

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
JNANA SANGAMA, BELAGAVI- 590018, KARNATAKA, INDIA



**A MINI PROJECT(BCS586) REPORT**

**on**

**“RIDESHARE”**

**Submitted in partial fulfilment of the requirements for the award of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**Submitted By**

<b>Name</b>	<b>USN</b>
<b>PRATHAM K CHANDRA</b>	<b>4VP22CS069</b>
<b>SUMANTHA NARAYANA M</b>	<b>4VP22CS104</b>
<b>VINITH G P</b>	<b>4VP22CS122</b>
<b>VIRAJ R GOWDA</b>	<b>4VP22CS124</b>

**Under the Guidance of**

**Prof. Pradeep Kumar K G**

**Assistant Professor**



---

**DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING**  
**VIVEKANANDA COLLEGE OF ENGINEERING & TECHNOLOGY**

[A Unit of Vivekananda Vidyavardhaka Sangha Puttur (R)]

Affiliated to Visvesvaraya Technological University and Approved by AICTE New Delhi & Govt., of Karnataka

Nehru Nagar, Puttur - 574 203, DK, Karnataka, India.

**December, 2024**

# **VIVEKANANDA COLLEGE OF ENGINEERING & TECHNOLOGY**

[A Unit of Vivekananda VidyavardhakaSangha Puttur (R)]

Affiliated to Visvesvaraya Technological University and Approved by AICTE New Delhi & Govt. of Karnataka

Nehru Nagar, Puttur - 574203, DK, Karnataka, India

## **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



### **CERTIFICATE**

Certified that the project work entitled “**RIDE SHARE**” is carried out by bonafide students **Mr. Pratham K Chandra, Mr. Sumantha Narayana M, Mr. Vinith G P, Mr. Viraj R Gowda** bearing USNs **4VP22CS069, 4VP22CS104, 4VP22CS122 and 4VP22CS124** respectively of **Vivekananda College of Engineering & Technology, Puttur** in partial fulfilment for the award of **Bachelor of Engineering** in **Computer Science & Engineering** under the **Visvesvaraya Technological University, Belagavi** during the year 2024-25. It is certified that all corrections/suggestions indicated during Internal Assessment have been incorporated in the report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of MiniProject work prescribed for the degree.

---

**Signature of the Guide**  
**Prof. Pradeep Kumar K G**

---

**Signature of the Project**  
**Coordinator**  
**Prof. Radhika Shetty D S**

---

**Signature of the HOD**  
**Dr. Nischay Kumar Hegde**

# DECLARATION

We, **Mr. Pratham K Chandra (4VP22CS069), Mr. Sumantha Narayana M (4VP22CS104), Mr. Vinith G P(4VP22CS122), Mr. Viraj R Gowda(4VP22CS124)** students of B.E. 5<sup>th</sup> Semester in Computer Science & Engineering, **Vivekananda College of Engineering & Technology**, Puttur, hereby declare that the mini project work entitled “**RIDE SHARE**” has been carried out by us at VCET, Puttur, under the guidance of **Prof. Pradeep Kumar K G**, Assistant Professor, Department of Computer Science & Engineering, Vivekananda College of Engineering & Technology, Puttur, and submitted in partial fulfilment of the requirements for the award of degree in **Bachelor of Engineering in Computer Science & Engineering** by **Visvesvaraya Technological University, Belagavi** during the academic year 2024-2025.

Name of the students	USN	Signature with date
Pratham K Chandra	4VP22CS069	
Sumantha Narayana M	4VP22CS104	
Vinith G P	4VP22CS122	
Viraj R Gowda	4VP22CS124	

Date:

Place: Puttur

# ABSTRACT

The growth of ride-sharing services has significantly reshaped urban transportation, providing users with flexible, affordable, and convenient commuting options. Key findings suggest that safety features, such as driver background checks, real-time ride tracking, and in-app emergency buttons, are crucial in enhancing user trust and adoption. These safety features significantly influence users' perceptions of ride-sharing platforms, often determining their choice to use these services over traditional taxis.

Operationally, ride-sharing companies leverage data analytics and machine learning to optimize demand prediction, reduce wait times, and improve resource allocation, thus enhancing service quality. However, the socio-economic implications are complex, as ride-sharing creates flexible income opportunities for drivers while raising concerns about job security, benefits, and fair labor practices within the gig economy. Environmental impacts also present a dual narrative: while ride-sharing can reduce private vehicle ownership, it may contribute to increased vehicle miles traveled, leading to higher emissions and congestion in urban areas.

# ACKNOWLEDGEMENT

We take this opportunity to express our deep heartfelt gratitude to all those people who have helped us in the successful completion of the project.

First and foremost, we would like to express our sincere gratitude to our guide, **Prof. Pradeep Kumar K. G.** for providing excellent guidance, encouragement and inspiration throughout the project work. Without his invaluable guidance, this work would never have been a successful one.

We would like to thank our Project Coordinator, **Prof. Radhika Shetty D. S.** for providing necessary guidance and support.

We would like to express my sincere gratitude to our Head of the Department of Computer Science & Engineering, **Dr. Nischaykumar Hegde** for his guidance and inspiration.

We would like to thank our Principal, **Dr. Mahesh Prasanna K.** for providing all the facilities and a proper environment to work in the college campus.

We would like to thank our management for providing the necessary infrastructure to carry out the project work.

We are thankful to all the teaching and non-teaching staff members of Computer Science & Engineering Department for their help and needed support rendered throughout the project.

# TABLE OF CONTENTS

<b>Title</b>	<b>Page No.</b>
<b>List of figures</b>	<b>I</b>
<b>List of tables</b>	<b>II</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>01</b>
1.1 Literature Survey	02
1.2 Proposed System	04
<b>CHAPTER 2 REQUIREMENT SPECIFICATION AND ANALYSIS</b>	<b>05</b>
2.1 Introduction	05
2.2 Functional Requirements	05
2.2.1 User Registration and Authentication	05
2.2.2 Ride Booking System	05
2.2.3 Driver Rider matching	05
2.2.4 Trip history and Management	06
2.2.5 Notifications and alerts	06
2.2.6 Ride Pooling	06
2.3 Non- Functional Requirements	06
2.3.1 Performance	06
2.3.2 Scalability	07
2.3.3 Reliability	07
2.3.4 Availability	07
2.3.5 Useability	07
2.3.6 Security	07
2.3.7 Maintainability	08
2.3.8 Portability	08
2.3.9 Compatibility	08
2.3.10 Efficiency	08
2.3.11 Backup and Recovery	08
2.4 Software Requirements	09
2.4.1 About Java	09
2.4.2 About Android Studio	10

2.5 Hardware Requirements	14
<b>CHAPTER 3    SYSTEM DESIGN</b>	<b>13</b>
3.1 Introduction	13
3.2 System Architecture Design	13
3.3 Flow Chart	16
3.4 Use Case Diagram	17
3.5 Sequence Diagram	18
3.6 Data Flow Diagram	21
<b>CHAPTER 4    SYSTEM IMPLEMENTATON</b>	<b>22</b>
4.1 Introduction	22
4.2 Implementation Approaches	22
4.3 Implementation Steps	23
<b>CHAPTER 5    SYSTEM TESTING</b>	<b>26</b>
5.1 Test Objectives	26
5.2 Features to be tested	26
5.3 Types of Tests	27
5.3.1 Unit testing	27
5.3.2 Integration testing	27
5.3.3 Functional testing	27
5.3.4 System testing	28
5.3.5 Acceptance testing	28
5.4 Test case to be satisfied	29
<b>CHAPTER 6    SCREENSHOTS</b>	<b>30</b>
<b>CHAPTER 7    CONCLUSION AND SCOPE</b>	<b>33</b>
<b>FOR FUTURE</b>	
<b>REFERENCES</b>	<b>34</b>

# LIST OF FIGURES

<b>Figure No.</b>	<b>Title</b>	<b>Page No.</b>
3.1	Architecture of the system	13
3.2	Flow chart diagram of the project	16
3.3	Use-case diagram	17
3.4	Illustrates the sequence diagram of rider/user	19
3.5	Illustrates the sequence diagram of passenger	20
3.6	Dataflow diagram of the system	21
6.1	Login Page	30
6.2	Rider/Passenger login details	31
6.3	User selecting roles	31
6.4	User entering destination and via route	32
6.5	Ride details saved	32



## LIST OF TABLES

Table No	Title	Page No
Table 5.1	Test Cases	29

## CHAPTER 1

### INTRODUCTION

Ride-sharing has transformed urban transportation, rapidly emerging as a prominent alternative to traditional taxi services and public transit. The rise of companies like Uber, Lyft, Didi, and Ola has redefined how people commute, providing flexible, on-demand transportation solutions that appeal to diverse populations, including daily commuters, tourists, and people in need of quick and reliable transport. This transformation reflects broader societal trends favoring the sharing economy, where access to services is valued over ownership, leading to innovations in various sectors, including lodging (Airbnb), shared workspaces (WeWork), and car rentals (Zipcar).

The popularity of ride-sharing services can be attributed to several factors: convenience, affordability, the ability to book rides via mobile apps, and the integration of advanced technologies like GPS tracking and digital payments. Users appreciate the ease of booking a ride with just a few clicks, the ability to track the driver's location, and the upfront cost estimation, which provides transparency in pricing. For drivers, ride-sharing offers flexible working hours and an opportunity for additional income. This flexibility attracts a diverse pool of drivers, including students, retirees, and full-time workers looking for supplementary earnings. Despite its widespread acceptance, the rapid growth of ride-sharing services has not come without controversy and significant challenges. Safety concerns, regulatory conflicts, and socio-economic impacts on traditional taxi services and labor rights have sparked debates among policymakers, users, and companies. Incidents involving driver misconduct, passenger harassment, and accidents have drawn attention to the need for robust safety measures.

Meanwhile, traditional taxi operators have raised concerns about unfair competition, arguing that ride-sharing companies often face less stringent regulations. These concerns extend to the gig economy's impact on workers, as ride-sharing drivers typically lack employee benefits such as health insurance, paid leave, and job security.

