Backend Services:**
  - Firebase Authentication for user registration and login.
  - NoSQL document database storage of all the application data (user profiles, ride offers, bookings) - Cloud Firestore.
  - Google Maps Platform API to display maps and use location services.
  - Google Geocoding API for converting GPS coordinates to addresses and vice versa. Location search and autocomplete Google Places API.
  - Google Directions API for route calculation and navigation.
  - HTTP package for using Google services via REST API.

- **Development Environment:**
  - Visual Studio Code (IDE).
  - Flutter extensions for VS Code.
  - Firebase CLI to setup and configure the project.
- **Third-Party Libraries/Plugins:** Essential Flutter packages such as `google_maps_flutter`, `geolocator`, `geocoding`, `firebase_auth`, `cloud_firestore`, and `intl` (for date formatting) are required.

subsectionHardware Requirements
This section describes the recommended hardware specifications at which the application should be developed, tested, and deployed.

- **Development Machine:**
  - Processor: Intel Core i5 (or equivalent AMD) Minimum, Intel Core i7 (or equivalent) Recommended.
  - RAM: 8 GB Minimum; 16 GB or more is recommended for smoother development.
  - Storage: At least 256 GB of free storage space for SSD (SDKs, emulators, and project files).
- **Testing Devices:**
  - Android device running Android 8 (Oreo) and above.
  - iOS device with iOS 12 or higher.
  - Desktop environment (Windows, macOS, Linux) for desktop target testing.

### 3.4.4 Use Cases

This section outlines major user interactions with the Carpoolio application using specific use case scenarios, as an example of how users achieve their goals. Each use case describes the major flow of events and user-system interactions.

The following diagram gives an idea of the high-level use cases of the Carpoolio system and their interactions with the main actors (Passenger, Driver) and external systems (Firebase System).

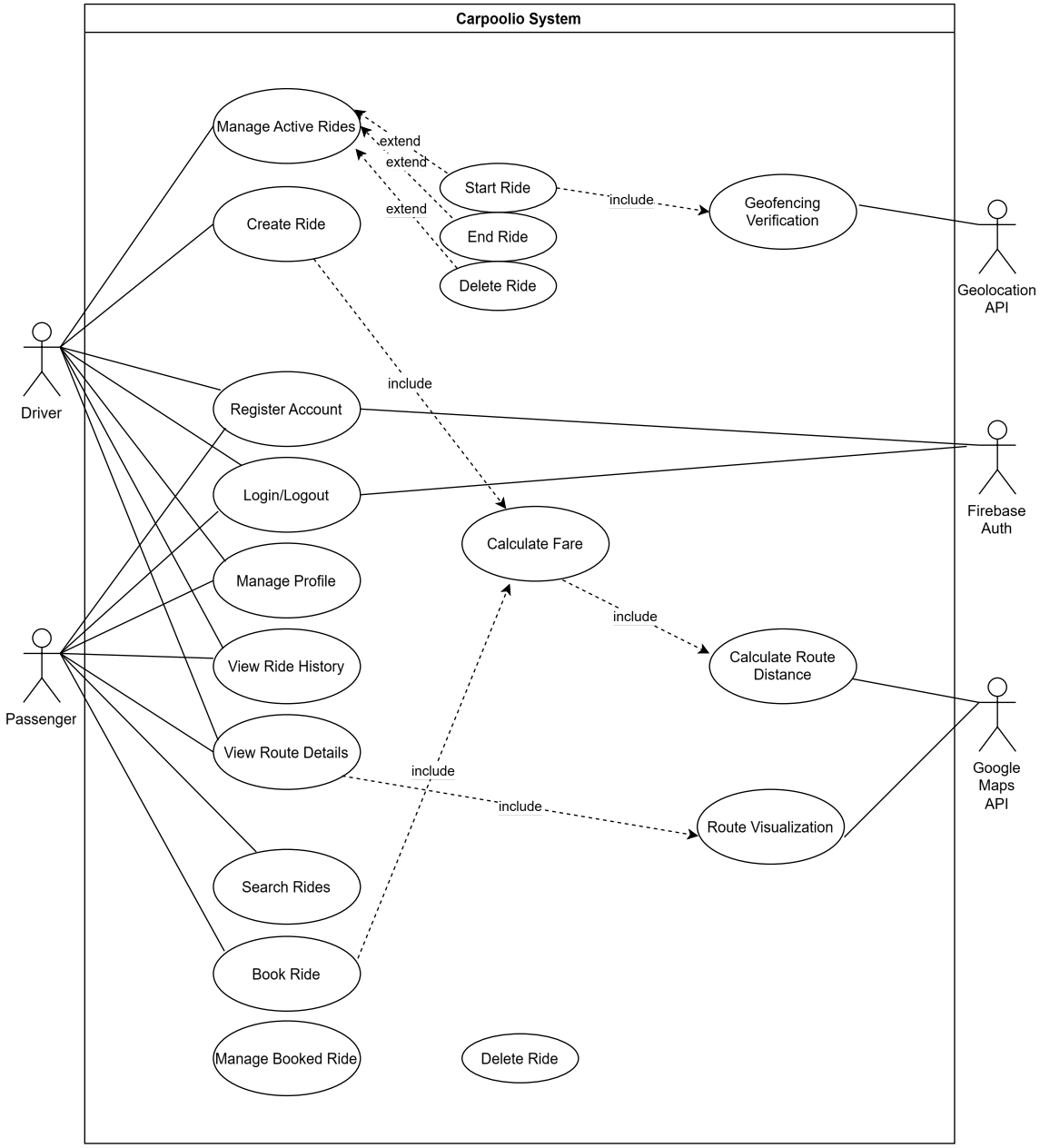


Figure 3.1: Use Case diagram of the system.

### 3.4.5 Use Case 1: Register/Login

Table 3.5: Use Case Specification: Register/Login

| | |
|---|---|
| **Use Case ID** | UC-001 |
| **Use Case Name** | Register/Login |
| **Actors** | Passenger, Driver |
| **Key Data Involved** | User Email, Password, User Type, Profile Details (Name, Phone, Car Details) |
| **Trigger** | App launch or selection of "Sign Up" / "Log In". |
| **Pre-conditions** | Internet connectivity. |
| **Description** | **Normal Flow:** User signs up (email, pass, profile) or logs in (email, pass). System authenticates via Firebase, verifies role, and navigates to main app based on role. **Alternate Flows:** Invalid credentials, registration failure, or no internet cause error messages. |
| **Post-conditions** | User authenticated, active session established. Profile data stored. |
| **Comments** | Covers registration and login. Firebase Auth manages security. |

### 3.4.6 Use Case 2: Driver Adds a Ride

Table 3.6: Use Case Specification: Driver Adds a Ride

| | |
|---|---|
| **Use Case ID** | UC-002 |
| **Use Case Name** | Driver Adds a Ride |
| **Actors** | Driver |
| **Key Data Involved** | Start Location, Destination, Departure Date/Time, Estimated Fare, Driver/Vehicle Details |
| **Trigger** | Driver selects "Add a Ride" tab. |
| **Pre-conditions** | Driver logged in, car details updated, internet connectivity. |
| **Assumptions** | Google Maps API and geocoding services are operational. |
| **Description/Steps** | **Normal Flow:** Driver navigates to "Add a Ride." Selects start/destination on map or by address. System calculates estimated fare. Driver specifies departure date/time. Driver taps "Add Ride." System creates new ride offer in Firestore. Confirmation displayed. **Alternate Flows:** Incomplete details, location permission denial, Firestore write failure, or no internet connection cause errors. |
| **Main Success Scenario** | Driver successfully publishes a new carpool ride offer, saved to database. |
| **Post-conditions** | A new ride document exists in "enlistedrides" collection in Firestore. Driver's ride list is updated. |
| **Comments** | Fare calculation is based on a predefined formula. |

### 3.4.7 Use Case 3: Passenger Searches and Books a Ride

Table 3.7: Use Case Specification: Passenger Searches and Books a Ride

| | |
|---|---|
| **Use Case ID** | UC-003 |
| **Use Case Name** | Passenger Searches and Books a Ride |
| **Actors** | Passenger |
| **Key Data Involved** | Passenger's Start/Destination, Desired Departure Time, Matching Ride Details (Driver, Route, Time, Fare, Seats) |
| **Trigger** | Passenger selects "Find a Ride" tab. |
| **Pre-conditions** | Passenger logged in, internet connectivity. |
| **Assumptions** | Available rides exist matching criteria; matching algorithm is accurate. |
| **Description/Steps** | **Normal Flow:** Passenger navigates to "Find a Ride." Selects start/end locations and desired departure time. Taps "Find Matching Rides." System queries and filters matching rides from Firestore. Displays list. Passenger selects a ride and taps "Book." New booking document created in Firestore. Confirmation displayed.<br>**Alternate Flows:** No matching rides, booking conflicts (ride unavailable), Firestore write failure, or no internet connection cause errors. |
| **Main Success Scenario** | Passenger successfully finds and books a carpool ride, and the booking is recorded in the system. |
| **Post-conditions** | A new booking document exists in "bookedride" collection. Passenger's ride list is updated. |
| **Comments** | Matching algorithm uses predefined location and time thresholds. |

### 3.4.8 Use Case 4: User Manages Profile Information

Table 3.8: Use Case Specification: User Manages Profile Information

| | |
|---|---|
| **Use Case ID** | UC-004 |
| **Use Case Name** | User Manages Profile Information |
| **Actors** | Passenger, Driver |
| **Key Data Involved** | User Name, Phone Number, Car Details (for Driver), Email Address |
| **Trigger** | User selects the "Profile" tab. |
| **Pre-conditions** | User logged in, internet connectivity. |
| **Assumptions** | User provides valid input formats. Firestore is accessible for profile data. |
| **Description/Steps** | **Normal Flow:** User navigates to "Profile." System displays current profile info (Name, Phone, Email; car details for drivers). User edits editable fields (Name, Phone, Car details). Taps "Update Profile." System sends updated info to Firestore. Confirmation displayed. Updated info reflected. **Alternate Flows:** Invalid input format, Firestore write failure, or no internet connection cause errors. |
| **Main Success Scenario** | User successfully updates their profile information, and changes are persisted in the database. |
| **Post-conditions** | The user's profile document in Firestore is updated with new information. |
| **Comments** | Email is displayed as read-only for security and account integrity. |

# Chapter 4

# System Design

This chapter describes in great detail, the architectural and component level design of the Carpoolio application. It goes further than just high-level requirements and outlines the actual technical blueprints for building it, including its basic structure, the technologies to be used, and the complexity of the interaction among different parts of the system that are interconnected. This section gives a basic blueprint for the system's solid and efficient implementation, so that the software development process is driven with a clear and well- defined technical vision. The design herein has prioritized several important non- functional requirements such as exceptional scalability to support a rapidly increasing user base and rising demand for the number of rides, stringent security measures to secure sensitive user data and transaction integrity, ease of maintainability to support further enhancements and bug fixes, and an intuitive user experience that is the core of the entire platform. All these points are for Carpoolio to become a reliable, fast, and easy-to-use carpooling app.

