**RIDE SHARE**

# A FULL STACK PROJECT REPORT

# Submitted by

# DHARANEESH R

**(23ITR026)**

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

***of***

**BACHELOR OF TECHNOLOGY IN**

**INFORMATION TECHNOLOGY**

# DEPARTMENT OF INFORMATION TECHNOLOGY

****

**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

# PERUNDURAI ERODE – 638 060

**NOVEMBER 2025**

# DEPARTMENT OF INFORMATION TECHNOLOGY

### KONGU ENGINEERING COLLEGE

**(Autonomous) PERUNDURAI, ERODE – 638060**

**NOVEMBER 2025**

# BONAFIED CERTIFICATE

This is to certify that the Project report entitled **RIDE SHARE** is the bonafide record of project work done by **DHARANEESH R(23ITR026),** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in **INFORMATION TECHNOLOGY** of Anna University, Chennai during the year 2025-2026.

### SUPERVISOR HEAD OF THE DEPARTMENT

#### (Signature with seal)

Date:

Submitted for the end semester viva voce examination held on .

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# DEPARTMENT OF INFORMATION TECHNOLOGY

### KONGU ENGINEERING COLLEGE

**(Autonomous) PERUNDURAI, ERODE – 638060**

**NOVEMBER 2025**

# DECLARATION

We affirm that the Project Report titled **RIDE SHARE** being submitted in partial fulfilment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

# Date: DHARANEESH R

**(23ITR026)**

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Name and Signature of the Supervisor with seal

# ABSTRACT

# The Ride Share Platform is a full-stack MERN application designed to provide a seamless, real-world carpooling and ride-sharing experience for riders and drivers. The React.js frontend delivers an intuitive, responsive interface that enables users to browse available rides, create new rides, apply filters, view ride details, and manage their bookings and history. Users can log in securely, access personalized dashboards, and interact with role-based features such as driver ride creation or rider ride booking.The backend, built with Node.js and Express.js, exposes a set of secure, JWT-based REST APIs responsible for authentication, ride management, booking workflows, and admin-level operations. MongoDB serves as the database layer, storing all ride, user, and booking data with efficient indexing for quick search, filtering, and status-based queries. The system enforces clear status transitions—available, booked, completed, and cancelled—ensuring consistent business logic across users and admin operations.To maintain ride integrity, the booking engine validates user roles, ensures live ride availability, updates booking states atomically, and prevents double-booking conditions. Admin users have a dedicated panel to monitor system-wide rides, execute filtered searches, and oversee operations. With modular architecture, strong role-based access control, and a clean separation of frontend-backend concerns, the platform is optimized for scalability, maintainability, and real-world expansion into production-grade ride-sharing environments.

# ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of Almighty throughout different phases of the project and its successful completion.

I wish to express my gratefulness to our beloved Correspondent **THIRU. E. R. K. KRISHNAN, M.Com.** and all the trust members of Kongu Vellalar Institute of Technology Trust for providing all the necessary facilities to complete the project successfully.

I express my deep sense of gratitude to our beloved Principal **Dr.R.PARAMESHWARAN M.E., Ph.D.,** for providing an opportunity to complete the project.

I express my gratitude to **Dr. S. ANANDAMURUGAN M.E., Ph.D.,** Head of the Department, Department of Information Technology for his valuable suggestions.

I am thankful to my Project Coordinators **Dr. E.M.ROOPA DEVI ME., PhD., Ms. S.SUJITHA M.Tech., and Ms.G.SASIKALA M.E.,** for the valuable guidance and support to complete our project successfully.

I am highly indebted to **Mrs. S.KRISHNAVENI ME.,** Department of Information Technology for her valuable supervision and advice for the fruitful completion of the project.

I am thankful to the faculty members of the Department of Information Technology for their valuable guidance and support.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER No** | **TITLE** | **PAGE No** |
|  | **ABSTRACT** | **iv** |
|  | **LIST OF FIGURES** | **viii** |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 INTRODUCTION | 1 |
|  | 1.2 OBJECTIVE | 1 |
| **2.** | **SYSTEM SPECIFICATION** | **2** |
|  | 2.1 HARDWARE REQUIREMENTS | 2 |
|  | 2.2 SOFTWARE REQUIREMENTS | 2 |
|  | 2.3 SOFTWARE DESCRIPTION | 3 |
|  | 2.3.1 Visual Studio Code | 3 |
|  | 2.3.2 NodeJS | 3 |
|  | 2.3.3 MongoDB | 5 |
|  | 2.3.4 ReactJS | 5 |
| **3.** | **SYSTEM DESIGN** | **9** |
|  | 3.1 USE CASE DIAGRAM | 9 |
|  | 3.2 CLASS DIAGRAM | 10 |
|  | 3.3 SEQUENCE DIAGRAM | 11 |
|  | 3.4 ACTIVITY DIAGRAM | 12 |
|  | 3.5 DATABASE DESIGN | 13 |