The socio-economic implications of ride-sharing services also raise important questions for the public transportation usage. While some studies suggest that ride-sharing can reduce the need for private vehicle ownership, others indicate an increase in vehicle miles traveled (VMT), leading to concerns about exacerbated traffic congestion and environmental impacts. Additionally, ride-sharing's impact on public transportation is complex; while it can serve as a complementary service, filling gaps in transit networks and providing first-mile/last-mile connectivity.

In addressing these multi-dimensional issues, this paper synthesizes insights from 23 research studies on ride-sharing, aiming to provide a holistic understanding of the topic. The analysis covers user behavior, safety features, operational strategies, socio-economic consequences, and regulatory dynamics, offering a comprehensive view of the industry's current state and future prospects. The reviewed studies employ various methodologies, including quantitative ,qualitative interviews, case studies, and statistical modeling, to explore the different facets of ride-sharing.

The discussion will begin by examining user perceptions and behaviors related to ride-sharing, focusing on factors influencing service adoption, safety concerns, and user satisfaction. It will then delve into the operational strategies used by ride-sharing companies to enhance service quality and efficiency. The socio-economic impact section will explore the implications for drivers, traditional taxi operators, and the broader labor market, while the environmental impact analysis will assess ride-sharing's effect on urban congestion, emissions, and sustainable transportation initiatives.

Finally, the paper will discuss the regulatory challenges facing ride-sharing companies and explore emerging trends and future opportunities in the industry, such as the integration of autonomous vehicles and the development of shared mobility ecosystems. By synthesizing these diverse perspectives, the paper aims to contribute to the ongoing discourse on the sustainable and equitable development of ride-sharing services, addressing key questions about the future of urban mobility in an increasingly digital and interconnected world.

Moreover, the regulatory environment surrounding ride-sharing continues to evolve, often lagging behind the rapid pace of technological innovation. While some jurisdictions have embraced ride-sharing as a modern solution to urban mobility challenges, others have imposed strict regulations to protect traditional taxi industries and address concerns related to congestion, pollution, and labor rights.

The regulatory landscape is complex, with different cities and countries adopting varied approaches to issues such as fare control, driver licensing, and insurance requirements.

These regulations not only impact the operational strategies of ride-sharing companies but also influence market dynamics, competition, and service quality.

## **1.1 LITERATURE SURVEY**

M Andersson et.al [1], discusses how peer-to-peer service-sharing platforms enable large-scale collaborative consumption, emphasizing the mutual benefits of sharing services and resources efficiently.

---

Nusrat Jahan Farin et.al [2], propose a framework for dynamic vehicle pooling and ride-sharing, aiming to enhance urban mobility and reduce congestion using real-time pooling algorithms.

NJ Farin et.al [3], elaborates on dynamic vehicle pooling frameworks, optimizing algorithms for better efficiency and usability.

C Lee et.al [4], investigates the competitive dynamics between ride-sharing platforms, examining pricing strategies, market entry barriers, and user adoption patterns.

Susan Shaheen et.al [5], discusses shared mobility trends, focusing on the opportunities and challenges of ride-hailing and pooling services in urban areas.

A Tafreshian et.al [6], highlights the advancements in ride-matching algorithms for P2P ride-sharing and suggests future research directions.

Y Sun et.al [7], focuses on nonprofit P2P ridesharing models, optimizing ride-sharing solutions for social good rather than profit maximization.

A Tafreshian et.al [8], presents a novel graph-based method to partition ridesharing trips dynamically, improving system efficiency and rider satisfaction.

Amirmahdi Tafreshian et.al [9], explore's of graph partitioning techniques applied to dynamic ride-sharing scenarios, emphasizing scalability and optimization.

Piyush Agrawal et.al [10], highlights the features and potential of a proposed ride-sharing app, POOL, which facilitates efficient P2P ride-sharing.

L Mitropoulos et.al [11], review's of the literature on ride-sharing platforms, examining user behavior, adoption factors, and barriers to implementation.

Barnali Gupta Banik et.al [12], explores using blockchain technology to create secure, decentralized ride-sharing platforms.

A Soltani et.al [13], analyze user segments in Adelaide, exploring preferences, challenges, and opportunities for ridesharing adoption in the region.

LC Martins et.al [14], addresses the challenges and opportunities of integrating ride-sharing into smart city infrastructures, emphasizing sustainability.

S Jang et.al [15], examines how quality indicators such as service reliability and user feedback impact travelers' demand for P2P ridesharing.

L Mitropoulos et.al [16], listing of the earlier Mitropoulos et al. work on ride-sharing platform adoption factors and barriers.

Sathya A. Renu et.al [17], show's duplicate of the earlier work on secure ride-sharing using blockchain.

Leandro do C et.al [18], explore's of algorithmic challenges in optimizing ride-sharing operations for sustainable smart cities.

---

Ajinkya Ghorpade et.al [19], presents the development and features of the POOL app, aimed at creating efficient P2P ride-sharing experiences.

S Abdikerimova et.al [20], studies the P2P concept to multi-risk insurance and mutual aid systems, exploring synergies with ride-sharing mode.

## **1.2 PROPOSED SYSTEM**

The proposed rideshare app system is developed using Android Studio with Java and leverages Firebase for backend services, offering a scalable, real-time, and cost-effective solution. The app connects riders and drivers through a seamless interface, enabling essential features like ride booking, real-time tracking, payment processing, and post-ride feedback. Riders can request rides by specifying pickup and drop-off locations, which are processed in the Firebase Realtime Database. A ride-matching algorithm identifies the nearest available driver .

Drivers benefit from features like an availability toggle, ride request management, and a dashboard for tracking earnings and trip history. The app supports real-time tracking during trips by continuously syncing location data between the rider and driver using Firebase's real-time capabilities.

Administrators access a web-based admin panel to oversee user accounts, resolve disputes, and monitor app performance. Firebase Authentication secures user login using email, phone numbers, or social media accounts, while Firebase Analytics provides insights into user behavior and system performance. The system is designed with scalability in mind, allowing it to handle increasing traffic and data as the user base grows.

The Firebase ecosystem offers significant advantages, such as real-time data synchronization for ride requests, authentication, and encrypted database storage. Additionally, Firebase's pay-as-you-go model is cost-effective, making it ideal for startups and small businesses. Future enhancements include carpooling, dynamic pricing based on demand, Offline mode features can also be integrated to ensure the app functions in areas with limited internet connectivity.

Overall, the combination of Android Studio's robust development environment and Firebase's powerful backend services creates a scalable, efficient, and user-friendly rideshare app. This system not only meets the current needs of riders and drivers but also provides a flexible foundation for future growth and feature expansions.

## **CHAPTER 2**

# **REQUIREMENT SPECIFICATION AND ANALYSIS**

## **2.1 INTRODUCTION**

Requirement analysis is the process of creating a score for a system's effort. The result is neither coordinated nor integrated and often simply does not work. Requirement analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

## **2.2 FUNCTIONAL REQUIREMENTS**

Functional requirements define the specific actions, tasks, or functions a system must perform to meet its objectives. These requirements are derived from user needs and business goals and describe how the system should behave in different scenarios. For a ride-sharing platform, the functional requirements can be outlined as follows:

### **2.2.1 User Registration and Authentication**

To access the ride-sharing system, users including drivers and passengers are required to create an account using their email, phone number, and a secure password. The system distinguishes between various user roles, such as Rider, Passenger, and Admin, to ensure appropriate access and functionality. To enhance security, features like password reset and are implemented, safeguarding user accounts against unauthorized access.

### **2.2.2 Ride Booking System**

A ride booking system is a digital platform that enables users to request transportation services via a smartphone application. It connects passengers with drivers, facilitating real-time ride requests, GPS tracking, fare estimation, and cashless payments. This system enhances urban mobility by providing convenient, efficient, and accessible transportation options

### **2.2.3 Driver-Rider Matching**

An efficient ride-sharing system employs real-time algorithms to match drivers and passengers based on proximity, route similarity, and vehicle availability. Once a match is identified, both parties receive immediate notifications. The system also allows for cancellations when necessary, ensuring flexibility and user satisfaction.

### **2.2.4 Trip History and Management**

This allows the users to access and manage their ride history. View past rides with date, fare, time and route details. Option to rebook a previous trip.

### **2.2.5 Notifications and Alerts**

Users will receive real-time notifications for various trip updates, including booking confirmations, arrival alerts, and ride completions. Additionally, they will be notified of promotional offers and ride reminders to enhance their experience. These updates will be sent via push notifications, SMS, and email alerts, ensuring users stay informed throughout their journey.

### **2.2.6 Ride Pooling**

This feature enables passengers with similar routes to share rides, promoting both cost-efficiency and sustainability. It identifies and groups passengers traveling along overlapping routes, then calculates a shared fare based on the distance each passenger travels. Additionally, all parties are notified of the designated pickup and drop-off points to ensure a smooth and coordinated ride-sharing experience.

## **2.3 NON-FUNCTIONAL REQUIREMENTS**

In system engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. These requirements focus on aspects related to usability, security, performance, reliability, scalability, maintainability, and other qualities that contribute to the overall user experience and system operation. Unlike functional requirements, which describe what the system should do, non-functional requirements describe how the system should perform. Non-functional requirements significantly influence the user experience and overall satisfaction of stakeholders. For instance, a system with poor performance or security vulnerabilities may result in frustration among users and stakeholders, impacting its adoption and success.

### **2.3.1 Performance**

The system's performance requirements define its ability to handle workloads efficiently and deliver fast responses to users. Under normal conditions, the app should load within 2 seconds, ensuring a smooth user experience. It must also be capable of handling up to 10,000 concurrent users without any noticeable performance degradation, demonstrate its scalability and robustness. Additionally, the average response time for critical operations, such as ride-matching algorithms, should remain under 1 second to

provide quick and reliable results, ensuring seamless service even during peak usage periods. These benchmarks collectively ensure the app operates effectively under varying loads while maintaining user satisfaction.