## 4.1 System Architecture

The system architecture carefully outlines the underlying architecture of the Carpoolio application and shows how its many software components and infrastructure elements are structured and how they work in concert with one another to provide the necessary functionalities. Our design has a conscious use of modern, cloud-native approach, fundamental use of the distributed architecture. This strategic choice is based on the imperative to ensure inherent scalability - for the system to seamlessly expand its capacity in response to fluctuating load, uncompromising reliability through redundancy and fault tolerance, and highly efficient data management through all operations. This section will give a high-level overview in a very detailed way, breaking down the main architectural layers which make up the Carpoolio system, as well as an introduction to the most important technologies and services that are strategically used within each layer.

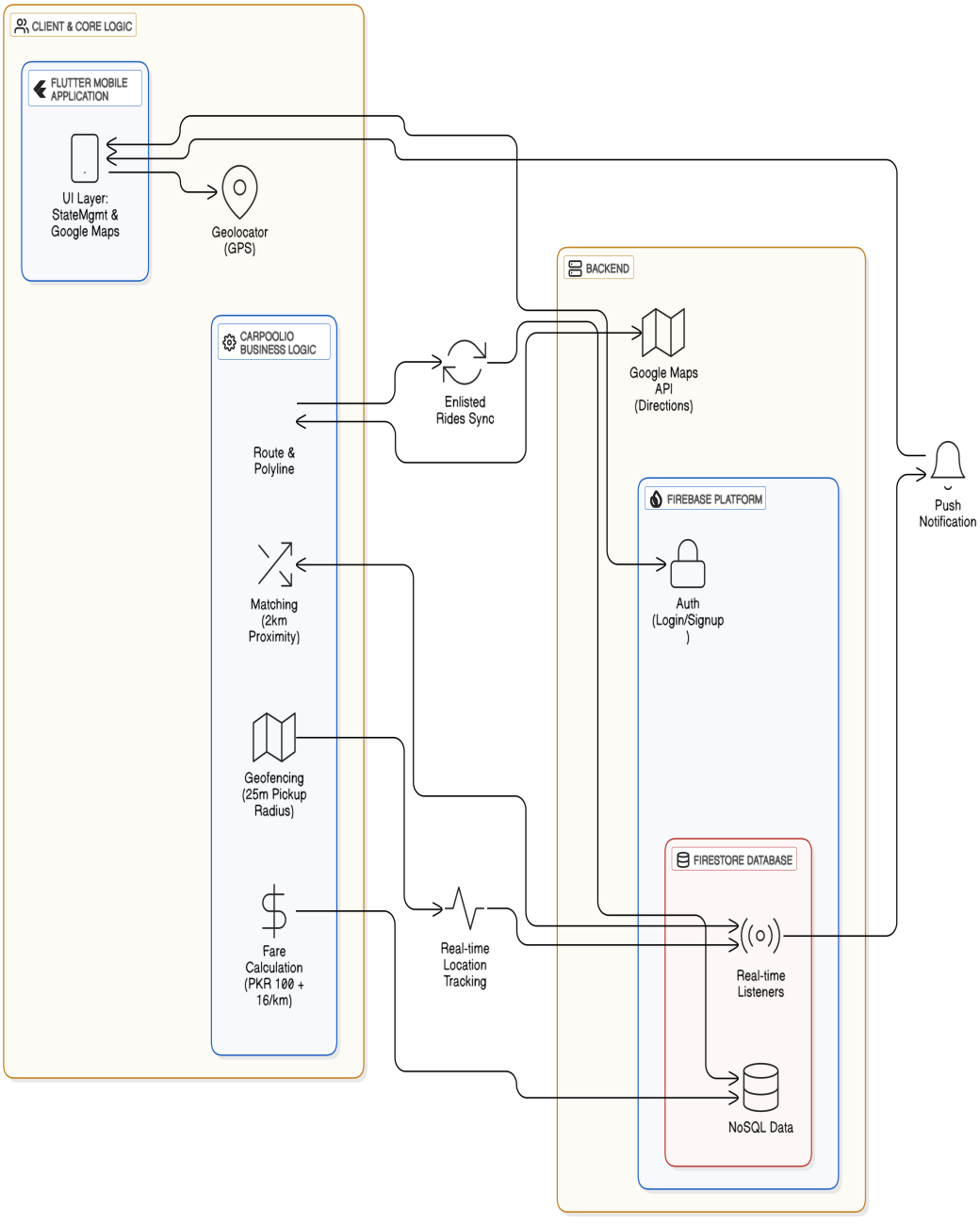


Figure 4.1: Carpoolio System Architecture Overview

### 4.1.1 Architectural Layers

Carpoolio system is made with separate layers. This makes it easier to change, fix, and keep parts separate. This layered approach guarantees the ability of different parts of the system to evolve separately, yet still have strong interfaces. These layers are meant to tackle specific responsibilities and help make the overall stability and efficiency of the application..

#### 4.1.1.1 Presentation Layer (Client-side)

This layer represents the user-facing aspect of the Carpoolio application as well, mainly consisting of the mobile application that has been developed for both Android and iOS platforms. Its primary duty is to control the user interface (UI) and user experience (UX) of the system via which the users can interact with it in an intuitive manner. It is responsible for all user inputs, rendering dynamic information fetched from the backend and is responsible for initiating communication with the application's backend services via secure API calls. No business logic or data storage is done directly at this layer and it is purely the interaction point.

#### 4.1.1.2 Application/Backend Layer

The Application Layer is mostly implemented in the Flutter client application as Carpoolio utilizes the managed services of the Firebase product (rather than a custom backend). Business logic like ride matching, booking validations, and data transformations are done using the client-side using Firebase rules for data validation.

#### 4.1.1.3 Data Layer (Database and Storage)

Dedicated to the persistent storage and the efficient retrieval of all application data, Data Layer is one of the critical parts of the application. It makes use of Cloud Firestore, a NoSQL document database, which has been selected for its flexibility and scalability, which is ideal for dynamic data models such as user profiles, ride offers and booking details. Temporary location data is stored for active rides for real-time tracking. This layer is responsible for ensuring the durability and consistency of data and also for providing high- performance access to data for the Flutter client applications.

#### 4.1.1.4 External Service Integration Layer

This layer is responsible for managing and orchestrating all of the interactions with the third party services which are critical for Carpoolio's full functionality. Key Integrations include Mapping and Location services (Google Maps API for real-time navigation, geocoding

and route calculation), and authentication providers (firebase authentication for secure user sign-up, login and session management). This layer abstracts the complexities of the external APIs and provides a unified and secure interface for the Application/Backend Layer.

## 4.2 User Flow and System Structure

This section gives an overview of the Carpoolio application's user experience, along with its basic structural elements, and is the basis for the application design. It describes in detail the common user path from the application launching time till its basic functionalities and covers comprehensively the different states and transitions that a user may go through. This visual representation is important to understand the design idea of the system. It shows how different parts of the system connect before looking at smaller details. The goal is to make the app easy to use for all users, for both ride-sourcing and ride-sharing.

The architectural design of the application is carefully focused on a very clear and intuitive navigation path, which ensures a seamless and logical path for all users. Upon opening the application of Carpoolio, users are immediately led to a first check of authentication. This is a very important preliminary step that efficiently determines their current login status. If a user has already been authenticated and has an active session, he/she passes through the first login/signup screens easily and gets directed to the Main App Screen. This screen has the dynamic role of central and personalized hub, which by intelligent mechanisms adapts its interface and available functionalities depending on the type of user who is registered, in this case a Passenger or a Driver. This is an adaptive design that ensures that users are presented with the most relevant options for their role immediately.

On the other hand, if a user is not logged in or has an expired session, he or she is intuitively led to the Landing Screen. At this point, users are provided with certain options. They have an option to proceed with existing account, or begin with creating a new Carpoolio account. Upon either logging in, the user is taken to his or her Main App Screen.

Main App Screen has numerous tabs that are readily accessible. These tabs are: Home tab, where the user can see overview of his/her role; Find a Ride tab, where Passengers can search and book rides; Add a ride tab, where Drivers are able to post new rides and Profile tab, where both types of users can manage personal information.

### 4.2.1 Driver User Flow

First, the driver logs in. They can set up a ride or manage the ones they already have. They pick where they are going and wait for people to book. Once passengers are found, they

can pick them up and start the ride, and after the ride is completed, they can end the ride.

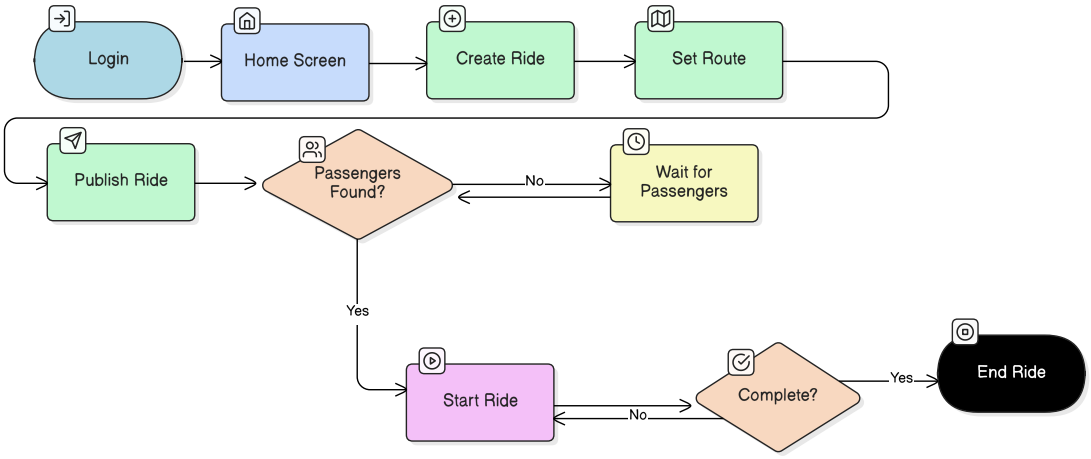


Figure 4.2: Driver User Flow Diagram

### 4.2.2 Passenger User Flow

First, the passenger signs in. They search for a ride and book it. The app shows on the app the exact location of the driver. After the ride is started, they can visually see their trip on the map as well.

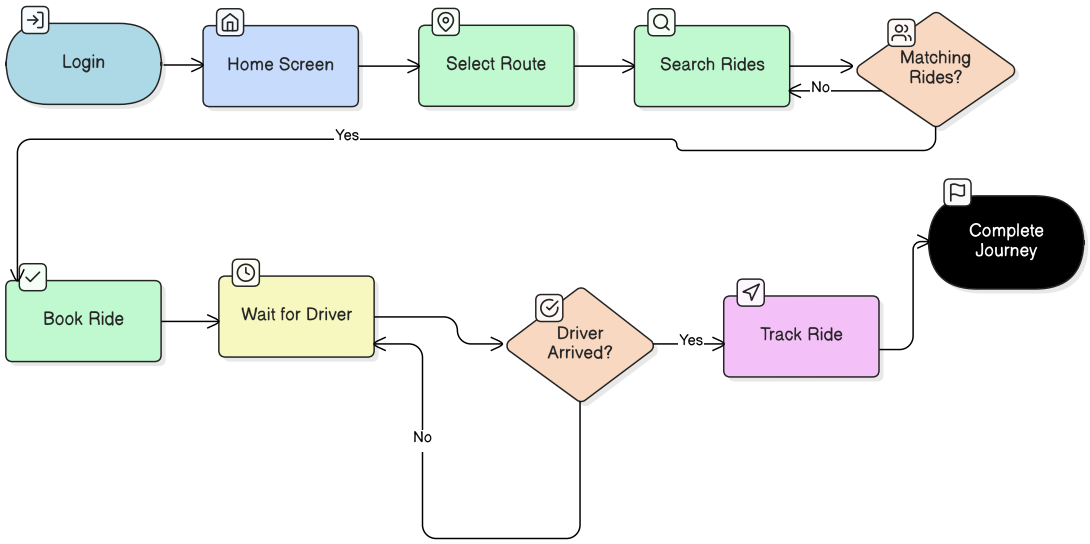


Figure 4.3: Passenger User Flow Diagram

## 4.3 Detailed Component Interaction Design

Following the high-level user flow, this section goes into the detailed step by step interactions that take place between the main components of the Carpoolio application. Using the sequence diagrams, it shows the chronological sequence of messages/actions exchanged between the User, the Carpoolio Application client, the Authentication Service (Firebase Authentication), and the Database (Cloud Firestore or other backend services) for critical functionalities. This detailed perspective is necessary for understanding the underlying logic, data flow and responsibilities of each system component in executing core user operations.