|  |  |  |
| --- | --- | --- |
|  | 3.6 [MODULES DESCRIPTION](#_bookmark0) | 14 |
|  | 3.6.1 Authentication Module | 14 |
|  | 3.6.2 Ride and Schedule Management Module | 14 |
|  | 3.6.3 Booking Management Module | 15 |
|  | 3.6.4 Notification Module | 15 |
|  | * + 1. Transaction and log Module     2. Reporting & Analytics Module     3. Availability Page Module | 15  16  16 |
| **4.** | **SYSTEM TESTING** | **17** |
|  | 4.1 INTRODUCTION | 17 |
|  | 4.2 [UNIT TESTING](#_bookmark1) | 17 |
|  | 4.3 MODULE TESTING | 18 |
|  | 4.4 INTEGRATION TESTING | 19 |
|  | 4.5 [VALIDATION TESTING](#_bookmark2) | 20 |
| **5.** | **RESULTS** | **21** |
| **6.** | **CONCLUSION AND FUTURE WORK** | **22** |
|  | [**APPENDIX 1- CODING**](#_bookmark3) | **23** |
|  | **APPENDIX 2- SNAPSHOTS** | **39** |
|  | **REFERENCES** | **43** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE No.** | **FIGURE NAME** | **PAGE No.** |
| 3.1 | USE CASE DIAGRAM | 9 |
| 3.2 | CLASS DIAGRAM | 10 |
| 3.3 | SEQUENCE DIAGRAM | 11 |
| 3.4 | ACTIVITY DIAGRAM | 12 |
| A2.1 | HOME PAGE | 42 |
| A2.2 | AVILABEL PAGE | 42 |
| A2.3 | BOOKRIDE PAGE | 43 |
| A2.4 | RIDE PAGE | 43 |
| A2.5 | RIDE CREATE PAGE | 44 |
| A2.6 | HISTORY PAGE | 44 |
| A2.7 A2.8 | USER DATABASE RIDER DATABASE | 45  45 |

## CHAPTER 1

### INTRODUCTION

* 1. **INTRODUCTION**

The Full-Stack Ride Share Platform is a modern, scalable MERN-stack application (MongoDB, Express.js, React.js, and Node.js) built to simplify and digitalize everyday carpooling and ride-sharing workflows. The system uses secure, JWT-based authentication to deliver a role-specific experience for riders, drivers, and administrators. Riders can search, filter, and book available rides in real time, while drivers can create rides, manage schedules, complete trips, and track bookings from an intuitive interface.

### OBJECTIVE

### The objective of the Full-Stack Ride Share Platform is to develop a secure, scalable, and fully functional web application that simplifies daily commute and carpooling operations for both riders and drivers. The system aims to streamline the processes of ride creation, discovery, real-time availability checking, and booking confirmation, providing users with a seamless and intuitive ride-sharing experience. Additionally, it equips administrators with a powerful, role-based dashboard to monitor rides, manage operational statuses, and oversee platform-wide activities.

## CHAPTER 2 SYSTEM SPECIFICATION

* 1. **HARDWARE SPECIFICATION**

**Processor :** Ryzen 7 5800HS

**Processor Speed :** 1.70 GHz

**RAM :** 16GB RAM

**Hard Disk :** 512 GB

**Keyboard :** Standard 104 enhanced

**Mouse :** Local PS/2

## SOFTWARE REQUIREMENTS

**Platform** : Visual Studio Code

**Server-Side Script** : NodeJS

**Database** : MongoDB

**Library** : ReactJS

# SOFTWARE DESCRIPTION

# 2.3.1Visual Studio Code

Visual Studio Code is a versatile and user-friendly code editor used by developers worldwide. It supports a wide range of programming languages and frameworks, including JavaScript, React, and Node.js. For this project, VS Code provides a convenient environment for writing, editing and debugging code related to the server-side scripting with Node.js, as well as client- side scripting with React.js. Its built-in terminal and debugging tools streamline the development process, while its customizable features allow developers to tailor the editor to their specific needs, enhancing productivity and efficiency throughout the project lifecycle.