### **2.3.2 Scalability**

The system's scalability defines its ability to grow and efficiently manage increasing user loads as demand rises. It must support a 10x growth in users during peak periods without requiring significant re-architecture, ensuring seamless expansion as the user base grows. To accommodate surges in demand, the system should leverage dynamic resource allocation, such as cloud-based infrastructure, to automatically scale and maintain optimal performance. This ensures the system remains responsive and reliable, even during unexpected spikes in usage.

### **2.3.3 Reliability**

The system's reliability ensures continuous operation with minimal interruptions. It should achieve 99.9% uptime, allowing for no more than 8.76 hours of downtime annually. In the event of a server failure, the system must seamlessly switch to backup servers within 5 seconds, ensuring uninterrupted service and maintaining user trust.

### **2.3.4 Availability**

The system's availability ensures it remains operational and accessible at all times. The app must be available 24/7 for booking and ride management.

### **2.3.5 Usability**

The system's usability ensures it is user-friendly and accessible to everyone. Users should be able to complete the ride-booking process in no more than 3 steps through an intuitive interface suitable for all age groups, including non-technical users. Additionally, accessibility features like voice commands and screen reader support must be provided for visually impaired users.

### **2.3.6 Security**

The system's security ensures protection against unauthorized access and attacks while safeguarding user data. All sensitive data, such as passwords and payment details, must be encrypted using strong standards like AES-256. Secure communication protocols, such as HTTPS and TLS 1.2 or higher, must be implemented for all data exchanges to prevent interception. Additionally, multi-factor authentication (MFA) should be included for user logins to enhance account security and protect.



### **2.3.7 Maintainability**

The system's maintainability ensures it can be easily updated, fixed, and enhanced over time. The codebase should follow industry best practices and be modular to simplify updates and improvements. Patches and updates must be deployable without disrupting live services, ensuring minimal downtime. Additionally, system logs should provide detailed error reports to facilitate efficient troubleshooting and issue resolution.

### **2.3.8 Portability**

The system's portability ensures it can operate effectively across different platforms and environments. The app should run seamlessly on Android devices, while the database and backend systems must be deployable on multiple cloud platforms, such as Firebase, to ensure flexibility and compatibility.

### **2.3.9 Compatibility**

The system's compatibility ensures smooth integration with other platforms and devices. The app must function seamlessly on Android versions 9 and above, ensuring broad accessibility and consistent performance.

### **2.3.10 Efficiency**

The system's efficiency ensures optimal use of resources to minimize costs and maximize performance. CPU usage should remain below 70% under normal conditions, while caching mechanisms must be implemented to reduce database queries by 50% for frequently accessed data, enhancing speed and resource utilization.

### **2.3.11 Backup and Recovery**

The system's data integrity ensures data is preserved and recoverable in the event of failures. Automatic daily backups should be implemented for all user data to safeguard against data loss. In the event of a disaster, the system must recover within 15 minutes with no data loss, ensuring continuity and minimizing downtime.

## **2.4 SOFTWARE REQUIREMENTS**

Operating System : 64-bit Microsoft® Windows® 11

Language : Java

Tool Kit : Android Studio Lady Bug 2024.2.

## 2.4.1 About Java

Java is a versatile, high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle) in 1995. It is designed to be platform-independent, enabling developers to "write once, run anywhere" (WORA). This is achieved through the Java Virtual Machine (JVM), which executes Java bytecode on any.

### Key Features of Java:

- **Platform Independence:** Java programs are compiled into bytecode, which can run on any device with a JVM, regardless of the underlying operating system.
- **Object-Oriented:** Java emphasizes object-oriented principles like inheritance, encapsulation, polymorphism, and abstraction, making it modular and reusable.
- **Robust and Secure:** Java has strong memory management, exception handling, and a security model that includes runtime checks and sandboxing for untrusted code.
- **Multithreading:** It provides built-in support for creating and managing multiple threads, enabling parallel processing and improved performance in applications.
- **Automatic Memory Management:** Java includes garbage collection to automatically handle memory allocation and deallocation, reducing memory leaks.
- **Rich Standard Library:** Java comes with an extensive set of APIs and libraries for tasks like data structures, networking, database connectivity, and graphical user interfaces (GUIs).
- **Dynamic and Extensible:** Java supports dynamic linking of new code, and its modularity allows adding functionality at runtime.

### Key Components of Java:

- **JVM (Java Virtual Machine):** Executes Java bytecode, making Java platform-independent.
- **JRE (Java Runtime Environment):** Includes the JVM and standard libraries required to run Java applications.
- **JDK (Java Development Kit):** A complete development environment containing the compiler, tools, and libraries for building and running Java applications.

### 2.4.2 About Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android application development, created by Google and based on IntelliJ IDEA by JetBrains. Launched in 2013, it provides developers with the tools and features needed to design, develop, debug, and deploy Android apps for a variety of devices, including smartphones, tablets, wearables, TVs, and more.

#### Key Features of Android Studio:

- **Code Editor:** Offers intelligent code completion, real-time syntax highlighting, and linting to detect errors and optimize performance.
- **Layout Editor:** Drag-and-drop visual design editor for creating XML-based UI layouts.
- **Emulator:** Built-in Android Emulator to test and debug apps on virtual device.
- **Gradle Build System:** Manages project builds and dependencies efficiently.
- **Instant Run:** Allows developers to see code changes reflected in the app without restarting it, speeding up development. Allows developers to see code changes reflected in the app without restarting it, speeding up development.
- **Version Control Integration:** Supports Git, SVN, and Mercurial for collaborative coding and version management.
- **Profiler Tools:** Analyze app performance, memory usage, CPU usage, and network traffic to optimize performance.
- **Support for Multiple Programming Languages:** Primarily supports Kotlin and Java, but also allows C++ (via JNI) for performance-critical tasks.
- **Integration with Firebase:** Seamless integration with Firebase for features like real-time databases, analytics, authentication, and cloud messaging.
- **Plugin System:** Extend functionality using plugins such as Kotlin plugins, Lint checks, and more. Allows developers to see code changes reflected in the app without restarting it, speeding up development.

**Version Control Integration:** Supports Git, SVN, and Mercurial for collaborative coding and version management.

**Profiler Tools:** Analyze app performance, memory usage, CPU usage, and network traffic to optimize performance.

**Support for Multiple Programming Languages:** Primarily supports Kotlin and Java, but also allows C++ (via JNI) for performance-critical tasks.

**Integration with Firebase:** Seamless integration with Firebase for features like real-time databases, analytics, authentication, and cloud messaging.

**Plugin System:** Extend functionality using plugins such as Kotlin plugins, Lint checks, and more.

### **Key Components of Android Studio**

#### **1. Project Structure**

- **Manifest:** Contains essential information about the app (e.g., package name, permissions, components).
- **Java/Kotlin Files:** Contains the business logic of the application.
- **Res (Resources):** Houses XML layouts, images, strings, colors, and styles.
- **Gradle Scripts:** Handles dependencies, build configurations, and project-level settings.

**2. Android Virtual Device (AVD) Manager:** Create and manage virtual devices for testing apps on different Android versions and hardware specifications.

**3. Debugging Tools:** Advanced tools like Logcat for viewing system logs and Breakpoints for analyzing runtime behavior.

**4. Device File Explorer:** Allows viewing, modifying and managing files on an emulator or connected device.

**5. Testing Frameworks:** Built-in support for JUnit (unit tests) and Espresso (UI tests).

### **Advantages of Android Studio**

**1. Official Support:** Backed by Google, it ensures compatibility with the latest Android versions and tools.

**2. Comprehensive Toolset:** Combines all tools needed for Android development in a single environment.

**3. Customizable:** Highly customizable to suit developer workflows through settings.

**4. Official Support:** Backed by Google, it ensures compatibility with the latest Android versions and tools.

5. **Comprehensive Toolset:** Combines all tools needed for Android development in a single environment.
6. **Customizable:** Highly customizable to suit developer workflows through settings and plugins.
7. **Cross-Platform Support:** Available on Windows, macOS, and Linux.
8. **Efficient Workflow:** Tools like instant previews and emulator integration enhance productivity.

#### Use Cases of Android Studio

1. **App Development:** Create native apps for phones, tablets, TVs, wearables, and automotive devices.
2. **Debugging and Testing:** Identify and fix performance bottlenecks and ensure compatibility across devices.
3. **Prototype Development:** Quickly create app prototypes with drag-and-drop layout tools.
4. **Learning Platform:** A great platform for students and beginners to learn Android development. It is an excellent platform for students and beginners to learn Android development. It provides all the essential tools and resources for building Android apps.

## 2.5 HARDWARE REQUIREMENTS

Processor : Intel i7

Processor Speed : 1.9GHz

System Type : 64-Bit Operating System

HDD : 100GB

RAM : 6-GB RAM

## CHAPTER 3

# SYSTEM DESIGN

### 3.1 INTRODUCTION

System design is the process of defining the architecture, components, modules, interfaces and data for a system to satisfy specified requirements. In this process requirements are translated into a representation of software. Initially, the representation depicts a holistic view of software. Subsequent refinement leads to a design representation that is very close to source code. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of system analysis, system architecture and system engineering.

### 3.2 SYSTEM ARCHITECTURE DESIGN

System architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviour of the system.