### 4.3.1 User Authentication and Registration Process Flows

This part explains how Carpoolio checks users. It covers both first-time sign-up and login. It shows how the app interface, Flutter app logic, Firebase Authentication, and Cloud Firestore work together to save and manage user profiles.

The sign up starts from taking input from the user about their email, password and basic profile details. After validating the credentials on the client side, these are then securely

sent to the Firebase Authentication in order to create the user account. Concurrently, additional profile information (e.g. name, phone, user type) is stored in Cloud Firestore, and is linked by the Firebase generated user ID. Successful registration results in automatic login and navigation to the appropriate main app screen with appropriate error messages for any failures.

For the returning users, the step of the login procedure will be entering the credentials through the UI. These are verified on the client-side, after which it is sent to the Firebase Authentication for verification. On successful authentication, the application retrieves the comprehensive profile information of the user from Cloud Firestore to personalize the experience. Successful login takes the user to the Main App Screen and invalid credentials will receive appropriate error messages. This detailed flow has the benefit of providing secure and efficient user access to the functionality of Carpoolio.

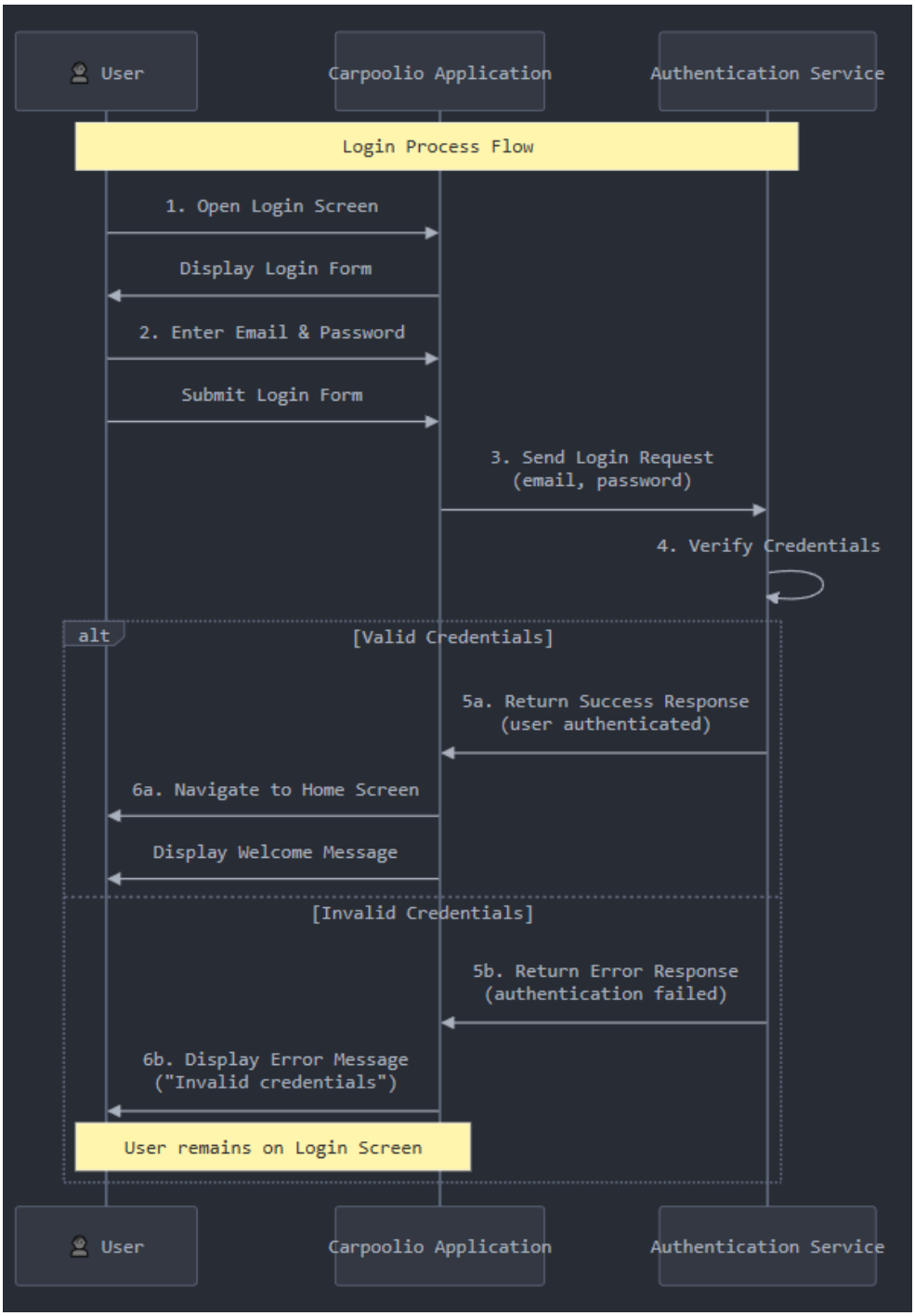


Figure 4.4: Login Process Sequence Diagram

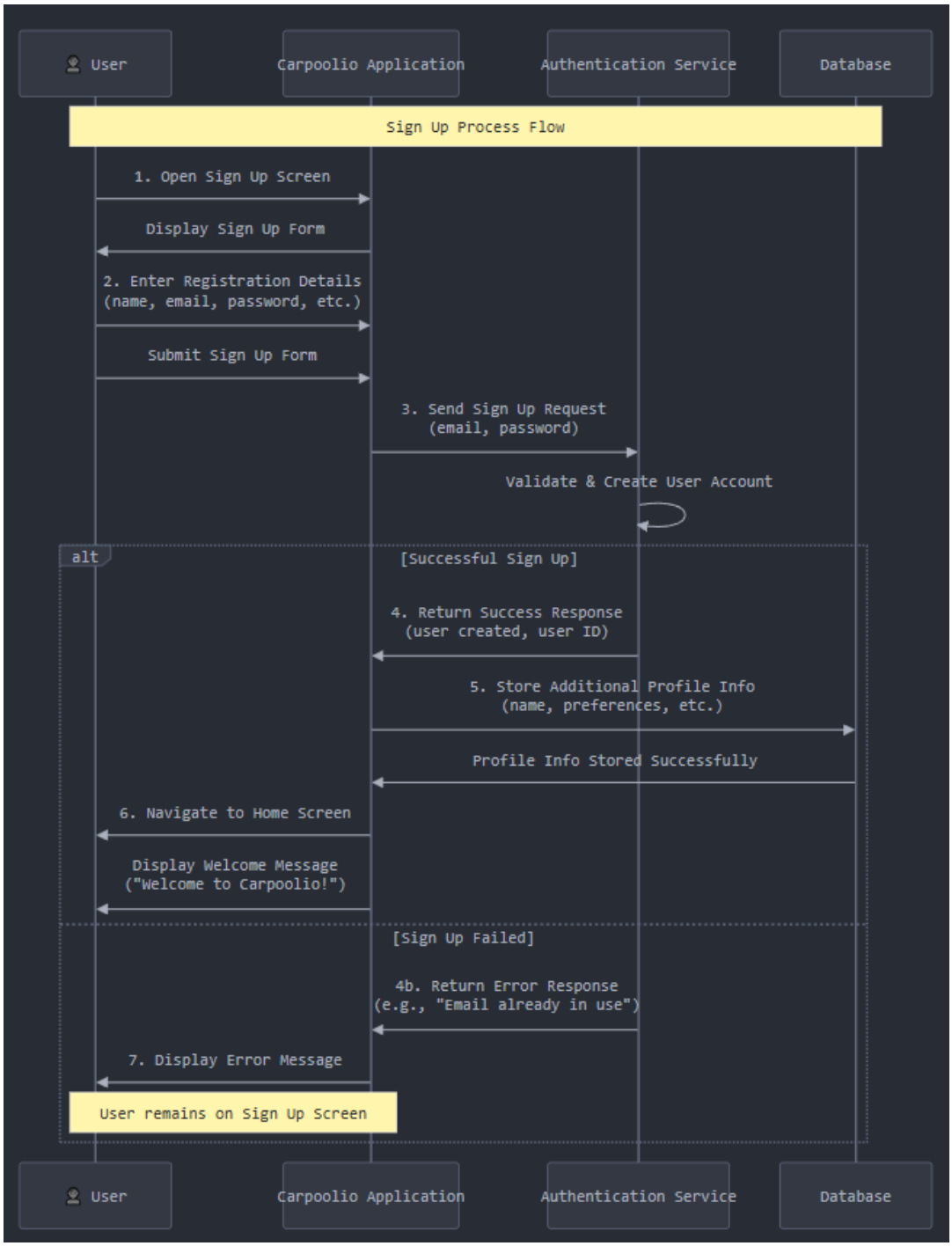


Figure 4.5: Sign Up Process Sequence Diagram

### 4.3.2 Ride Request and Processing Flow

This subsection shows the detailed sequence of events that occur when a passenger makes a ride request in the Carpoolio application. It describes the interactions from user's input

to application's validation, query generation and storing in the database and ending in the presentation of matching rides.