#### ES6+ React/Redux/React-Native

ES6 React/Redux/React-Native snippets provide code templates designed to simplify and accelerate the development process for building applications with React, Redux and React Native. These snippets utilize the powerful features introduced in ES6, allowing developers to quickly scaffold common patterns like React components, Redux actions, reducers and React Native components. By incorporating ES6 syntax, such as arrow functions, destructuring, template literals and modules, these snippets make it easier to write clean and concise code. With ES6 snippets, developers can efficiently generate standardized code structures that promote consistency and reduce the chance of errors, cutting down on boilerplate code. This collection of snippets supports the creation of both web applications in React and cross- platform mobile applications in React Native, while simplifying state management in Redux. Whether developing user interfaces, managing state or working on cross-platform features, these ES6 snippets help developers stay focused on creating features and solving problems instead of repeating boilerplate setup.

## 2.4 NodeJS

Node.js is a powerful server-side JavaScript runtime environment commonly used for building scalable and efficient web applications. In our project, Node.js serves as the server- side scripting language, handling tasks such as routing, handling HTTP requests and interacting with the MongoDB database. Its event-driven architecture and non-blocking I/O

operations make it well-suited for handling concurrent connections and processing requestsefficiently. With its vast ecosystem of libraries and frameworks, Node.js allows for rapid development and deployment of server-side applications. Additionally, its compatibility with Visual Studio Code provides a seamless development experience, enabling developers to write, test, and debug server-side code effectively within a unified environment. Overall, Node.js plays a crucial role in this project by powering the backend infrastructure and facilitating the seamless integration of frontend and backend components.

#### 2.4.1 Bcryptjs

Bcryptjs is a library used for hashing passwords in JavaScript. It provides a secure way to hash passwords before storing them in a database or comparing them during authentication processes. By using bcryptjs, developers can ensure that user passwords are encrypted and protected against common security threats like brute force attacks and rainbow table attacks.

#### Express

Express is a minimalist web application framework for Node.js. It provides a robust set of features for building web servers and APIs, including routing, middleware support, template engines and HTTP utilities. With its simple and flexible design, Express enables developers to quickly create scalable and maintainable web applications.

#### Jsonwebtoken:

JSON Web Token is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. JWTs are commonly used for authentication and authorization in web applications. The jsonwebtoken library provides utilities for generating, parsing and verifying JWTs, making it easy to implement token-based authentication and authorization systems.

#### Nodemon:

Nodemon is a utility that monitors for changes in Node.js applications and automatically restarts the server when changes are detected. It eliminates the need to manually stop and restart the server every time a code change is made, improving developer productivity and workflow efficiency. Nodemon supports features like watching for file changes, ignoring

specific files or directories and running custom scripts before and after restarting the server, making it an essential tool for Node.js development.

#### MongoDB

#### MongoDB is the ideal choice for the Hotel Booking Application due to its flexible and horizontally scalable NoSQL architecture. Its document-oriented model efficiently handles diverse data structures, including dynamic hotel/room details, booking records, and user profiles. The schema-less nature allows the system to easily accommodate evolving data models (e.g., adding new amenities) without complex migrations, which is perfect for the hospitality domain. MongoDB's inherent scalability ensures the platform maintains optimal performance even as transaction volumes and user data increase. Furthermore, its advanced querying and indexing capabilities enable fast, complex searches (e.g., filtering by price, city, and availability), and features like high availability and automatic failover guarantee the booking platform remains continuously accessible and reliable for both travelers and administrators.

#### 2.4.6 ReactJS

React.js is a widely-used JavaScript library for building dynamic and interactive user interfaces. In the Hotel Booking project, React enables the development of responsive and reusable UI components for dashboards, task lists, team collaboration views, notifications, and activity feeds. Its component-based architecture allows developers to create modular and maintainable code, simplifying the management of complex interfaces.

React’s virtual DOM optimizes performance by updating only the parts of the interface that change, ensuring a smooth and fast user experience even when handling multiple hotels, rooms, and bookings simultaneously. Additionally, React integrates seamlessly with Redux for state management, allowing consistent and efficient handling of

application-wide data such as hotel registering and room bookings.

The extensive React ecosystem provides numerous libraries and tools to enhance development, including routing, API handling, and UI components. Overall, React plays a crucial role in HOTEL-APP by enabling an interactive, high-performance, and user-friendly interface that supports Hotel Registering, Booking and so on.