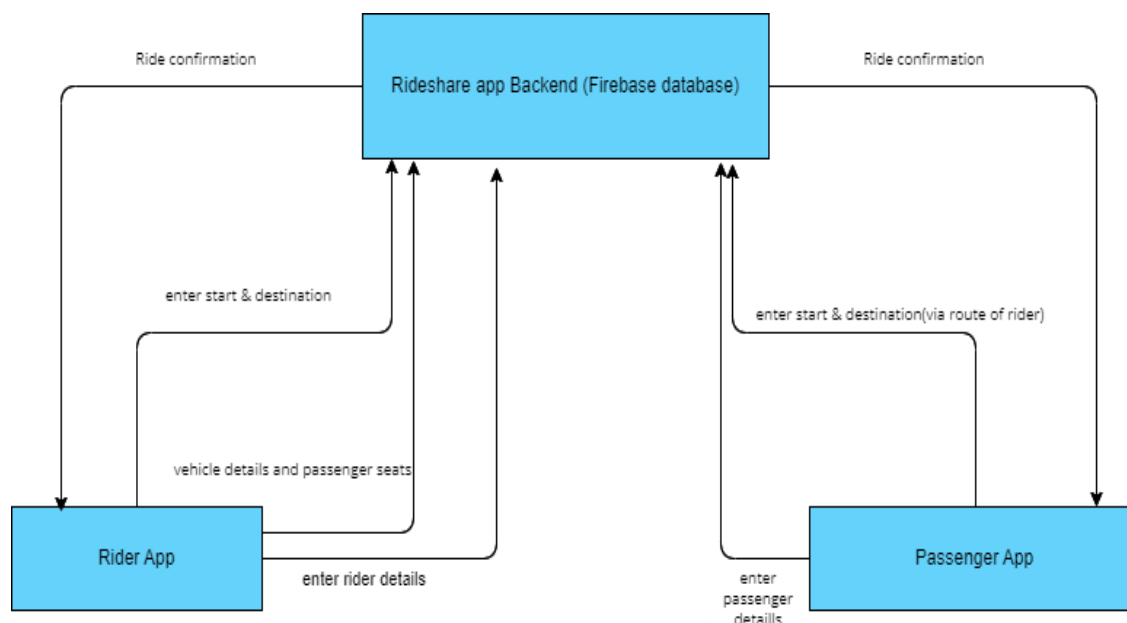


Figure 3.1: Architecture of the System

The diagram represents a System architecture of a rideshare app, showcasing how the Rider App, Passenger App, and the Backend (Firebase Database) interact with each other. Below is a step-by-step explanation of each component and the flows between them.

**Components in the Diagram:**

1. Rider App: The app used by the driver to view requests, enter vehicle details, and confirm ride statuses.
2. Passenger App: The app used by the passenger to book rides, provide journey details, and receive confirmations.
3. Rideshare App Backend (Firebase Database): The server-side system powered by Firebase that manages and processes all ride data, including rider and passenger details, ride statuses, and communication between the two apps. The app used by the passenger to book rides, provide journey details, and receive confirmations.
4. Rideshare App Backend (Firebase Database): The server-side system powered by Firebase that manages and processes all ride data, including rider and passenger details, ride statuses, and communication between the two apps.

**Step-by-Step Explanation of Flows****Passenger App to Backend**

1. Enter Start & Destination:
  - The passenger provides their pickup and drop-off locations in the app.
  - This data is sent to the backend for processing and ride matching.
2. Enter Passenger Details:
  - The passenger inputs personal information such as name, contact details, and payment preferences. These details are stored in the Firebase database.
3. Ride Confirmation:
  - Once a driver (rider) is assigned to the trip, the passenger app receives a notification confirming the ride details .

**Rider App to Backend**

1. Enter Start & Destination:
  - The rider (driver) inputs their current location and availability.
  - This data is updated in the backend to match them with nearby ride requests.
2. Vehicle Details and Passenger Seats:
  - The driver provides information about their vehicle (e.g., car model, license plate) and the number of available seats.

### 3. Ride Confirmation:

- When the driver accepts a ride request, this status is updated in the backend and communicated to both the passenger and rider apps.
- The ride confirmation process is designed to provide a reliable and efficient experience, ensuring that both riders and drivers have the necessary information for a successful journey.

### **Backend to Both Apps:**

1. **Matching Process:** The backend uses the passenger's start/destination and the rider's current location/availability to assign the closest driver to the passenger.
2. **Ride Confirmation:** Once the match is successful, the backend sends confirmation details (driver info for passengers, passenger info for riders) to both apps.
3. **Live Updates:** The backend continuously syncs data between both apps, ensuring real-time updates such as ride status, driver location, and ETA.

### **Key Functionalities Enabled by this Flow:**

- **Real-Time Data Sync:** The Firebase backend ensures instant data synchronization for ride requests, confirmations, and updates.
- **Ride Matching:** The backend matches passengers with the nearest available riders efficiently.
- **Communication:** The backend acts as a central hub to relay information between riders and passengers, ensuring a seamless experience.
- **Scalability:** Firebase's serverless architecture allows the system to scale as user demand increases.

This architecture is designed to facilitate seamless communication between users and the backend of the application. It ensures that all interactions, such as booking rides and managing user data, are handled efficiently. The system is built to support the core functionalities of a rideshare app, providing a smooth and reliable experience. By utilizing optimized protocols and technologies, it allows for real-time updates and responses. This architecture ensures that both user requests and backend processes are executed without delay. It also supports scalable data management, allowing for easy updates and integration of new features.



### 3.3 FLOW CHART

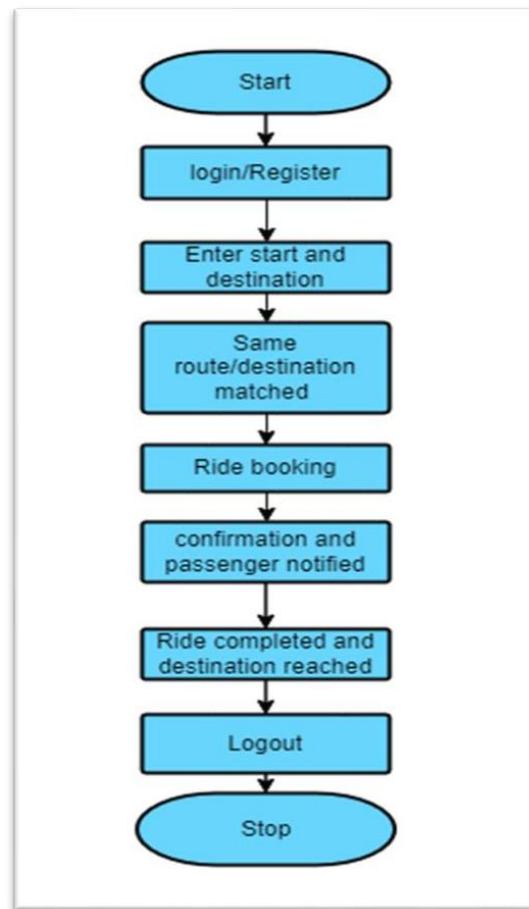


Figure 3.2: Flow Chart Diagram of the Project

**1. Start**

- The application process begins.

**2. Login/Register**

- Users either log in if they have an account or register to create one.

**3. Enter Start and Destination**

- Users input their starting point and destination details.

**4. Same Route/Destination Matched**

- The system checks for a match between the user's route and other available users (riders/passengers)

**5. Ride Booking**

- Once a match is found, the ride is booked.

**6. Confirmation and Passenger Notified**

- Both the rider and the passenger are notified about the booking confirmation.

### 7. Ride Completed and Destination Reached

- The ride is completed upon reaching the destination.

### 8. Logout

- Users can log out of the application.

### 9. Stop

- The process ends.

## 3.4 USE CASE DIAGRAM

A use-case is a set of scenarios that describe an interaction between a source and a destination. The two main components of a use case diagram are use cases and actors. It displays the relationship between them. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

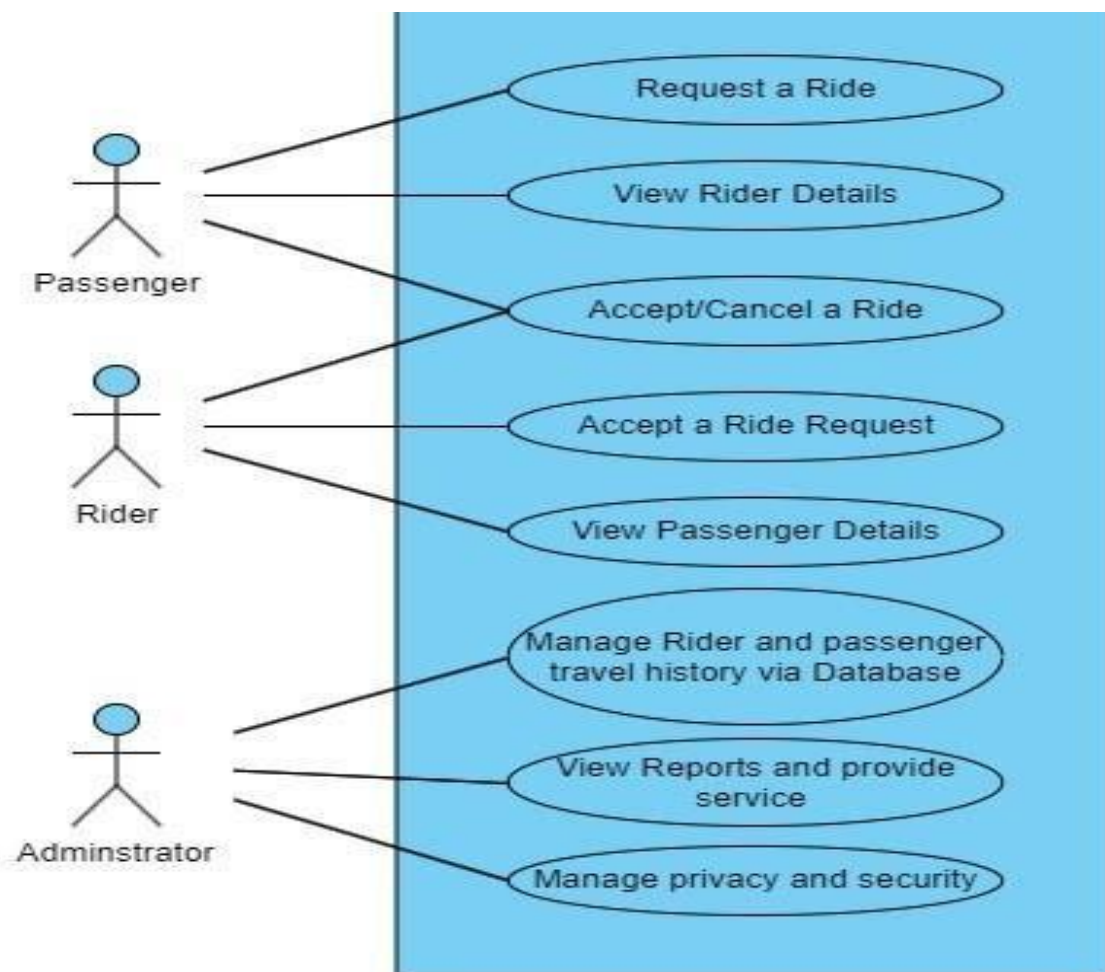


Figure 3.3: Use-Case Diagram

**Passenger Actions:**

- **Request a Ride:**

The passenger uses the system to request transportation by providing details such as pickup and drop-off locations.