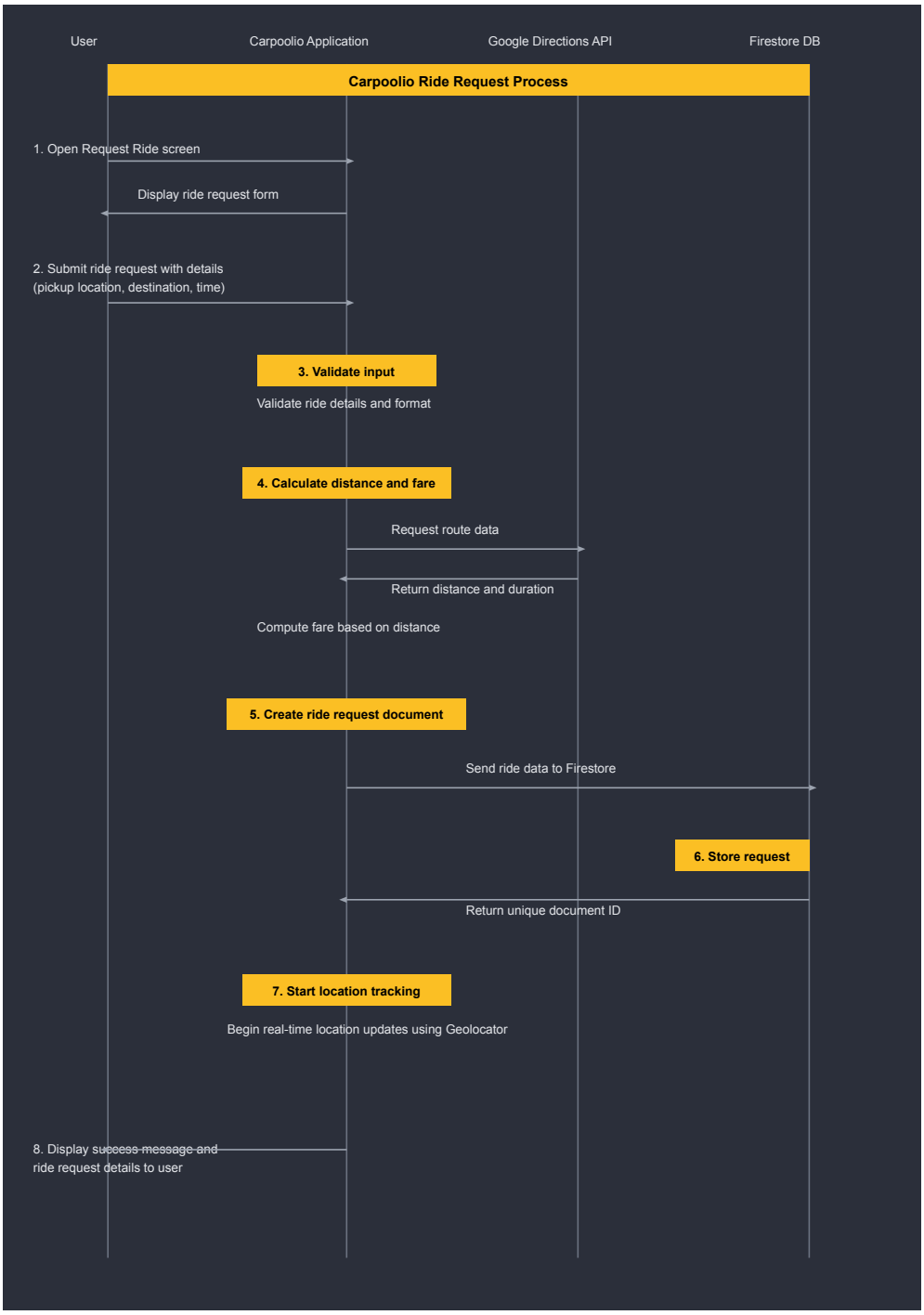


Figure 4.6: Carpoolio Ride Request Process Sequence Diagram

## 4.4 Design Constraints

When building Carpoolio a lot of the design and technical decisions come from certain limits we have to follow. These limits could be things like rules we must obey, outside restriction or conditions we cannot change. They constrain the possible ways we can build the system and sometimes constrain us to choose certain technical or operational ways of building it. It is really important to know these limits and plan for them carefully. It helps to ensure that the app will actually work, and remain reliable and successful in real use, as well as balancing what we want the app to do and what is realistically possible.

## 4.5 Technical Constraints

Technical constraints are those constraints arising from the technology we use, the current infrastructure, or the way our tools work. These are important since they directly influence the way the system can be built, and how it will perform.

### 4.5.1 Cross-Platform Development Requirement

One of the main limitations is the requirement to implement the application in both Android and iOS mobile operating systems within a tight development schedule. This limitation greatly limits the selection of frontend frameworks, effectively forcing a single cross-platform framework (for example, Flutter) to be used to the maximum extent to maximize this sharing of code and development efficiency, so that there is no need for separate native development for each platform.

### 4.5.2 Third Party API Usage Restrictions

Reliance on external mapping and location APIs (e.g., Google Maps Platform, Google Places API) will introduce limitations concerning API rate limits and usage quotas and the associated costs. The system design needs to consider these external restrictions by implementing such strategies as intelligent caching, optimized query patterns, and client-side data processing so as to reduce API calls and control budgetary overhead that would result in an interruption to a service or an unexpected expense.

### 4.5.3 Cloud Provider Ecosystem Lock-in

Selecting Google Cloud Platform and Firebase as backend services such as Authentication and Firestore has a drawback that is vendor lock-in. This implies that we cannot easily switch to a different provider in future, but instead go by the data models and methods of operation prescribed by Google so that we achieve positive performance and ensure that costs remain manageable.

### 4.5.4 NoSQL Database Model of Operation

There are limitations to using document database like Cloud Firestore that is based on NoSQL. There is no native support of complex relational queries and certain aggregate functions and atomic operations across collections are difficult due to the distributed nature of NoSQL. This implies that we must structure the data thoughtfully, frequently denormalizing it and making it scale and speed-optimized rather than relying on conventional relational mechanisms.

## 4.6 Operational Constraints

Operational limitations are associated with the way the system is operated and maintained on a daily basis.

### 4.6.1 Limitations of Client-Side Processing

The app relies on the processing power and battery of the device since all sales logic is made by the client. Constant tracking of location and expensive computations may deplete battery or slow down the application in low-end phones. Therefore, the route calculations and real time updates should be well adjusted to prevent performance issues.

### 4.6.2 Limited Monitoring Resources Dedicated

We do not have a complete team to oversee the system around the clock. This implies that the system must have excellent automated logging, intelligent notices, and self-healing facilities where applicable. In this manner, even a small team would be able to react to critical problems and maintain the system in good health without supervision at all times.

## 4.7 Constraints Related to Security and Compliance

These are the rules concerning security, regulations, and protection of user data that cannot be negotiable.

### 4.7.1 Regulatory Requirements for Data Privacy

The laws such as GDPR and local privacy regulations ensure that user personal data and real-time location cannot be obtained without a specific reason. This restricts the ability to gather, store and process the data. It also implies that we should have good encryption, good permissions, and explicit consent system to remain within the law.

### 4.7.2 Industry Standard Authentication Protocols

Authentication and authorization should be based on the accepted standards such as the OAuth 2.0. This does not allow us to use our custom procedures, which may be unsafe, and makes sure the system is compatible with known security procedures.

## 4.8 Resource Constraints

These are constraints on the individuals, time, and money to be used on the project, which influences scope and execution.

### 4.8.1 Fixed Development Timeframe

The first release has a very tight deadline in the project. This constrains the number of features we can add, meaning that we need to select the most important features required in the MVP and postpone the rest which are not critical. Agile practices can be used to cope with this.

### 4.8.2 Defined Budgetary Ceiling

We have a set budget. This influences decisions such as cloud services, open source and proprietary tools and external APIs. The system must be economical in terms of development and usage.

### 4.8.3 Limited Team Expertise in Novel Technologies

The team is powerful in terms of core development but we lack an in-depth knowledge of certain new or very specialized technologies. This implies that we select the easier to learn tools and platforms, those that are well documented and those with good community backing in order to proceed with development without any difficulties.

## 4.9 Design Methodology

Carpoolio is written in a contemporary component driven implementation that is founded on Object-Oriented Programming (OOP). This is quite suitable to our Flutter and Firebase stack, which allows us to maintain the code in a modular, reusable and easy-to-maintain manner. This section identifies the main principles and architecture patterns that we are using in the design.

### 4.9.1 Object-Oriented Programming (OOP) Paradigm

The whole Carpoolio application codebase is organized around the fundamental tenets of OOP: the core principles for organizing code structure, managing complexity, and collaboration.

#### 4.9.1.1 Encapsulation and Abstracting

Every important functional unit inside the application is encapsulated in dedicated classes and objects. For example, user interface elements (screens, widgets), data structures (models), and UI screens are all defined as separate classes. This imposes a clear separation of concerns so that details of an internal implementation are hidden (encapsulated) from the outside world, and interaction is only through well-defined public interfaces (abstraction). This greatly alleviates coupling and eases reasoning about individual parts of the system.

#### 4.9.1.2 Inheritance and Reusability

Flutter's architecture is widget-based and therefore encourages inheritance. All UI components inherit from basic classes such as `StatelessWidget` or `StatefulWidget` and thus inherit basic properties and behaviors. This mechanism is used extensively to establish a hierarchy of widgets so that consistent styling, layout, and interaction patterns can be used throughout the application, while achieving maximum code reuse.

#### 4.9.1.3 Polymorphism

Carpoolio polymorphism largely depends on the Flutter system of widgets whereby various types of widgets could be used interchangeably in the assembly of the UI. As an example, the generic type, i.e., Generic Widget can model various specific UI elements, i.e., a button, text field, custom component, etc. of different renderings based on context or data state. This enables the development of dynamic UIs which show variation of the widgets but with the same interfaces.

### 4.9.2 Component Based Design (Flutter's Paradigm)

In addition to general OOP, the very component-driven architecture is required by Flutter due to its unique architecture, which is a combination of widgets, and that is the secret of the development of the frontend of Carpoolio.

#### 4.9.2.1 Everything is a Widget

Every visual component of the application, a simple text label up to a entire screen, is a `Widget` in Flutter language. This atomistic approach promotes the disaggregation of

complex UIs into small and manageable components which are also reusable. The layout of each widget is a separate piece of logic, with (in the case of thrustable) changing state, which encourages a separation of concern in the UI layer. This promotes the rapid generation of UIs and visual identity consistency.

#### 4.9.2.2 Declarative UI

we used Flutter in terms of the UI since it is declative. This means that the screen is actually merely a reflection of the state of the app rather than us making everything change manually. Flutter simply re-creates the layers that require the state to change. This ensures that the UI is constantly equal to the data and we avoid numerous bugs where the screen does not update correctly.

### 4.9.3 Event-Driven and Service Oriented Design

To achieve a responsive and smooth app, the app is based on event-driven and service-oriented design.

#### 4.9.3.1 Event-Driven Interactions

The application is responsive to user activity as well as system changes. Events are triggered by actions such as button clicks or form submissions, and they are processed by the respective methods or controllers. Likewise, backend updates like live Firestore transactions can also be produced, which can be responded to in real time by the app.

#### 4.9.3.2 Direct Backend Integration

The code for the backend is right inside the screens. We do not use separate files to handle the data. We just use Firebase directly on each screen. It is faster and easier to build this way. The application is less complicated this way, but it means the screens and the database are stuck together.

### 4.9.4 State Management Strategy

The state management is a component of the design approach, where the process of managing changes in data in relation to their propagation across the user interface is defined. In this project, the native Flutter tools (local state), and Firebase Authentication (global user session state) are the primary tools, which are both simple and easy to integrate with the framework.

#### 4.9.4.1 Local State Management (StatefulWidget, setState)

For handling state in individual screens and UI components, the application largely uses Flutter's `StatefulWidget` and the accompanying `setState` method. This approach deals with localized state requirements (form input values, loading indicators, temporary UI changes). By encapsulating state within the `State` object of a `StatefulWidget`, rebuilds are precisely targeted, minimizing unnecessary UI updates and maintaining performance. This provides a clear, simple, and reactive way to reflect changes in the user interface.