2.4.6.1 **axios:**

Axios is a promise-based HTTP client used to communicate between the frontend and backend. In HOTEL APP, Axios handles API requests for CRUD operations on Adding ,Updating and deleting process ensuring seamless interaction with the server.

#### 2.4.6.2 react-dom:

React DOM is responsible for rendering React components into the browser’s Document Object Model (DOM) and updating the interface when component state changes. It ensures that task lists, dashboards, and activity feeds are displayed correctly and respond dynamically to user actions.

**2.3.3.3 react-redux:**

React Redux provides bindings for integrating Redux with React. In HOTEL-APP, it manages the global state efficiently, keeping track of room bookings across the application. This ensures consistency and smooth updates throughout all components.

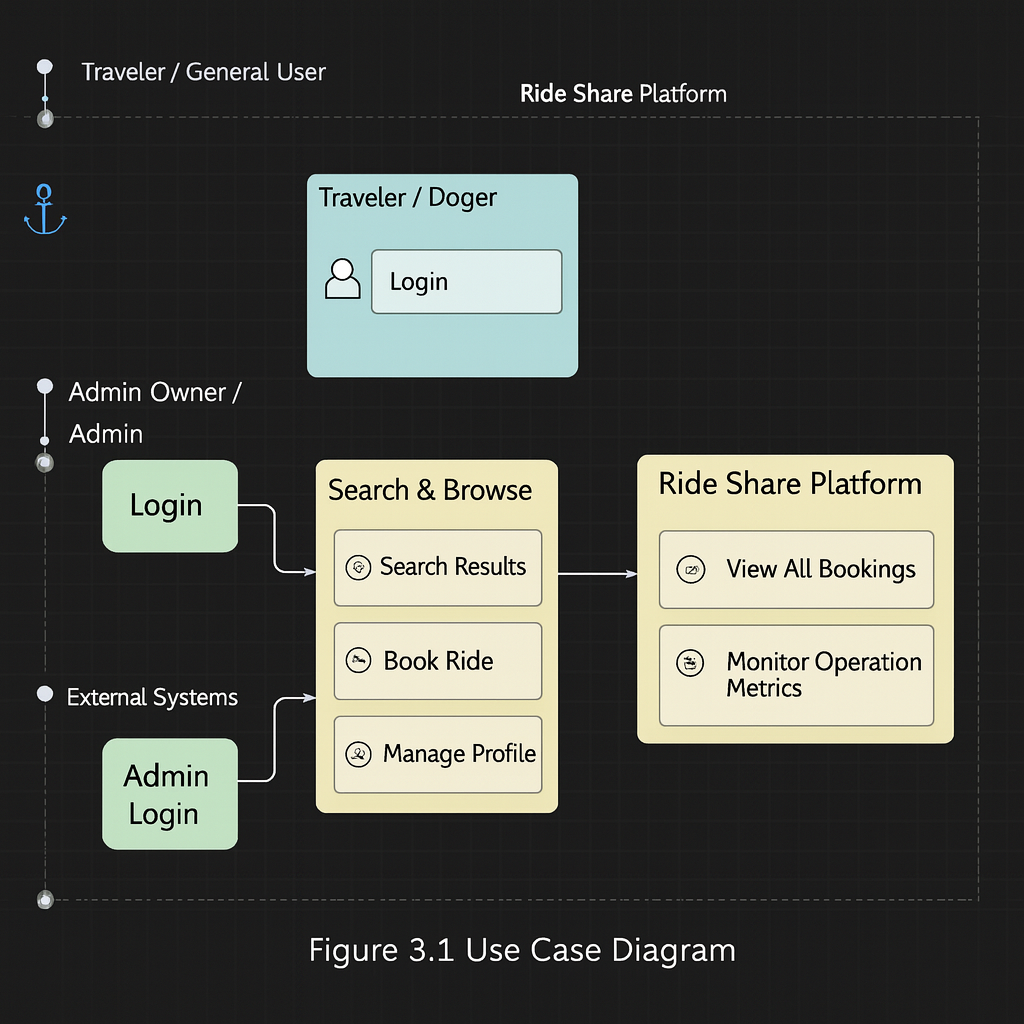
#### 2.3.3.4 react-router-dom

React Router DOM enables declarative client-side routing, allowing HOTEL-APP to have multiple views such as bookings, List yout hotel, dashboard and saving without fail when page reloads. It ensures smooth navigation and enhances the overall user experience.