- **View Rider Details:**

After a ride request is accepted, the passenger can view details about the assigned rider (e.g., name, contact information, vehicle details).

**Rider Actions**

- **Accept/Cancel a Ride:**

Riders can either accept or decline ride requests based on availability or preferences.

- **View Passenger Details:**

After accepting a ride request, the rider can access passenger information (e.g., name, contact, and pickup location) to facilitate the trip

**Administrator Actions**

- **Manage Rider and Passenger Travel History via Database:** The administrator oversees and maintains the database containing travel history for both riders and passengers to ensure smooth operations and resolve disputes if necessary.
- **View Reports and Provide Service:** The administrator generates and reviews system reports, such as ride statistics, revenue, or user feedback, to enhance the service quality.
- **Manage Privacy and Security:** The administrator ensures that the system complies with privacy and security policies, protecting user data and preventing unauthorized access.

### 3.5 SEQUENCE DIAGRAM

A sequence diagram for a rideshare app visually represents the step-by-step interactions between key entities such as the Passenger, Rider, and the Rideshare System during a ride-booking process. It begins with the Passenger initiating a ride request by providing pickup and drop-off locations. The Rideshare System processes this request, searches for available riders nearby, and sends a notification to the closest Rider. The Rider responds by either accepting or declining the ride. If accepted, the system confirms the match and sends a notification to both the Passenger and the Rider.

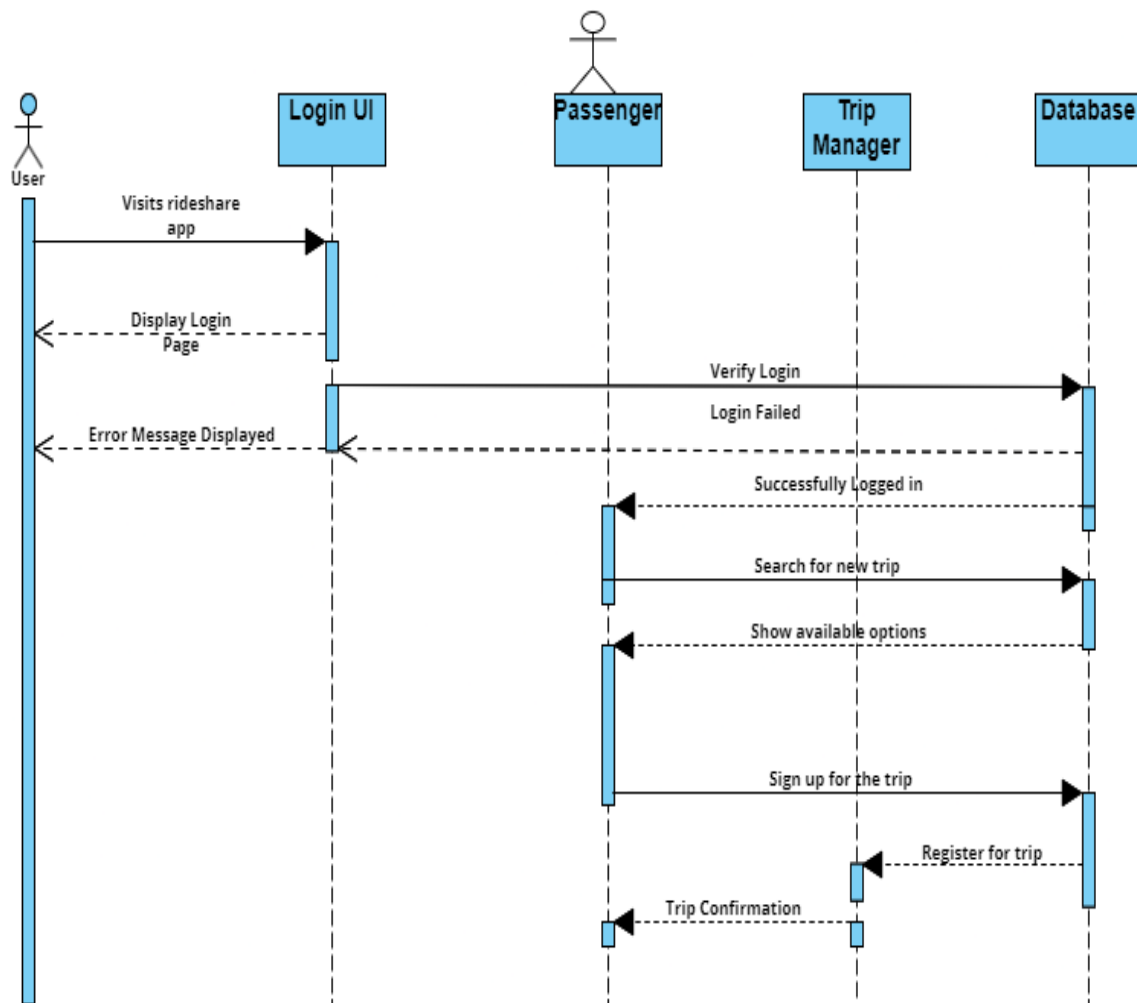


Figure :3.4 illustrates the sequence diagram of the Rider/user.

1. **Login UI:** The user visits the app, which displays the login page. If login credentials are incorrect, an error message is shown.
2. **Successful Login:** Upon providing valid credentials, the Login UI verifies the login details with the database, and the user is successfully logged in.
3. **Trip Manager:** After logging in, the user searches for a new trip, which is handled by the Trip Manager module.
4. **Trip Options:** The Trip Manager retrieves and displays available trip options to the user.
5. **Trip Registration:** The user selects and registers for a trip, which is confirmed by updating the database, and a confirmation message is sent back to the user.

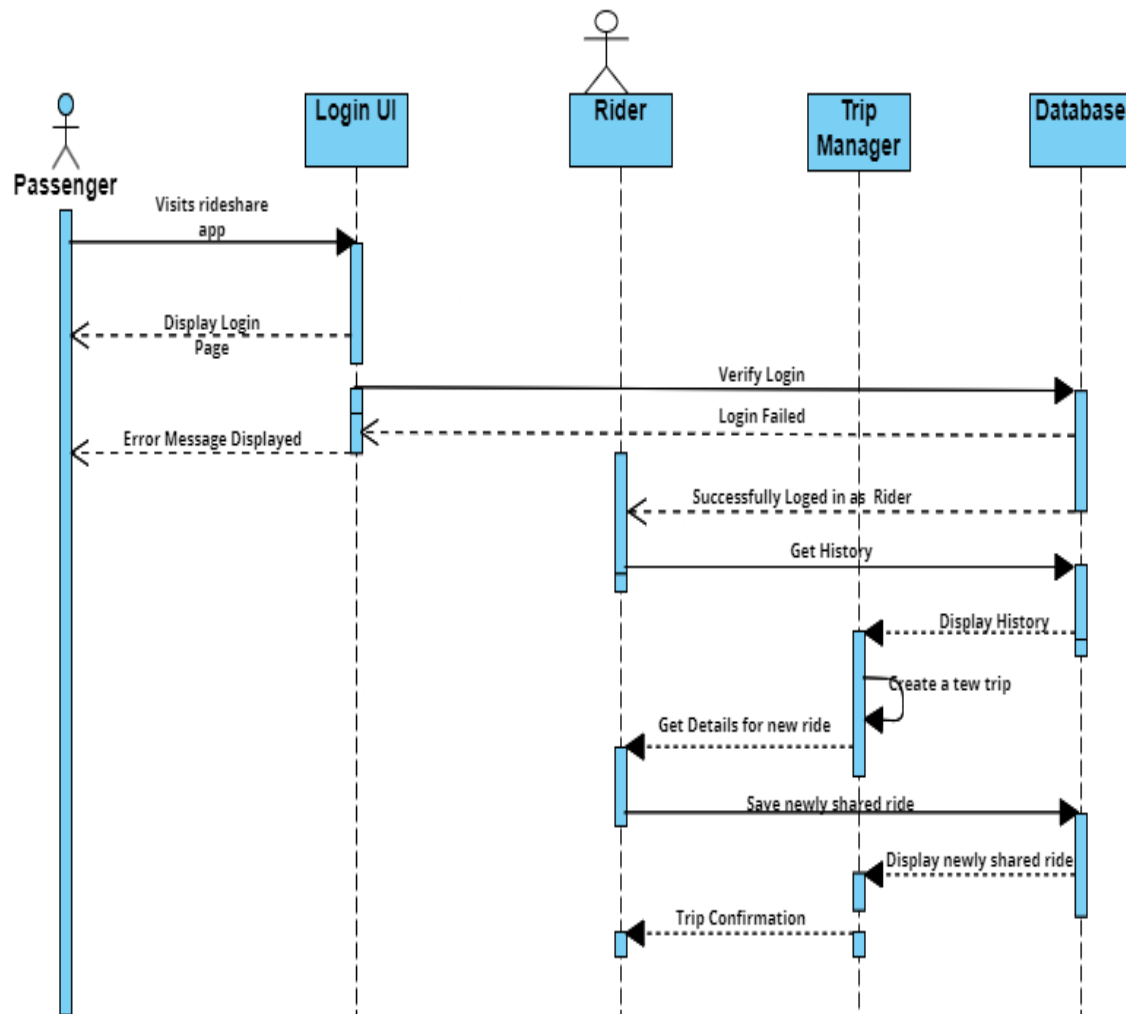


Figure 3.5: illustrates the sequence diagram of the passenger

1. **Login Attempt:** The driver accesses the ridesharing app, and the Login UI displays the login page. If the login fails (due to incorrect credentials), an error message is shown to the driver.
2. **Successful Login:** Upon entering valid credentials, the Login UI verifies the login details with the database, and the driver is successfully logged in.
3. **Trip Availability Check:** After logging in, the driver checks for new trip requests using the Trip Manager module.
4. **Available Trip Requests:** The Trip Manager retrieves trip requests from the database and displays them to the driver.
5. **Trip Assignment:** The driver selects a trip to fulfill, confirms their availability, and the database is updated. A trip confirmation is then sent to the drive.