#### 4.9.4.2 Global State Management (Firebase Authentication Integration)

To ensure critical global state especially user authentication and session management, the project makes use of Firebase Authentication, which belongs to the Google Cloud package. The application replaces custom global state mechanism with the use of `FirebaseAuth.instance.currentUser` to get the current authentication status. This maintains that the knowledge (user login status) of the application is always updated and synchronized with the backend. The reactive behavior of Firebase streams enables various components of the application to respond to changes of login/logout/session automatically, without explicit management of user-dependent user interfaces flows and user access control [2].

#### 4.9.4.3 Benefits of this Approach

This totalistic style of state management has a number of advantages. It makes the state paradigm simple, fits the reactive UI paradigm of Flutter, lowers the number of external dependencies, and does not introduce complexities such as third party state management libraries. This gives a light, thin, and serviceable codebase, which improves long term stability and productivity of the developers at a high rate.

## 4.10 Scalability, Maintainability, & Testability

In addition to basic paradigms, the design methodology is concerned with real-life issues regarding the sustainability of the application.

### 4.10.1 Code Cleanliness and Documentation

There is a focus on the readability, maintainability as well as good documentation of the code. It is obligatory to follow Dart linting conventions, the use of clear naming conventions, and inline documents (doc comments). This enhances maintainability, eases when hiring new developers and minimizes the risk of adding bugs to future enhancements.

### 4.10.2 Layered Architecture (Separation of Concerns)

The application is modeled in such a way that it has been structured as a logical layered architecture where various responsibilities are allocated to distinct layers. Such separation of concerns guarantees that the elements in one layer are loosely along with the elements in other layers and contributes to modularity, testability, and general understanding of the system.

### 4.10.3 Focus on Testability

The selected design patterns (OOP, layered architecture) encourage testability. The app has gone through with a lot of manual testing through different critical user flows and business cases such as the fare calculation, ride matching algorithms, and user authentication flows. Elements of the interface were examined on other devices and screens. This type of manual testing is very strong and reliable. The modular design is also useful in future automated unit and component testing as required.

## 4.11 Database Design

The database design for the Carpoolio application is critical in terms of the efficient storage, organizing and retrieval of all transactional and master data. Given the application's dependency on the NoSQL document database, Firebase, Cloud Firestore, has been chosen as the main data persistence layer. This selection offers inherent scalability, flexibility and real-time synchronization capabilities, which is a good fit for the dynamic nature of a carpooling platform. The design spends most of its attention in creating a flexible schema, optimized for read performance, and organized to provide support to the application's core functionalities while ensuring the integrity of the data.

### 4.11.1 Data Model

The conceptual data model provides the essential entities and their relationships in the Carpoolio system and is a high-level blueprint to design the data. This model is shown in the Entity-Relationship Diagram (ERD) given below. Each entity in the ERD represents a top-level collection in Cloud Firestore, and individual records are represented in the form of documents within these collections.

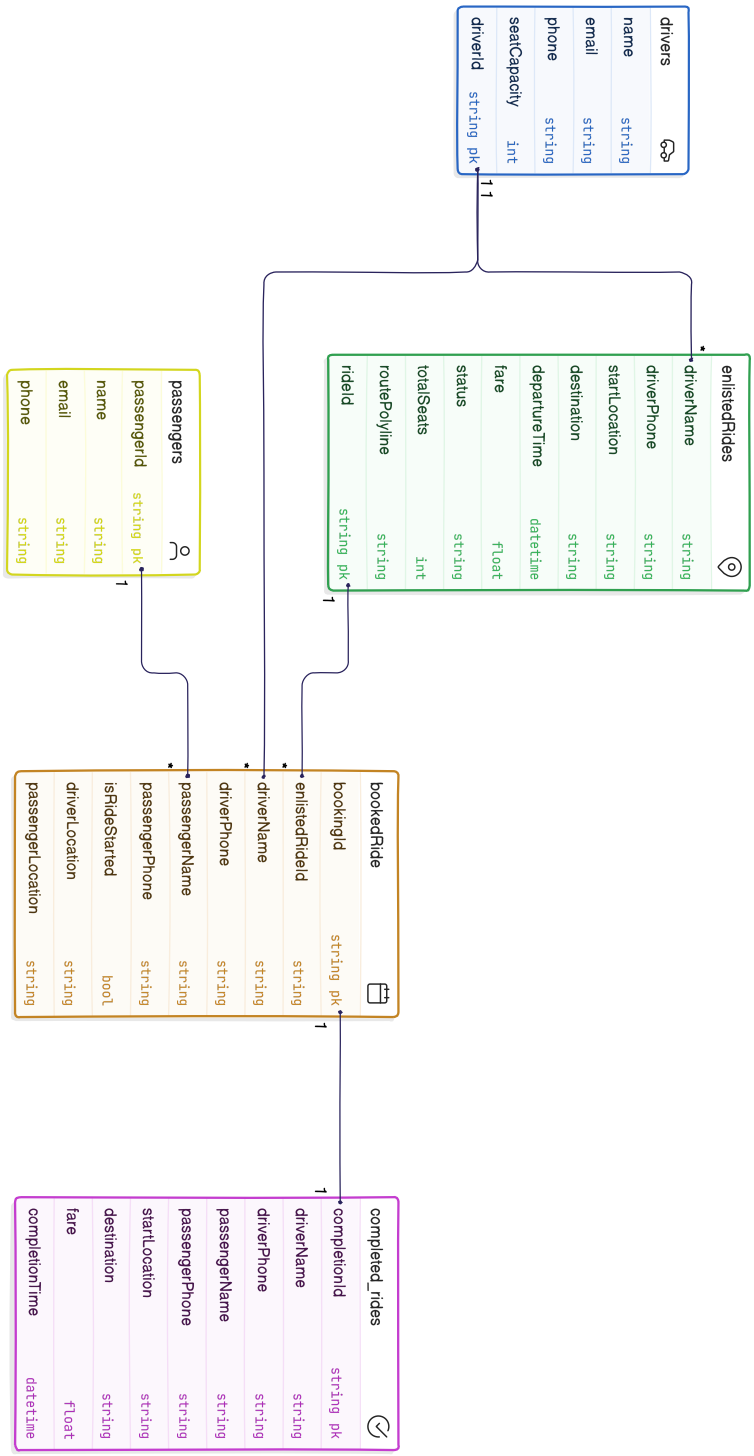


Figure 4.7: Carpoolio Entity-Relationship Diagram (ERD)