# CHAPTER 3 SYSTEM DESIGN

* 1. **USE CASE DESIGN**

A use case diagram shows how different actors (users) interact with a system and the actions the system performs. Actors are the users, and use cases are the tasks they can perform. For the **RIDE SHARE** , it shows users searching for rides, checking availability, creating bookings, making payments, and **administrators managing rides and schedules**, as illustrated in Figure 3.1.

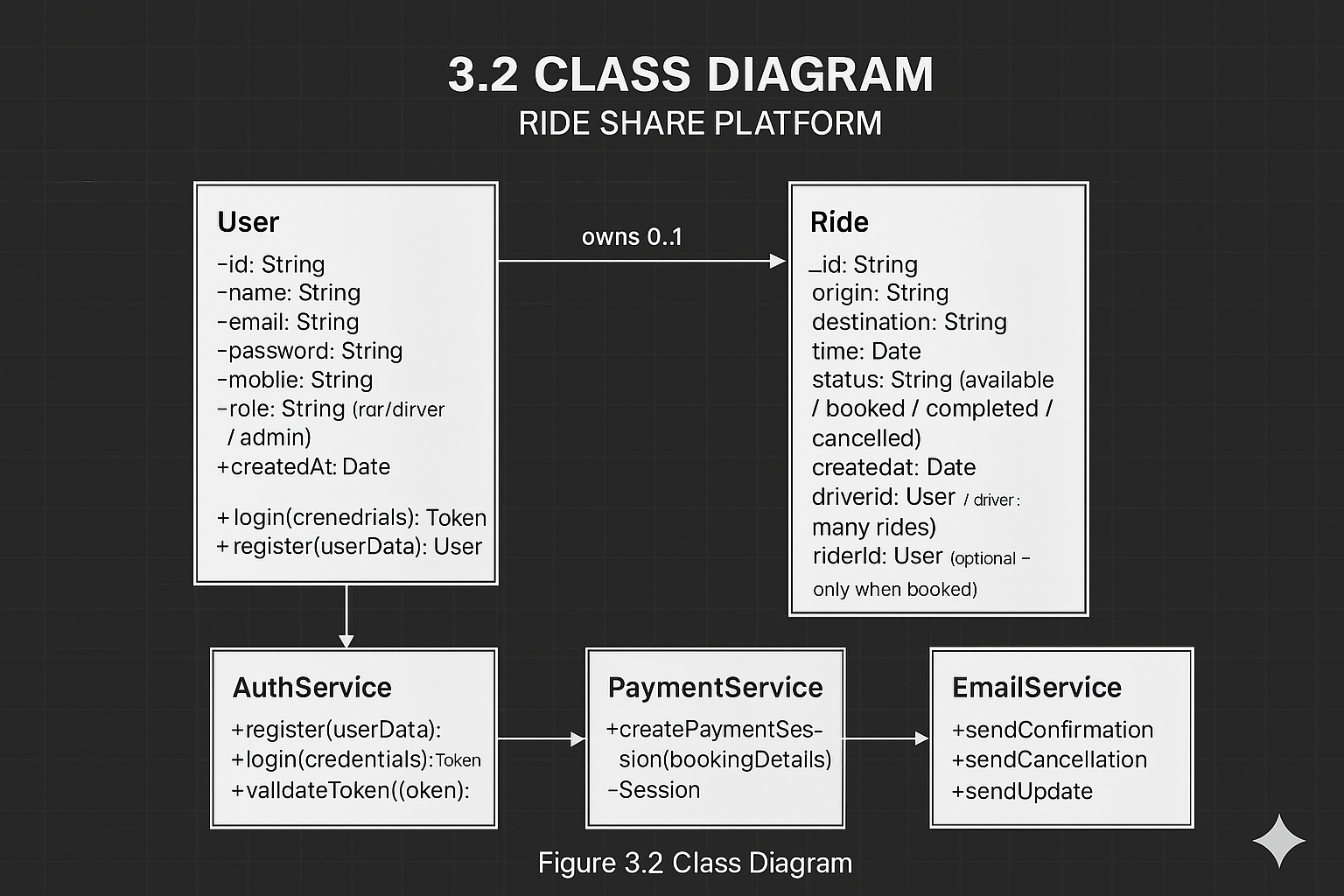


**Figure 3.1 Use Case Diagram**

# CLASS DIAGRAM

A class diagram shows the structure of a system by displaying its classes, attributes, methods, and the relationships between them. Classes are templates for objects, attributes are the data they hold, and methods are the actions they can perform. For the **RIDE SHARE**, the class diagram illustrates how central entities like User, Rider, Rides, and Booking interact, as shown in