## 3.6 DATA FLOW DIAGRAM

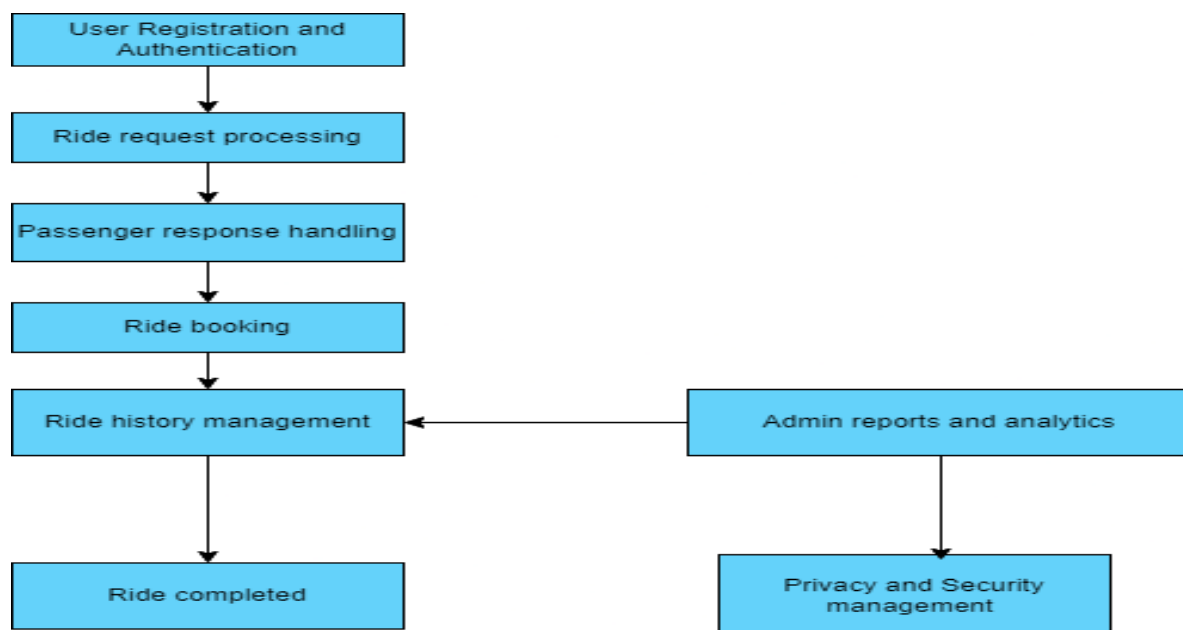


Figure 3.6: Dataflow Diagram of the System.

### 1. User Registration and Authentication

Passenger and rider provide details (e.g., name, email, phone, password, vehicle details for riders). Validates user inputs, encrypts sensitive data (e.g., passwords), stores user data in the database. Authenticates users during login by comparing credentials with stored data. Registration confirmation or login success message.

### 2. Ride Request Processing

Passenger inputs pickup and drop-off locations. Query (string matching) the database to find nearby available riders. Show the passenger the estimated ride details, and confirmation.

### 3. Ride History Management

Completed ride data (e.g., passenger and rider details, trip duration, fare). Store trip data securely in the database. Retrieve past trip data for passengers and riders on request. Use aggregated data for system analytics (e.g., peak hours, popular routes). Viewable trip history for users (passenger and rider apps). Detailed reports for administrator.

### 4. Privacy and Security Management

User data (personal and payment). Encrypt sensitive data during transmission and storage. Monitor system for unauthorized access.

## CHAPTER 4

# SYSTEM IMPLEMENTATION

### 4.1 INTRODUCTION

Building a rideshare app using Android Studio with Java involves several core components, from setting up the project and implementing essential features to integrating external APIs for functionality like mapping, payments, and notifications. Below is a detailed step-by-step guide for implementing a rideshare app:

#### 1. Setting Up the Development Environment

**Install Android Studio:** Download and install Android Studio from the official website: [Android Studio](https://developer.android.com/studio). This IDE will be used to write, test, and build your Android applications.

**Create a New Project:** Open Android Studio and create a new project using the Empty Activity template. Name the project (e.g., Rideshare App), select Java as the language, and choose a minSdk Version that supports the devices you target.

**Install SDKs and Dependencies:** Make sure the required Google Play Services SDK is installed for accessing location services and maps. Add dependencies to build.gradle (Module: app) for libraries like GoogleMaps, Firebase, Stripe, etc.

### 4.2 IMPLEMENTATION APPROACHES

1. **Firestore Authentication Setup:** Firestore simplifies user authentication. First, create a Firestore project and link it to your Android app via the Firestore Console.
  - Enable Email/Password Authentication on Firestore.
  - Add Firestore Authentication to your Android project by configuring it in the Firestore Console and integrating the Firestore SDK.
2. **Registration Activity:** Create an activity (Register Activity) for user registration where the user enters an email, password, and personal details. Use Firestore's authentication methods to register users.

#### 3. Platform Selection

**Decide whether the app will be:**

- **Native:** Separate apps for iOS and Android for better performance and user experience.
- **Cross-Platform:** Using frameworks like Flutter or React Native for faster development and reduced cost.
- **Web-Based:** A progressive web app (PWA) for easier maintenance, though less

common in rideshare.

#### 4. Backend and Database Design

- **Backend Frameworks:** Use robust and scalable frameworks like Node.js, Django, or Ruby on Rails.
- **Database:** Choose based on scalability:
  - **SQL Databases:** MySQL or PostgreSQL for structured data.
  - **NoSQL Databases:** MongoDB for handling real-time data like user requests and location updates.
- Implement real-time communication using WebSockets or Firebase.

#### 5. Key Features and Technologies

##### Real-Time Location Tracking

- Integrate GPS for real-time tracking.
- Use efficient algorithms for route optimization and matching riders with nearby drivers.

##### Ride Matching Algorithm

- Implement a matching system based on:
  - Proximity of drivers.
  - Driver ratings and ride history.

### 4.3 IMPLEMENTATION STEPS

#### Step 1: Define Objectives and Scope

1. **Define Target Audience:**
  - Riders (users looking for transportation).
  - Drivers (individuals providing transportation).
  - Administrators (managing the platform).
2. **List Features:**
  - Core features: User registration, ride booking, real-time tracking, fare estimation.
  - Additional features: Ride history, ratings/reviews, customer support.

#### Step 2: Plan Architecture

1. **Platform Decision:**

Native (iOS, Android) or Cross-Platform (React Native, Flutter). Backend scalability using cloud services (AWS, Azure, Google Cloud).
2. **Tech Stack:**
  - Frontend: React Native, Flutter, or Swift/Kotlin.
  - Backend: Node.js, Python (Django), or Ruby on Rails.



- Database: PostgreSQL, MongoDB for real-time data.
- APIs: Google Maps for geolocation, Twilio for SMS, Stripe for payments.

### **Step 3: Backend Development**

1. **Database Design:** Tables for users, drivers, rides, payments, and feedback.
2. **API Development:** Create REST or GraphQL APIs for frontend-backend communication. Core APIs such as Authentication, ride matching, fare calculation, payment processing.
3. **Real-Time Communication:** Use WebSockets or Firebase for live updates (e.g., ride status, driver location).

### **Step 4: Frontend Development**

1. **User App**
  - User registration and profile management.
  - Search and book rides.
  - Track rides in real time.
2. **Driver App**
  - Driver onboarding and verification.
  - Accept/reject ride requests.
  - Navigate to rider's location.
  - Dynamic pricing and surge pricing during peak hours.
3. **Admin Panel**
  - Manage users and drivers.
  - Monitor rides and resolve disputes.
  - Analyze business metrics.

### **Step 5: Integration of Core Features**

1. **Real-Time Tracking**
  - GPS integration using Google Maps API.
  - Show live driver and rider locations.
2. **Payment System**
  - Integrate gateways (e.g., Stripe, PayPal).
  - Support multiple options: card, wallet, cash.
3. **Notifications**
  - Push notifications for ride requests, status updates.
  - SMS alerts for booking confirmation.

### **Step 6: Implement Ride Matching Algorithm**

1. Match riders with nearby drivers based on Proximity, Availability, Driver ratings and ride history.
2. Optimize for efficiency using algorithms like Dijkstra's or A\* for route mapping.

**Step 7: Testing**

1. **Unit Testing:** Test individual modules (e.g., booking, payment).
2. **Integration Testing:** Verify seamless operation between backend and frontend.
3. **Performance Testing:** Simulate heavy traffic to test app performance.
4. **Beta Testing:** Release to a small group for real-world feedback.

**Step 8: Deployment**

1. Deploy backend services to the cloud (e.g., AWS EC2, Google Cloud).
2. Publish mobile apps on Google Play Store (Android).Apple App Store (iOS).
3. Set up CI/CD pipelines for continuous integration and updates.

**Step 9: Maintenance and Scaling**

1. **Monitor Performance:** Use tools like Google Analytics, New Relic for insights.
2. **Gather Feedback :** Continuously improve based on user and driver feedback.
3. **Scale Infrastructure:** Add servers and optimize the database to handle growing users.
- 4.**Implement Advanced Features:** Carpooling, Subscription plans. Electric vehicle- specific rides.