## 4.12 GUI Design

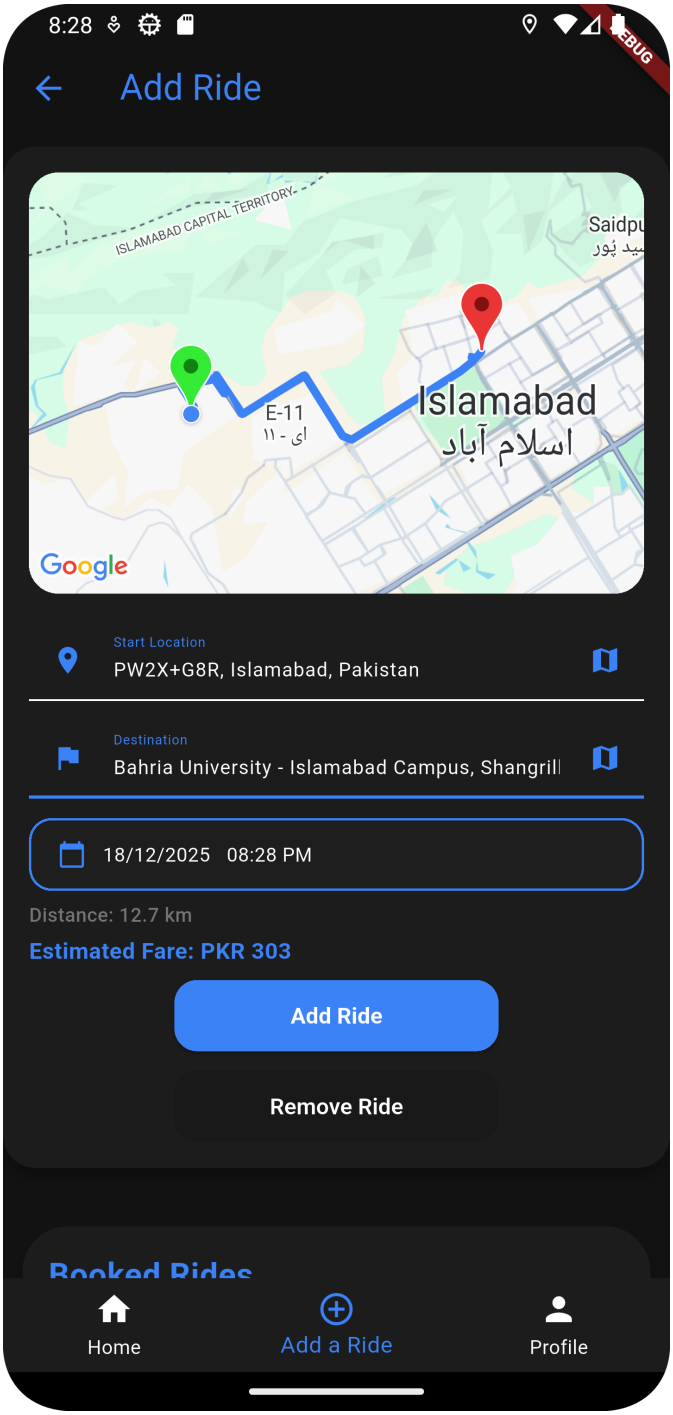


Figure 4.8: Add Ride Page

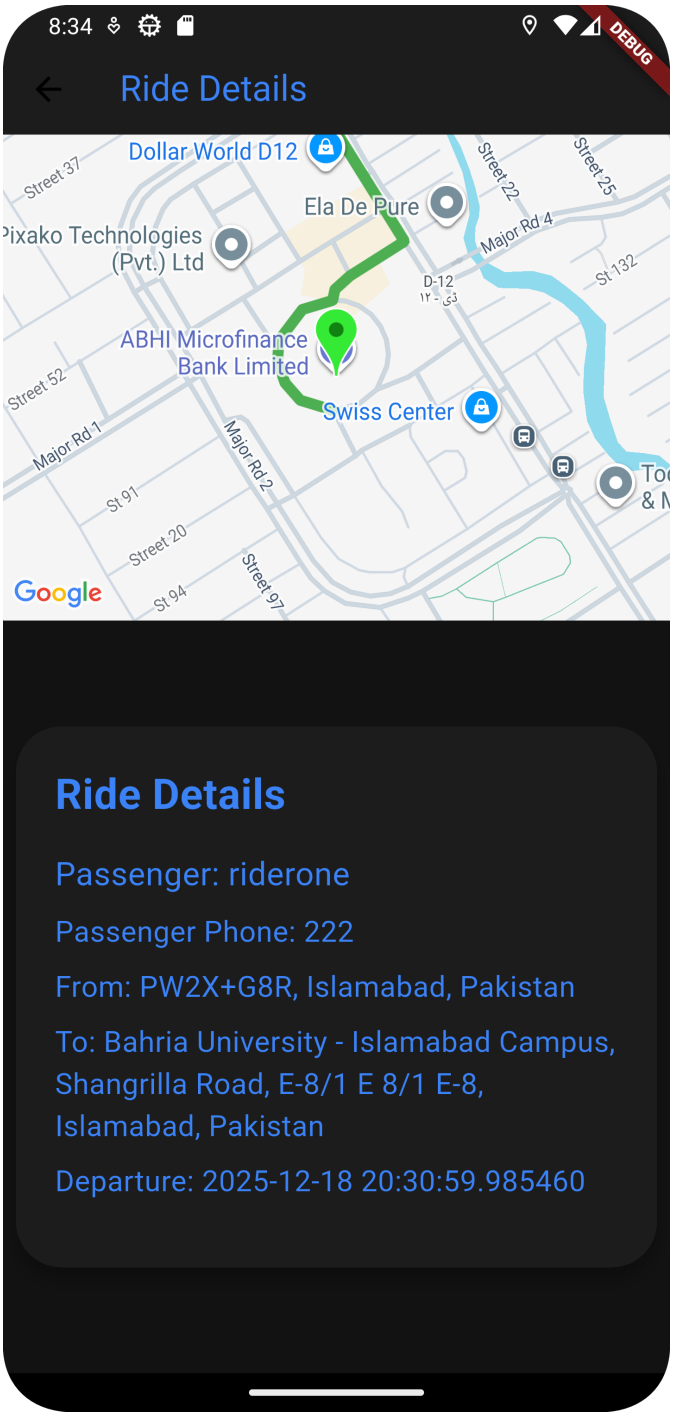


Figure 4.9: Ride Details

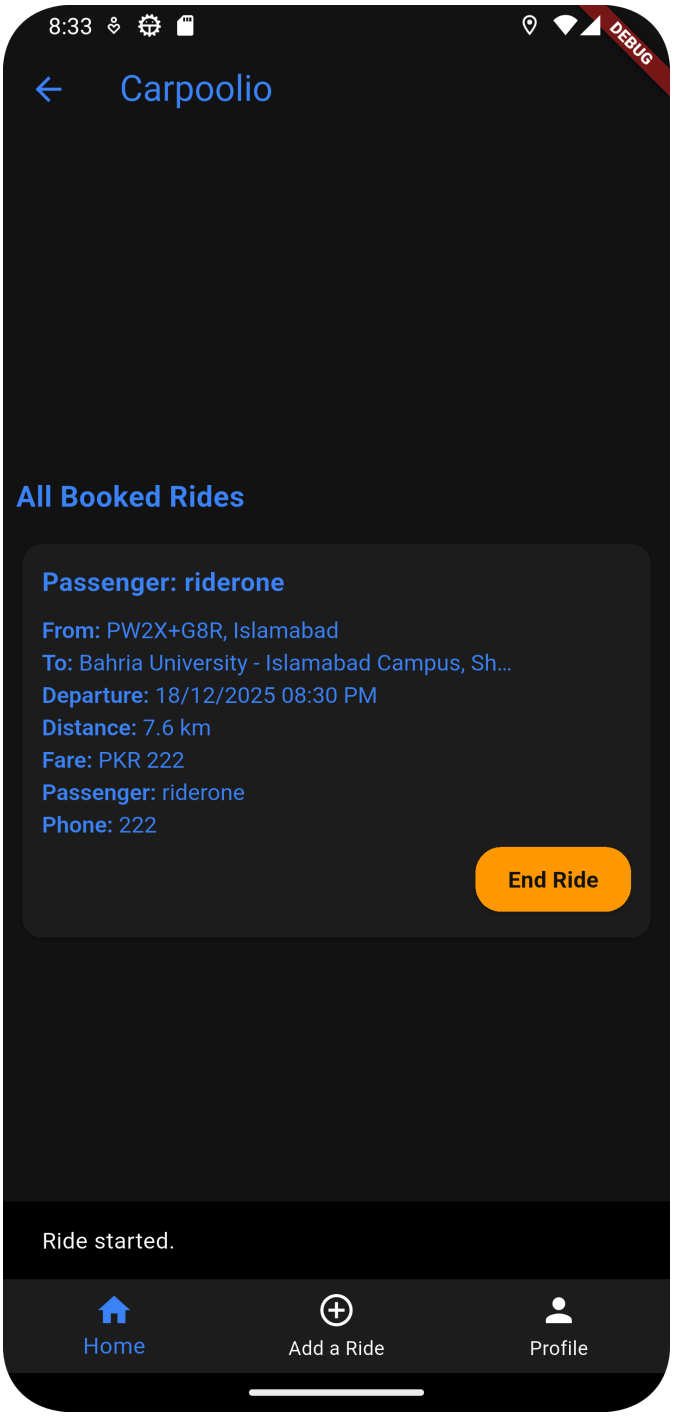


Figure 4.10: Driver's homepage shows all booked rides

# Chapter 5

# System Implementation

This chapter covers the complete description of technical implementation of the Carpoolio application. It fills the gap between the theory of design described in the previous chapter, and the actual implementation of the system. This part of the document describes the development environment, the technologies specifically used, the modular decomposition of the application, and the integration approaches for the APIs and the database, and ends with a description of how the user interface was built. The focus is on how the architectural decisions and principles of the design were transformed into working codes and deployable components.

## 5.1 System Architecture Realization

The section builds on the conceptual system architecture defined in Chapter 4 (System Design), and elaborates on the practical realization of Carpoolio's multi-layered architecture. The chosen cloud-native and serverless approach has been implemented to have inherent scalability, maintainability and responsiveness.

### 5.1.1 Presentation Layer Implementation (Flutter)

The Presentation Layer is fully realized with the help of the Flutter framework by leveraging Dart programming language. This choice supported developing a single codebase that is natively compiled for both Android and iOS platforms which ensured a consistent user experience across different mobile devices[12]. UI components are created as a tree of widgets, following the declarative paradigm in which the UI is a mirror image of the state of the application (Flutter). Key Flutter packages such as `Maps_flutter` and `geolocator` were integrated to offer location based functionality directly in the UI.

### 5.1.2 Application/Backend Layer Implementation

The main business logic of Carpoolio is implemented on the client side inside the Flutter application. Business operations like fare calculations, ride matching algorithms are done directly in UI widgets. Firebase Security Rules are used to perform data validation and access control on the server and all complex business logic is handled on the client side using the app. This approach makes development easy and puts the processing burden on the devices.

### 5.1.3 Data Layer Implementation (Cloud Firestore)

The Carpoolio application uses Cloud Firestore as the main document database that is a NoSQL database. The user, vehicle, ride offer, ride request and booking data models are organized as top-level collections, and each record is represented as a document. The real time syncing features of firestore are used in order to give the client application the most recent ride availability and booking statuses in real-time so that the user can always see the latest update. Patterns of access to data are streamlined to facilitate the efficient reading patterns and security guidelines established to support data integrity and access control.

### 5.1.4 External Service Integration Layer Implementation

Development of integration against the necessary third-party services is handled by special Flutter extensions and Firebase SDKs. The services of Google Maps Platform (Maps SDK in Flutter, Geocoding API, Places API) are incorporated to have an interactive map, the selection of location precisely, and the resolution of addresses. Firebase Authentication is used to provide secure user registration, login, and maintenance of the user session, and hides the user identity complexities.

## 5.2 Development Environment

The Carpoolio application was developed in a consistent and efficient environment that guaranteed congruency, cooperation and productivity.

### 5.2.1 Operating Systems

Development was done on Windows Operating System, leveraging its robust support for Flutter and Android/iOS development toolchains.

### 5.2.2 Integrated Development Environment (IDE)

The main Integrated Development Environment was Visual Studio Code (VS Code). It was perfect to use in rapid development due to its light weight, the wide range of its plugins, and its support of Dart and Flutter. The medium extensions, such as the Flutter and Dart extensions, were essential extensions, which included syntax highlighting, code completion, debugging, and hot reload/restart.

### 5.2.3 SDKs and Toolchains

- **Flutter SDK:** The most recent stable release of the Flutter SDK was continuously utilized, which offers the structure, structural components, and instruments that allowed to develop a cross-platform UI.
- **Dart SDK:** Dart programming language was used to build this application.
- **Android SDK:** Android SDK was used.
- **Firebase CLI:** The Firebase Command Line Interface was used.

## 5.3 Technology Stack

The choice of technologies used in Carpoolio was motivated by the fact that, it needed to develop quickly, and be compatible across various platforms.

### 5.3.1 Frontend Technologies

- **Framework:** Flutter (version 3.24.0)
- **Language:** Dart (version 3.5.3)
- **Key Packages/Libraries:**
  - `google_maps_flutter`
  - `geolocator`
  - `geocoding`
  - `firebase_auth`
  - `cloud_firestore`

### 5.3.2 Backend Technologies

- Firebase Authentication
- Cloud Firestore
- Firebase Storage

### 5.3.3 Mapping and Location Services

- Google Maps Platform
- Google Maps SDK for Flutter
- Google Geocoding API
- Google Places API.

## 5.4 Module Implementation

The codebase of the Carpoolio application is divided into separate modules, and it follows the principles of Object-Oriented Programming (OOP) and facilitates the division of concerns. This modular architecture improves maintainability, testability and scalability.

### 5.4.1 User Management Module

This module includes implementation of user registration, log-in, log-out and profile management. It uses Firebase Authentication SDK directly on the UI screens to provide secure user identity management and makes direct Firestore calls to store user-specific information (i.e., name, phone, user type, car details of drivers) in the users (passengers) collection and Drivers collection. The module supports the two types of accounts, passenger and driver with relevant data fields and validation rules of each category of user. The profile management capabilities enable users to modify their personal details.

### 5.4.2 Ride Offering Module (Driver)

Introduced primarily within the 'EnlistRideScreen', the drivers have the ability to choose pickup and drop-off points on a map, enter the details of the ride (date, time, AC preference) and look at the fare that is automatically calculated with the help of the method Ride.calculateFare. In Firestore, ride information is stored in the enlistedrides collection.

### 5.4.3 Ride Requesting and Booking Module (Passenger)

This module enables the passenger to search and book the ride. The RideRequestScreen enables the passenger to select their starting location and destination. Direct Firestore queries are made on the collection of enlistedrides. Relevant rides are filtered using client-side matching logic. On successful booking, booking records are added in the bookedride collection.

## 5.5 API Integration

### 5.5.1 Firebase Authentication API

Firebase Authentication API was utilized in order to validate identities and conduct user authentication. Firebase Authentication is integrated with the flutter package `firebase_auth` and direct API calls within UI screens. This integration offers functions for:

- User registration
- User login
- User logout

Identity authentication is achieved using direct calls of the Firebase SDK which provides secure management of user identities.

### 5.5.2 Google Maps Platform APIs

The following Google Maps Platform APIs are used:

- Google Maps Flutter
- Geolocator
- Geocoding
- Google Places API:

## 5.6 Database Integration

Cloud Firestore is the backend data database for the application, offering the advantages of real-time data synchronization and a flexible NoSQL data model.

### 5.6.1 Data Structure and Collections

Data is organised into top-level collections: 'users', 'Drivers', 'enlistedrides' and 'bookedride'. Each document contained in these collections is a separate entity and relationships are handled using mostly document IDs.

### 5.6.2 CRUD Operations

The `cloud_firestore` Flutter package is used to perform standard Create, Read, Update, and Delete (CRUD) operations.