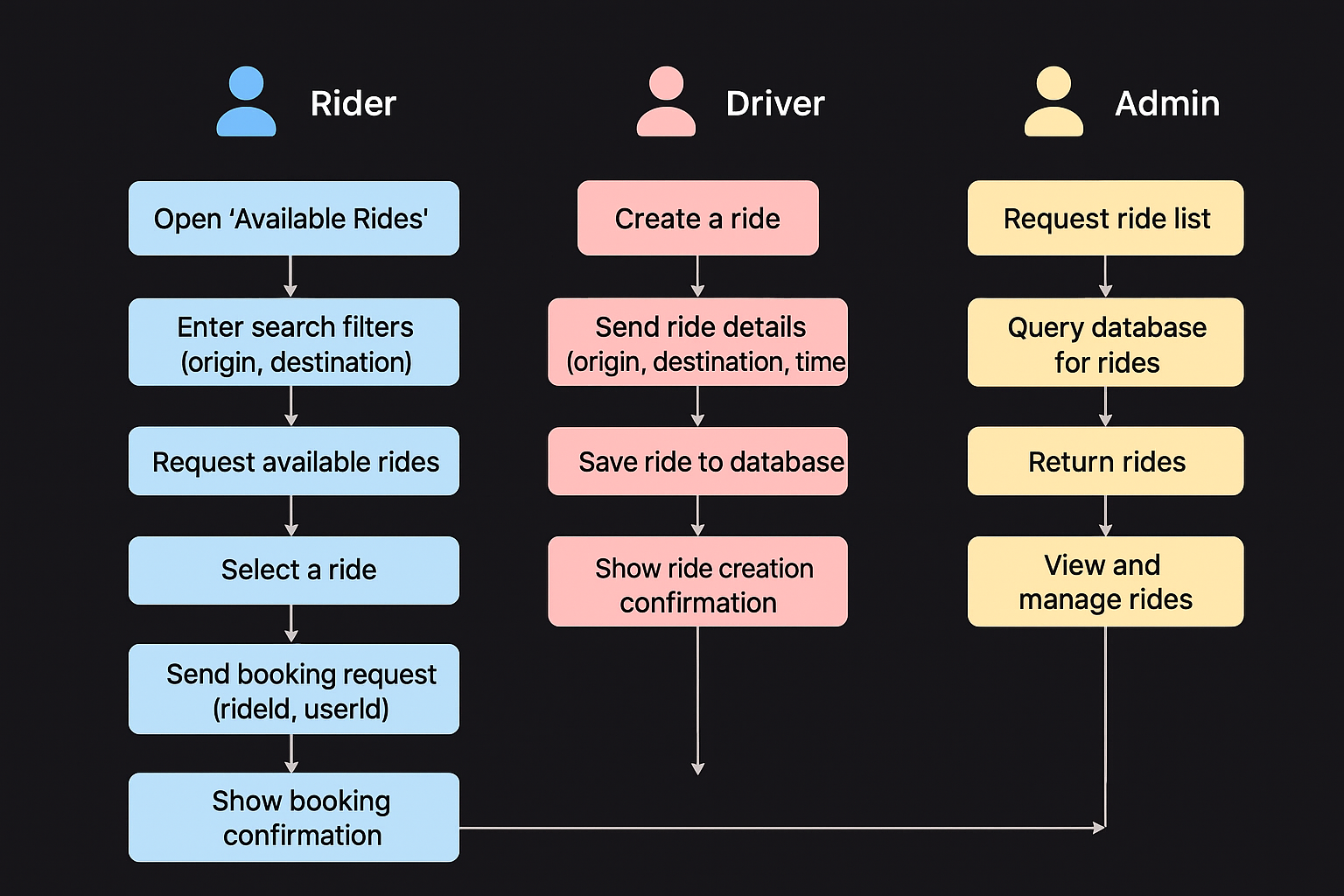
Figure 3.2.