## CHAPTER 5

# SYSTEM TESTING

System testing is a vital phase in the software development lifecycle, aimed at verifying that the entire application meets its specified requirements and functions seamlessly in real-world scenarios. For the Ride Share App, system testing ensures smooth integration and proper functionality of features like user registration, ride matching, booking and notifications. The goal is to confirm that all components operate cohesively, providing users with a reliable, secure, and user-friendly experience. This phase identifies bugs or inconsistencies, ensuring the app is prepared for deployment in live environments.

### 5.1 TEST OBJECTIVES

The main objectives of testing the Ride Share App include:

- **Functional Validation:** Ensure the correct functioning of critical features, including user role selection, source and destination entry, ride matching, booking, and ride history.
- **Data Integrity:** Validate the accuracy of user data, ride details, and the proper saving and retrieval of information from the Firebase Realtime Database.
- **Security:** Ensure sensitive user data such as personal details and ride information are handled securely.
- **Usability:** Assess the app's interface to guarantee a seamless user experience for both riders and passengers.
- **Performance:** Verify the app's efficiency under varying load conditions, such as multiple users simultaneously matching and booking rides.
- **Real-Time Updates:** Validate the real-time functionality of notifications and booking updates.

### 5.2 FEATURES TO BE TESTED

The Ride Share App includes several key features requiring thorough testing:

- **User Role Selection:** Validate the proper functioning of the rider and passenger selection interface.
- **Source and Destination Entry:** Ensure users can accurately input and save their source, destination, and via points.
- **Ride Matching:** Test the matching algorithm to confirm users are paired with others traveling to the same destination.
- **Booking and Confirmation:** Verify that rides can be booked, confirmed, and stored in the database correctly.

- **Ride History:** Ensure completed rides are accurately saved and accessible in the Firebase Realtime Database.
- **Real-Time Notifications:** Test the app's ability to send booking confirmations, alerts, and reminders.
- **Firebase Integration:** Validate the storage and retrieval of ride and user data from Firebase Realtime Database.

## 5.3 TYPES OF TESTS

Testing the Ride Share App is essential to ensure that all functionalities work as intended, interactions between modules are seamless, and the app provides a secure, reliable, and user-friendly experience. The following types of tests were conducted:

### 5.3.1 Unit Testing

- Unit testing focused on individual components to ensure their proper functionality:
- **Authentication Module:** Confirmed that users (riders and passengers) could log in and were routed to the appropriate interfaces for entering trip details or viewing matched rides.
- **Ride Matching Module:** Tested the algorithm for matching riders and passengers based on their entered destinations and via points.
- **Booking Module:** Verified that users could book a ride from the matched list and that the ride details were stored correctly.

### 5.3.2 Integration Testing

Integration testing validated interactions between different components:

- **Authentication and Role Selection:** Ensured that users were redirected to the correct screens based on their role (rider or passenger).
- **Trip Details and Ride Matching:** Verified that source, destination, and via points entered by users were correctly passed to the matching system.
- **Booking Workflow:** Confirmed that booked rides were updated in both the matched rides list and ride history seamlessly.
- **Firebase Database Integration:** Validated that data entered by users (e.g., trip details, matched rides, and bookings) was accurately stored and retrieved.

### 5.3.3 Functional Testing

Functional testing focused on validating the core operations of the app:

- **Input Validation:** Ensured that all input fields, such as starting point, destination, and via point, were validated for correctness and completeness.
- **Role-Based Navigation:** Verified that users were directed to appropriate interfaces

based on their role (rider or passenger).

- **Matching Algorithm:** Tested the accuracy of the algorithm in pairing users traveling to the same destination.
- **Booking Process:** Confirmed that users could successfully book rides and receive confirmation notifications.

#### 5.3.4 System Testing

System testing ensured that the app worked as a cohesive system:

- **Rider's Flow:** Verified that riders could log in, enter trip details, view matched passengers, and book rides.
- **Passenger's Flow:** Confirmed that passengers could log in, enter trip details, view matched riders, and book rides.
- **Notifications:** Ensured that real-time notifications for booking confirmations, ride status, and ride completion were triggered appropriately.
- **Database Consistency:** Validated that user data, trip details, and bookings were stored and retrieved accurately from Firebase.

#### 5.3.5 Acceptance Testing

Acceptance testing was conducted with potential end-users (riders and passengers) to ensure the system's usability and functionality:

- **Riders:** Confirmed that riders could successfully input trip details, view matched passengers, book rides, and see completed rides in the history section.
- **Passengers:** Verified that passengers could input their trip details, view matched riders, and book rides without issues.
- **User Feedback:** Collected feedback on the app's design, usability, and responsiveness to identify areas for improvement.

## 5.4 Test case to be satisfied

**Table 5.1: Testcase**

<b>Test Case</b>	<b>Expected Result</b>	<b>Status</b>
<b>User Registration</b>	User can sign up with valid credentials.	PASS
<b>Login Authentication</b>	Valid credentials allow access to user profile.	PASS
<b>User Profile</b>	Adding details like name, phone no, gender.	PASS
<b>Role Selection</b>	User will select rider or passenger.	PASS
<b>Source and Destination</b>	User can enter their source and destination.	PASS
<b>Ride Booking</b>	Passenger can book a ride according to rider availability.	PASS
<b>User Notification</b>	Rider will get a passenger notification.	PASS
<b>Logout</b>	Users can logout their account.	PASS

## CHAPTER 6

### SCREENSHOTS

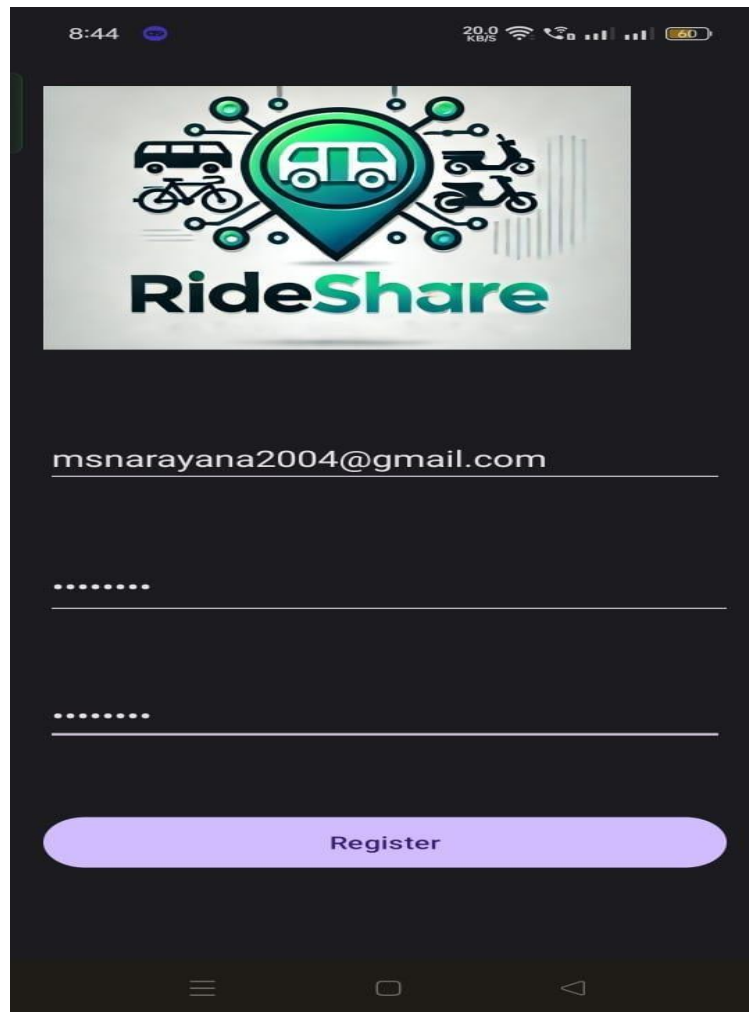


Figure 6.1: Login page

Here the user registers his profile by giving his mail and creating a password which can be used for further logins.

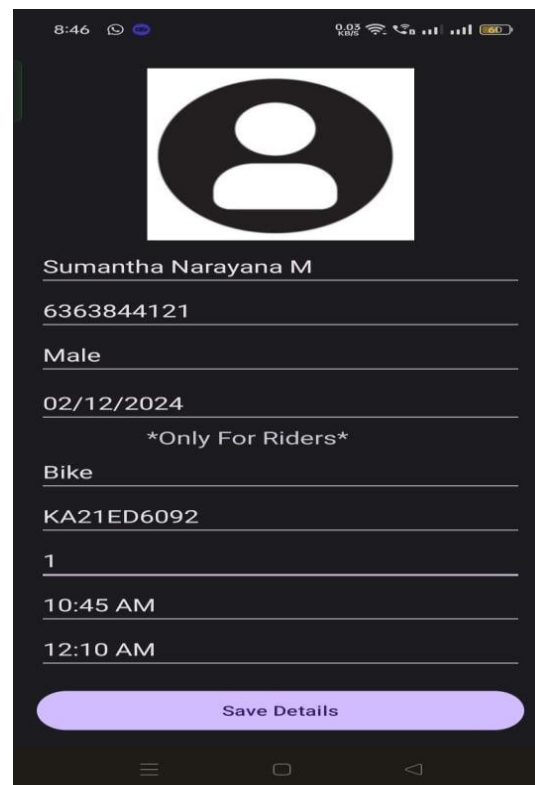
A mobile app screenshot showing a login form for a rider or passenger. The form has a dark background with white text and input fields. At the top, there's a status bar with the time 8:46, signal strength, and battery level. Below the status bar is a large white circular icon with a black person silhouette. The form fields are: Name (Sumantha Narayana M), Phone Number (6363844121), Gender (Male), Date of Birth (02/12/2024), a note '\*Only For Riders\*', Vehicle Type (Bike), Vehicle Number (KA21ED6092), Number of Passengers (1), and two time slots (10:45 AM and 12:10 AM). A green 'Save Details' button is at the bottom.

Figure 6.2: Rider/Passenger login details

In the figure 6.2, user enters his details and if he is a driver there are some mandatory details he has to fill i.e driver has to fill vehicle details also.