### 5.6.3 Real-time Data Synchronization

One of the most important features of Firestore is real-time data synchronization. The application leverages 'snapshots()' listeners on relevant collections (e.g. rideoffers for passengers, 'bookings' for drivers and passengers) and the UI automatically gets updated whenever there is a change in data in the backend. This means that users can always see the most up to date information without having to refresh manually.

### 5.6.4 Firebase Security Rules

Firebase Security Rules are implemented to control access to Firestore data [13], and ensure that users can only read or write data that they have permission to access. Rules are defined to:

- Enforce authentication for all data access.
- Restrict read/write access based on user roles (e.g., only a driver can update their 'vehicle' document).
- Ensure data integrity by validating incoming data against predefined schemas.

## 5.7 User Interface Implementation

The user interface (UI) of the Carpoolio application is carefully implemented using the declarative UI framework of the popular open-source development and release framework, Flutter, with a strong emphasis on providing an intuitive and responsive user experience.

### 5.7.1 Widget-Based Composition

The whole UI consists of a hierarchy of widgets of the Flutter framework. Each screen is constructed by smaller, reusable widgets (e.g. 'Text', 'Container', 'ElevatedButton',

'TextField', 'GoogleMap'). Complex layouts are created using 'Column', 'Row', 'Stack' and 'ListView' widgets, which ensure flexibility and adaptability.

### 5.7.2 Responsive Design

The UI is developed as fully responsive and it can adapt gracefully to screen sizes and orientations of various mobile devices (portrait / landscape). This is achieved through:

- **Flexible Layout Widgets:** 'Expanded' and 'Flexible' widgets are used to create responsive layouts that adapt to different screen sizes and content requirements.
- **Adaptive UI Elements:** Ensuring interactive elements (buttons, text fields) are appropriately sized on all devices.

### 5.7.3 Navigation Flow

Navigation between screens is handled with the help of Navigator API of the Flutter. Named routes are defined for the important screens, making it easy to have clean and easy to maintain logic for navigation. The navigation hierarchy is clearly implemented in the application, with a bottom navigation bar for the main sections of the application (Home, Enlist Ride, Request Ride, Profile) and app bars with back buttons for deeper navigation.

### 5.7.4 User Feedback and Interaction

- **SnackBars:** Used heavily for transient, non-invasive user feedback, i.e., a success message ("Ride offered successfully!"), an error message ("No internet connection") or a warning.
- **Loading Indicators:** Circular progress indicators that are shown for asynchronous operations (e.g. logging in, fetching data, submitting ride) can inform the user that a process is underway.
- **Dialogs:** Used for important user confirmations (e.g. Are you sure you want to cancel this ride? ) or presenting detailed information (e.g. matching ride options ).

### 5.7.5 Adherence to Material Design

The interface design adheres to the Google's Material Design and therefore is clean and intuitive. This implies uniformity in terms of typography, use of color schemes, height and component interactions that contribute to a familiar and pleasant user experience.

# Chapter 6

# System Testing and Evaluation

This chapter presents the methodology that has been employed for the testing of Carpoolio application. The focus of the testing included verification on the core functionalities, the user interactions and system performance in form of manual testing and user feedback iterations.

## 6.1 Testing Approach

The Carpoolio application was tested using the practical approach of manual testing based on an iterative approach to development. Testing took place after each iteration to make sure that new features were functioning correctly and old functionality was preserved.

### 6.1.1 Testing Types

- **Manual Testing:** The most important testing method used to test the user's interaction with the system in a systematic way.
- **User Feedback Testing:** Peer users were recruited to test the application so that they could try to identify any usability problems and suggest any improvements.
- **Device Compatibility Testing:** The application was tested on various Android devices to ensure that it operates and appears the same on all devices.

## 6.2 Test Environment

The testing environment was set up to simulate real-time conditions of use and to provide reliable test results.

### 6.2.1 Development Setup

- **Development Platform:** Windows with Visual Studio Code and Flutter SDK.
- **Backend Services:** Firebase project with Authentication and Firestore for test data management.
- **Testing Devices:** A set of Android devices with different screen size and operating systems versions.

### 6.2.2 Firebase Configuration

A different project of the app on FireBase was used for testing purposes so that no interference occurs with the production data. For test scenarios, this configuration used independent user authentication and database instances.

## 6.3 Test Results

This section introduces the results obtained on testing the main functionalities of the Carpoolio application. All the test cases have been executed manually and each of them passed successfully which proves that the application is working as per the design specification.

### 6.3.1 Authentication Testing

User registration and login were tested to ensure that access control and role based navigation were secure.

### 6.3.2 Core Functionality Testing

Primary features of carpooling such as ride creation, ride booking, and ride tracking in real time were evaluated.

### 6.3.3 Dynamic Matching Testing

The innovative dynamic matching feature was severely tested to ensure that passengers have the ability to join in rides that are underway.

## 6.4 Test Cases

This section describes a few test cases that have been created specifically to validate the core functionalities of the Carpoolio application systematically. Each test case is given in the tabular format, which contains descriptions of the unique identifier, title, initial state,

required input, and the expected outcome. The "Actual Output" and "Status" fields show the success of test execution and verified that the application is successful as per the design requirements.

### 6.4.1 Test Case: Successful Passenger Registration

This is a test case which checks the end to end process of registering a new passenger successfully in the Carpoolio application from initial input to the final navigation.

Table 6.1: Test Case for Successful Passenger Registration

| | |
|---|---|
| **Test Case ID** | TC-001 |
| **Title** | Successful Passenger Registration |
| **Initial State** | The application shows the Landing Screen. No user checked in. |
| **Input** | The user taps "Sign Up", types a valid email, password, full name, phone number, "Passenger" as user type, and taps "Sign Up" button. |
| **Expected Output** | A user account is created in Firebase Authentication. The user profile (name, phone, type of user) is stored in Firestore Database. The user is automatically logged in and taken to the Passenger Home Screen. A “Registration successful!” SnackBar notification is shown. |
| **Actual Output** | A user account was created, profile data was saved to Firestore, the user was navigated to the Home Screen and the "Registration successful!" message was displayed as expected. |
| **Status** | Pass |

### 6.4.2 Test Case: Invalid Login Credentials

This test case is to evaluate the application’s behavior on incorrect or invalid user credentials when logging in to the application, providing appropriate feedback for the same.

Table 6.2: Test Case for Invalid Login Credentials

| | |
|---|---|
| **Test Case ID** | TC-002 |
| **Title** | Invalid Login Credentials |
| **Initial State** | Application is on the Login Screen. No user is logged in. |
| **Input** | The user types in a falsely formatted email or enters an ill-formatted password for an already existing user and clicks the "Log In" button. |
| **Expected Output** | The user is still on the Login Screen and an "Invalid email/password" or "User not found" SnackBar error message is shown. |
| **Actual Output** | The login attempt failed, the user was not taken on the Login Screen and an meaning error message was displayed. |
| **Status** | Pass |

### 6.4.3 Test Case: Driver Successfully Offers a Ride

This test case is for verification of the entire workflow for a given driver to successfully create and submit a new ride offer: location selection and data persistence.

Table 6.3: Test Case for Driver Successfully Offering a Ride

| | |
|---|---|
| **Test Case ID** | TC-003 |
| **Title** | Driver Successfully Offers a Ride |
| **Initial State** | A driver user is logged in and viewing the Home Screen. |
| **Input** | The driver navigates to "Enlist ride" tab, select valid start and destination locations on the map, provide future date and time, and tap "Add Ride" button |
| **Expected Output** | A new `ride_offer` document is created in Firestore containing all the details. A “Ride offered successfully!” SnackBar message appears. The new ride is shown in the driver’s "My Rides" list. |
| **Actual Output** | New ride offer is successfully created in Firestore. Confirmation SnackBar displayed, and the ride is visible in the driver’s list. |
| **Status** | Pass |

### 6.4.4 Test Case: Passenger Successfully Books a Ride

The present test case thoroughly scrutinizes the passenger’s entire interaction with the Carpoolio application and is focused on the significant workflow that starts with the passenger’s transportation demand and ends with the successful reservation of the suitable ride. It validates the system’s ability to guide a passenger through the whole sequence, starting first with searching rides, then displaying available rides, and finally ensuring the confirmation of the booking. This scenario is critical when it comes to validating Carpoolio’s core value proposition from the passenger’s side of the coin, to ensure that the

search, selection and booking mechanisms work united and reliably to provide a convenient carpooling experience.

Table 6.4: Test Case for Passenger Successfully Books a Ride

| | |
|---|---|
| **Test Case ID** | TC-004 |
| **Title** | Passenger Successfully Books a Ride |
| **Initial State** | A Passenger User is logged in and is on the Home Screen. There is at least one corresponding ride in Firestore. |
| **Input** | Passenger navigates to "Find a Ride" tab. Enters start and end locations for the ride. Sets preferred date and time. Taps "Find Matching Rides". Selects a ride that is available from the results. |
| **Expected Output** | A new `booking` document is added to Firestore, linking the rider to the ride. A "Ride booked successfully!" SnackBar message is displayed. The ride that was booked is listed in the "Booked Rides" of the passenger. |
| **Actual Output** | Ride successfully booked, new document of `booking` created in Firestore. Confirmation message dis-played, and ride is visible in passenger's "My Rides". |
| **Status** | Pass |

### 6.4.5 Test Case: User Updates Profile Information

This test case is used to validate the functionality of both driver and passenger users to update their personal information.

Table 6.5: Test Case for User Updating Profile Information

| | |
|---|---|
| **Test Case ID** | TC-005 |
| **Title** | User Updates Profile Information |
| **Initial State** | Any user (Passenger or Driver) is logged in and on the Home Screen. |
| **Input** | User navigates to the "Profile" tab. Edits their Name and/or Phone Number. (If Driver, also has edits car info). Taps "Update Profile" button. |
| **Expected Output** | The user's profile document is updated in Firestore with the new information. A "Profile successfully updated!" SnackBar message is displayed. The revised information can be seen in the profile view. |
| **Actual Output** | Changed profile and vehicle details (if applicable) in Firestore. Confirmation SnackBar displayed and Changes reflected in UI. |
| **Status** | Pass |

### 6.4.6 Test Case: Dynamic Matching - Passenger Joins Active Ride

This test case validates the innovative functionality of dynamic matching where the passenger joins rides already in progress, this is Carpoolio's key technical advantage over static systems of carpooling.

Table 6.6: Test Case for Dynamic Matching - Passenger Joins Active Ride

| | |
|---|---|
| **Test Case ID** | TC-006 |
| **Title** | Dynamic Matching - Passenger Joins Active Ride |
| **Initial State** | A Driver user has started a ride. The driver is already on the way from start to destination. A Passenger user is logged in and on Home Screen. |
| **Input** | Passenger clicks "Find a Ride" tab. Enters starting and ending locations overlapping with the active route of the driver. Taps "Find Matching Rides". Active ride is on the search results. Passenger selects the ride. |
| **Expected Output** | Active ride shows in search results. A new booking document is created in Firestore which links the passenger to the active ride. Then, real time location tracking starts for the passenger. Driver can see the new passenger booking in their active ride view. Both driver and passenger are able to see each other's real-time position. |
| **Actual Output** | Active ride found and booked successfully. Real-time location tracking began immediately. Driver interface updated to display new passenger. Passenger can keep track of the position of the driver in real time. Dynamic matching functionality working as expected. |
| **Status** | Pass |

# Chapter 7

# Conclusions

This chapter is dedicated to the final conclusion of Carpoolio project. It summarizes the entire work done through the project and explains what was achieved at the end of the project. This chapter also discusses the main achievements, challenges faced during the development and lessons learnt from this project. In addition, possible improvements in the future are also discussed. The aim of this chapter is to provide an overall clear picture of the project and the project outcome. It helps in understanding the performance of the project and is effective in solving transportation problems. This chapter also explains how this project helped in learning the practical mobile application development skills.