**3.2 CLASS DIAGRAM**

## SEQUENCE DIAGRAM

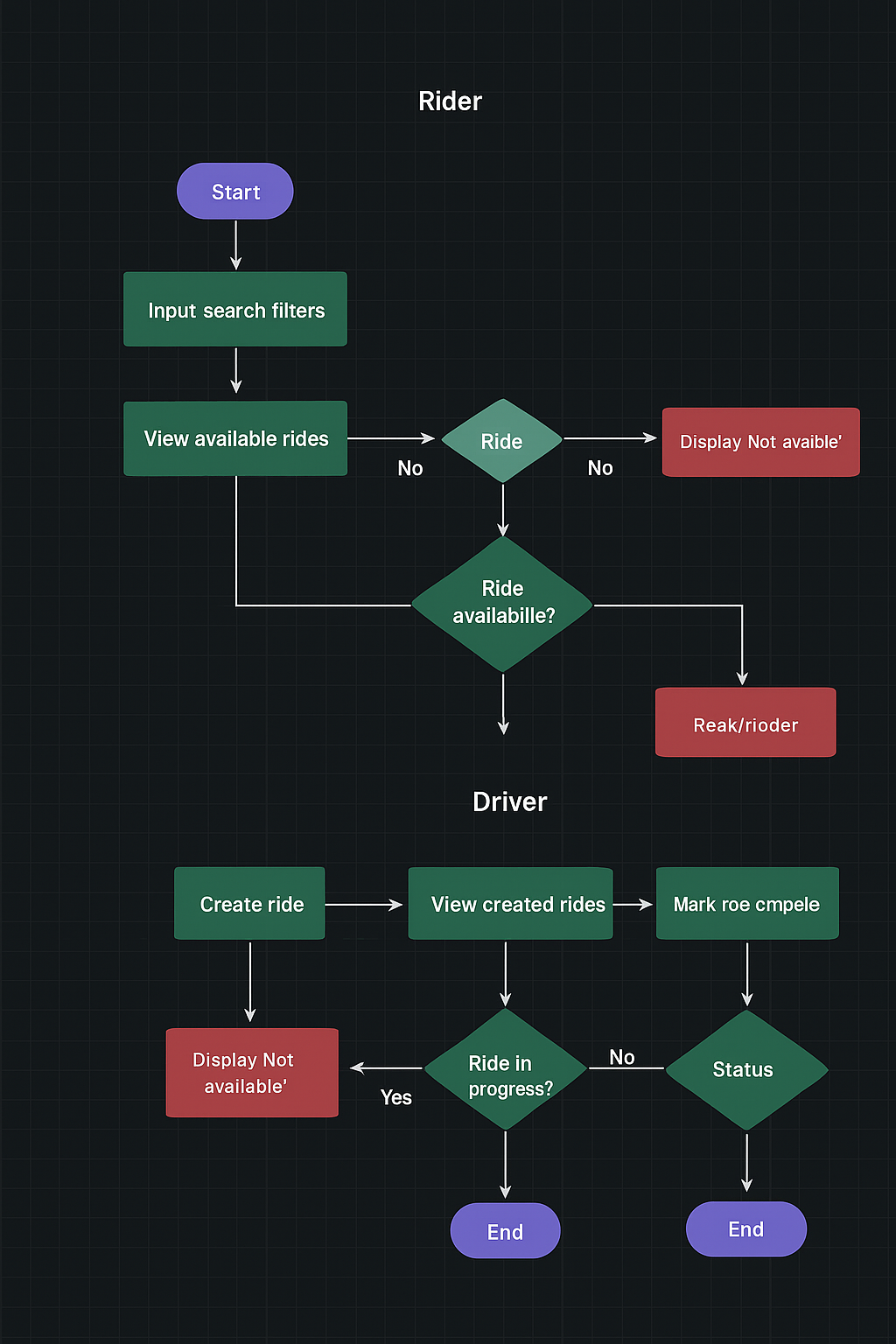
A sequence diagram shows how objects in a system interact over time. Objects are shown as lifelines, and messages between them are shown as arrows. It helps to understand which objects communicate and what actions they trigger. For the **RIDE SHARE**, Figure 3.3 illustrates the sequence of actions when a rider searches for drivers and makes a booking.



## 3.3 SEQUENCE DIAGRAM

## 3.4. ACTIVITY DIAGRAM

An activity diagram shows the flow of activities in a system. It helps visualize how tasks move from one step to another, including sequences, branches, and parallel actions. Elements like forks and joins show different flow paths. For the **RIDE SHARE**, Figure 3.4 illustrates the comprehensive activity flow involved when a Traveler searches for and books a ride, including the branching paths for payment and confirmation.



## 3.4 ACTIVITY DIAGRAM

## DATABASE DESIGN

* + 1. **USER SCHEMA**

// server/models/user.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

// Using a MongoDB ObjectId for the primary key.