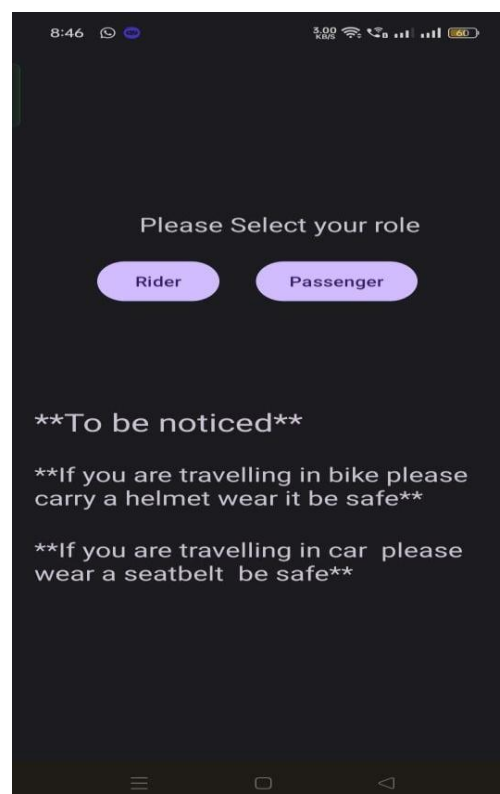
A mobile app screenshot showing a screen where the user selects their role. The screen has a dark background with white text. At the top, there's a status bar with the time 8:46, signal strength, and battery level. Below the status bar is the text 'Please Select your role'. There are two green buttons: 'Rider' and 'Passenger'. Below the buttons is a section titled '\*\*To be noticed\*\*' with two lines of text: '\*\*If you are travelling in bike please carry a helmet wear it be safe\*\*' and '\*\*If you are travelling in car please wear a seatbelt be safe\*\*'. The screen ends with a dark footer bar.

Figure 6.3: User selecting Role

In the figure 6.3, the user has to choose whether he is rider or passenger.



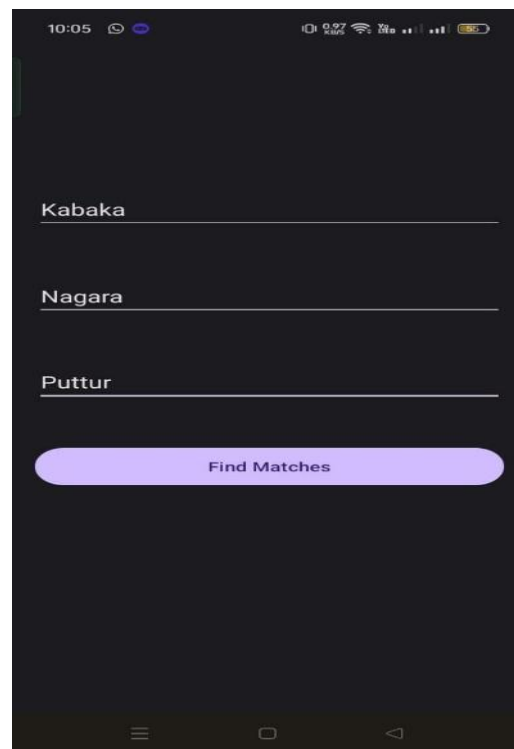


Figure 6.4: User entering destination and via route  
Here the user has to enter his starting point, via route and destination place.

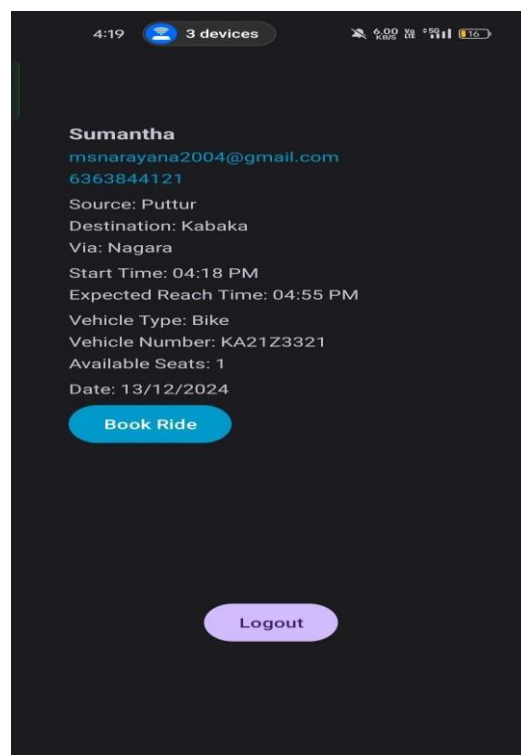


Figure 6.5: Ride Details saved

Here the passenger will get the information of rides in that route, and they can book the ride by clicking the Book Ride option.

## **CHAPTER 7**

# **CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT**

### **7.1 CONCLUSION**

In conclusion, our rideshare app leverages modern technologies like Firebase and Android Studio with Java to deliver a seamless, user-friendly experience for passengers and drivers. The app integrates core functionalities such as user authentication of ride requests, driver matching, and secure storage of ride history. With a scalable backend powered by Firebase and real-time data synchronization, the platform ensures efficient communication between users, accurate ride tracking, and prompt updates. By addressing the essential needs of a rideshare ecosystem, the app not only provides convenience and reliability but also lays a solid foundation for future enhancements and features, ensuring its adaptability in a competitive market.

### **7.2 SCOPE OF FUTURE ENHANCEMENT**

Our rideshare app provides a robust foundation with essential features, but there is significant potential for future enhancements to improve functionality, user experience, and market competitiveness. Dynamic Pricing, Introduce surge pricing based on demand, traffic, and weather conditions to maximize revenue during peak hours. Carpooling Services, Allow users to share rides with others traveling on similar routes, reducing costs and promoting eco-friendly travel. Pre-scheduled Rides, Enable users to schedule rides in advance for specific dates and times.

## REFERENCES

- [1] M Andersson et.al “Peer-to-Peer Service Sharing Platforms:Driving Share and Share Alike on a Mass-Scale”, International Conference of Information Systems, vol.49, no.4, pp.1275-1288, 2013.
- [2] Nusrat Jahan Farin et.al “A Framework for Dynamic Vehicle Pooling and Ride-Sharing System”, 2016 International Workshop on Computational Intelligence (IWCI) , vol.11, no.19, pp.32089-32104, 2016.
- [3] NJ Farin et.al "A Framework for Dynamic Vehicle Pooling and Ride-Sharing System", 2016 International Workshop on Computational Intelligence (IWCI) ,vol.21, pp.1-5, 2016.
- [4]C Lee et.al “Dynamics of Ride-Sharing Competition”, ISEAS Economics Working Paper No. 2017-05, 2017.
- [5]Shaheen et.al “Shared Mobility: The Potential of Ride Hailing and Pooling”, International Workshop on Computational Intelligence (IWCI), vol.5,Pp 120-132, 2018.
- [6] A Tafreshian et.al “Frontiers in Service Science: Ride Matching for Peer-to-Peer Ride Sharing: A Review and Future Directions”, Service Science, vol. 12, No.2-3, 2020.
- [7] Y Sun et.al “Nonprofit Peer-to-Peer Ridesharing Optimization Summary”, Transportation Research Part E: Logistics and Transportation Review, vol. 114, pp. 532-553, 2020.
- [8] A Tafreshian et.al “Trip-based Graph Partitioning in Dynamic Ridesharing”, Transportation Research Part C: Emerging Technologies, vol. 114, Pp. 532-553, 2020
- [9] Amirmahdi Tafreshian et.al “Trip-based graph partitioning in dynamic ridesharing”, Transportation Research Part C: Emerging Technologies, vol. 114, pp. 532-553, 2020
- [10] Piyush Agrawal et.al “SURVEY ON PEER-TO-PEER RIDE SHARING FOR “POOL” A RIDE SHARING APP”, International Journal of Engineering Applied Sciences and Technology, vol. 5, Issue 8, ISSN No. 2455-2143, pp. 180-189, 2020.
- [11] L Mitropoulos et.al “A systematic literature review of ride- sharing platforms, user factors and barriers”, European Transport Research Review, Volume 13, article number 61, (2021).
- [12] Barnali Gupta Banik et.al “Implementation of a Secure Ride-Sharing DApp Using Smart Contracts on Ethereum Blockchain”, International Journal of Safety and Security Engineering Vol. 11, No. 2, pp. 167-173 April, 2021.

- [13] A Soltani et.al “Ridesharing in Adelaide Segmentation of Users”, Journal of Transport Geography, vol.92, pp. 180-189, 2021.
- [14] LC Martins et.al “Optimizing Ride-sharing in Smart Sustainable Cities”, Computer and industrial engineering, vol. 142,2021.
- [15] S Jang et.al “The Effect of Quality Cues on Travelers' Demand forPeer-to-Peer Ridesharing”, Journal of Travel Research, vol. 60, 2021.
- [16] Lambros Mitropoulos et.al “A systematicliterature review of ride-sharing platforms, user factors and barriers”, European Transport Research Review, vol. 13, pp. 6, 2021.
- [17] Sathya A Renu et.al “Implementation of a Secure Ride-Sharing DAppUsing Smart Contracts on Ethereum Blockchain”, International Journal of Safety and Security Engineering, vol. 11, No. 2, pp. 167-173 2021.
- [18] Leandro do C. Martins et.al “Optimizing ride-sharing operations in smart sustainable cities: Challenges and the need for agile algorithms”, Computers and industrial engineering, vol.153, 2021.
- [19] Ajinkya Ghorpade et.al “POOL: A PEER-TO-PEER RIDE SHARING APP”, International Journal of Engineering Applied Sciences and Technology, vol. 5, Issue 12, ISSN No. 2455-2143, pp. 268-274, 2021.
- [20] S Abdikerimova et.al “Peer-to-Peer Multi-Risk Insurance and Mutual Aid”, IEEE Access, vol.10, pp.119193-119205, 2022.