## 7.1 Project Summary

Carpoolio is a mobile-based carpooling application developed in order to offer an easy, cheap transportation solution. The application is made with the help of flutter for mobile development and other back-end service like firewall. The main aim of this project is to reduce the issues of transportation in the urban areas of Pakistan.

In big cities of Pakistan traffic congestion has been a big problem to tackle. Due to increasing travel by people to pursue their education and work, there is an increase in fuel and road traffic. Carpoolio helps to reduce these issues as it allows people to share rides with others who are travelling on the same route. The application is mainly for students and office workers since they tend to follow set daily routes. Drivers can provide rides and passengers can find available rides and reserve seats. This helps both drivers and passengers to conserve money and time. Carpoolio successfully puts drivers and passengers in one place. It gives features like ride creation, ride booking and real-time location tracking. This project has also proven the importance of using modern mobile technologies to solve real-life problems in Pakistan.

## 7.2 Key Achievements

The development of Carpoolio led to a number of important achievements. These achievements indicate that the project was successful from both technical and functional points of view.

### 7.2.1 Dynamic Matching Innovation

One of the most important things achieved by Carpoolio is the dynamic ride matching feature. This feature enables passengers to participate in already running rides. This is different from traditional carpooling systems where rides have to be booked beforehand. This feature is very helpful for the people who require the transportation suddenly. Passengers can discover the rides around them and easily join them. This makes the application flexible and practical for use in everyday life. Dynamic matching also benefits the drivers by filling empty seats in their vehicles. This helps to improve vehicle usage and aids in the concept of shared transportation. This feature makes Carpoolio more advanced as compared to simple ride-sharing applications.

### 7.2.2 Core Functionality

Carpoolio has all the basic and necessary features of a carpooling application. Full user registration and log in system is put in place. Users are able to create accounts and secure login. The system provides for role-based functionality. Users can be drivers or passengers as per their choice. Drivers can create rides by entering route details, time and available seats. Rides can be searched for and seats booked by passengers. Real-time tracking of locations is also accomplished. This enables the passengers to track the ride, and it boosts trust between users. All the basic features were tried out numerous times to make them work properly.

### 7.2.3 Technology Integration

This project is able to integrate different modern technologies successfully. Google Maps are used for route display, and the tracking of location. Firebase (used for authentication, database management, real-time data updates). Flutter is used to develop the application using single codebase [12]. This integration helped in the development of a smooth and responsive application. It has also made development time shorter and the system easier to manage. These technologies form a very good base for future improvements.

### 7.2.4 User Interface

The user interface of Carpoolio is considered to be a simple and easy to understand system. The principal focus was to make the application easy to understand in all users. Separate interfaces are made for drivers and passengers. Navigation is easy and features are easy to access. The design is avoiding unnecessary complexity and going for basic functionality. The simple design makes the application appropriate for users with basic knowledge of smartphones.

## 7.3 Challenges Faced

During the development stage of Carpoolio, there are many challenges faced. These problems were technical as well as design related. Solving these challenges helped in improving skills and understanding.

### 7.3.1 Firebase Integration

At the beginning, integrating with Firebase was challenging. Understanding authentication, real-time database and security rules took some time. Real-time data synchronization was not easy to do correctly. Initial errors necessarily caused application crashes and data problems. Through testing and learning these problems were solved. This challenge was for better understanding of backend services and cloud based system.

### 7.3.2 Real-time Location Tracking

One of the hardest things to do was the real-time location tracking. Handling continuous location updates while not impacting performance was difficult. Accuracy of the location information was also a problem. Several tests were performed to make it more reliable. This had to be done with patience and careful coding.

### 7.3.3 UI/UX Design

Designing a single application for both the drivers and passengers was not easy. The needs of both types of users are different. Several design changes were made after the testing. Feedback was used to enhance navigation and layout. This iterative process helped to improve the overall user experience.

## 7.4 Lessons Learned

This project offered a lot of important lessons regarding mobile application development, problem solving, and project management. Working on Carpoolio helped in understanding

how real applications are planned, developed, tested and improved with time. I learned many technical and non-technical lessons during this project that will be useful for future work. Building an application from start to finish helped me in gaining confidence in coding, design, and decision-making. I also learned how important careful planning and patience is when working on software projects.

### 7.4.1 Agile Development Benefits

One of the most important things learned was how good it is to use an agile development approach [1]. The idea for developing this project was to not work on the entire application at once, but rather break it down into small tasks and features. Each feature was developed, tested and improved one step at a time. This approach made it easier to cope with problems. If a problem arose, the problem immediately went away without impacting the entire system. Feedback from testing helped features to be improved before entering the next stage. Agile development also helped to manage the time better. Small goals were established on each phase, hence the work was less stressful. This method was found to be very effective for student level projects where the requirements are allowed to be changed during the development process.

### 7.4.2 Firebase Efficiency

Another important lesson learned was that it is efficient to use Firebase as a backend service. Firebase took control of many backend functionalities like authentication, database and real-time. This lowered the entire workload and complication in the project. Instead of spending time on server setup and maintenance more focus was given towards the application features and user experience. Firebase also made real-time data synchronization easier; which is very important for a carpooling application. This project demonstrated that cloud-based backend services are really helpful for small applications to medium applications. Firebase proved to be a reliable tool that was easy to scale for future improvements.

### 7.4.3 Importance of Testing

The fact that testing is important was one of the best lessons that was learned during this project. Continuous manual testing helped in identifying errors and unexpected behaviour in the early stages. While testing, I found a lot of little bugs which could have made bigger problems for later on. Testing each feature individually before combining everything really helped to make the app more stable. It also allowed me to have a better idea of how users might actually use the app. Doing tests over and over again made me more confident that everything works the way it should. This project really showed me that testing is not

something that you can skip, it is an important part of making any software, even a simple app, reliable and easy to use.

### 7.4.4 Problem Solving and Debugging Skills

This project resulted in improving my problem solving and debugging skills. I made many mistakes in the development, in particular linked to the integration with FireBase. Instead of giving up, I overcame these issues by researching, testing, and trying out different solutions. Debugging helped in understanding the internal working of the application.

### 7.4.5 Time Management and Planning

Time management was also another important lesson gained during the project. At the start some of the tasks took more time than anticipated. This helped in understanding the importance of proper planning. By breaking things down into smaller chunks, one was able to get work done on time. Setting realistic deadlines made the process of development smoother. This lesson will be really helpful for the future in academic and professional life project.

### 7.4.6 Belief in Mobile Application Development

This project benefited in understanding the importance of user requirements. Drivers and passengers have different requirements and the application had to support both. Design decisions were enhanced after taking user perspective into account This helped in the creation of a more usable and practical application. Understanding user behavior is very important for developing successful applications.

### 7.4.7 Understanding User Needs

Finally, this project brought more confidence in the field of mobile application development. Completing a whole working application was a huge proof-of-concept that complex systems can be built given proper efforts and learnings. This confidence will be helpful in future projects and professional job. The lessons learned from Carpoolio will be useful when it comes to developing better and more advanced applications in the future.

## 7.5 Future Enhancements

Although Carpoolio is functional, there are many improvements in the future that can be made to enhance the application.

### 7.5.1 Payment System Integration

Currently, cash payments are the only payments that are supported. Adding digital payment options can be added. This will improve convenience along with the ability to track the transactions.

### 7.5.2 Expanded Device Testing

Testing was carried out on limited devices. more android devices will improve the compatibility while testing. This will make sure that performances of different devices will be better.

### 7.5.3 Performance Optimization

The database queries and loading speed can be optimized. This will help in decreasing the delay and enhancing the user experience during peak-time usage.

### 7.5.4 Basic Admin Features

An admin dashboard can be added, which can be used to control the users and system activity. This will help control the misuse and monitoring of performance.

### 7.5.5 Enhanced Notifications

Push notifications can be added to include ride updates and confirmations. This will keep the users informed and enhance the engagement.

## 7.6 Final Assessment

In the end, the conclusion can be made that the Carpoolio project was successfully completed and that it was able to meet its primary objectives. The application is functional and can be used to carpool in the Pakistani cities. The concept of linking passengers and drivers on a unified mobile platform was done in an appropriate manner.

Carpoolio assists in transportation issues encountered by students and office workers on a daily basis. Sharing of rides will help users to save money and lessen fuel usage. This also assists in traffic reduction on roads that are congested. It is indicated in the given application that the ride sharing can be effective in those cities when the number of people utilizing public transport is scarce or excessive.

Technically, it was a good learning experience. Flutter and Firebase were useful in the development of an actual mobile application. Key aspects like user authentication, ride creation, booking and live location tracking were developed and put in practice. Although

a few issues were realized in the course of development, they were corrected in testing and amendments.

This project was also useful in getting to realize how the theory that is covered in classes can be applied to life. Other than merely writing the code, the project involved planning, designing, testing and troubleshooting. The steps were useful in acquiring hands-on knowledge in software development.

Overall, Carpoolio provides a strong base for future development. Additional features can be added and performance can be enhanced in the course of time. The project demonstrates that student-level academic work may produce useful and practical solutions. To sum up, it is evident that Carpoolio is a successful project that achieves its goals and can be further developed.

# Appendix A

# User Manual

# References


[1] Agile Alliance. Manifesto for agile software development. Foundational document for agile methodologies. Cited on pp. 6 and 63.

[2] Google Cloud Documentation. Firebase authentication: Secure authentication. Official documentation. Cited on pp. 7 and 41.

[3] Google Maps Platform. Google maps platform for flutter. Official Flutter package documentation. Cited on p. 8.

[4] R. Singh and A. Kaur. A review on ride-sharing applications and their impact on urban transportation. *International Journal of Computer Applications*, 180(43):35–40, 2018. Cited on p. 10.

[5] Business Research Insights. Global carpooling market size, share, growth, analysis, report 2025-2035. Market analysis and growth projections. Cited on p. 10.

[6] International Energy Agency. Transport and co2 emissions. Environmental impact of transportation. Cited on p. 10.

[7] S.A. Khan, Z. Iqbal, and M. Usman. Design and implementation of a smart carpooling system for university commuters. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 789–794. IEEE, 2019. Cited on pp. 10 and 13.

[8] Savari AI. Savari multi-service transportation platform. Platform overview and services. Cited on p. 11.

[9] DriveLU. Drivelu (ride-sharing/carpooling service). Official Website. Cited on p. 11.

[10] inDrive. indrive | ride-hailing, intercity, cargo, courier. Official Website. Cited on p. 12.

[11] inDrive Team. How indrive works: Fair prices for everyone. Business model and pricing mechanism. Cited on p. 12.

[12] Flutter Documentation. Build apps for any screen. Official documentation. Cited on pp. 47 and 61.

[13] Google Cloud Documentation. Cloud firestore: Flexible, scalable, serverless database. Official documentation. Cited on p. 52.