\_id: mongoose.Schema.ObjectId,

// Stores the user's name.

name: {

type: String,

required: [true, 'User name is required.'],

trim: true,

},

// Email is needed for application logic and unique identification.

email: {

type: String,

required: [true, 'User email is required'],

unique: true,

lowercase: true,

},

// Stores the hashed password for secure authentication.

password: {

type: String,

required: [true, 'User password is required'],

minlength: 6

},

// Custom field to manage user permissions (Traveler vs. Admin).

role: {

type: String,

enum: ['user', 'admin'],

default: 'user',

},

// Auto-generated timestamp for tracking when the user was created.

createdAt: {

type: Date,

default: Date.now,

},

});

const User = mongoose.model('User', userSchema);

module.exports = User;

## BOOKING SCHEMA

// server/models/booking.js

const mongoose = require('mongoose');

const bookingSchema = new mongoose.Schema({

// Link to the user who made the booking.

user: {

type: mongoose.Schema.ObjectId,

ref: 'User',

required: true,

},

// Link to the specific seat reserved (if confirmed).

seat: {

type: mongoose.Schema.ObjectId,

ref: 'Seat', // Refers to the Seat Model (a new model for your project)

required: false, // Can be false for Waitlisted (WL) bookings

},

train: {

type: mongoose.Schema.ObjectId,

ref: 'Train', // Refers to the Train Model

required: true,

},

journeyDate: {

type: Date,

required: [true, 'Journey date is required'],

},

passengers: [{

name: { type: String, required: true },

age: { type: Number, required: true },

gender: { type: String, enum: ['Male', 'Female', 'Other'], required: true },

// ... other passenger details

}],

totalPrice: {

type: Number,

required: true,

min: [0, 'Total price cannot be negative'],

},

// Status reflects the state of the booking, crucial for RAC/WL flows.

status: {

type: String,

enum: ['pending', 'confirmed', 'cancelled', 'waitlisted', 'rac'],

default: 'pending',

},

paymentId: {

type: String,

required: true,

}

}, { timestamps: true });

const Booking = mongoose.model('Booking', bookingSchema);

module.exports = Booking;

# MODULES DESCRIPTION

### AUTHENTICATION MODULE

The Authentication Module is responsible for managing secure user access across the Ride Share Platform. It implements **JWT-based authentication** to protect all private routes and ensure that only authorized users—categorized as **Riders**, **Drivers**, or **Admins**—can interact with the system. The frontend provides custom-designed login and registration interfaces, allowing users to authenticate using their email and password in a smooth and intuitive manner. During login, the backend validates the submitted credentials against user records stored in **MongoDB**. Upon successful verification, the system issues a **signed JWT token** that encodes the user’s identity and role. This token is stored securely on the client side to maintain an active session, enabling seamless navigation without repeated authentication prompts.

### RIDE AND SCHEDULE MANAGEMENT MODULE

### The Ride and Schedule Management Module oversees the complete lifecycle of ride data within the Ride Share Platform, offering full CRUD (Create, Read, Update, Delete) capabilities, status management, and secure driver–ride associations. Accessible exclusively through the Admin Dashboard, this module enables administrators to create new ride entries with essential details such as origin, destination, scheduled time, driver assignment, and current ride status. Key features include the ability to instantly toggle a ride’s operational state (available, booked, completed, or cancelled), enabling administrators to quickly control system-wide availability. Admins can edit ride details, update schedules, or delete inactive rides as needed. The module also provides comprehensive ride listings with built-in options for filtering, sorting, and searching, ensuring efficient navigation through large datasets

### 3.6.3 BOOKING MANAGEMENT MODULE

The Booking Management Module provides comprehensive oversight of all ride bookings and serves as a central administrative control point within the Ride Share Platform. Accessible exclusively through the **Admin Dashboard**, this module enables administrators to monitor, validate, and manage booking activity across all rides in real time.The booking lifecycle begins when a rider selects an available ride and confirms their reservation. Once a booking is created, the system immediately updates the ride’s status—such as transitioning from *available* to *booked*—ensuring data integrity and real-time synchronization across the platform.

### NOTIFICATION MODULE

The Notification Module is crucial for providing reliable transaction status updates in the IRCTC Ticketing Platform. It uses the backend to **automatically dispatch detailed booking confirmations** to the traveler upon reservation success. Concurrently, it processes **Stripe webhooks** to accurately update the booking's payment status, acting as a critical system integrity alert. Frontend feedback and error messages are managed instantly, ensuring accountability. Overall, the module guarantees transparent communication and system responsiveness for all users.

### TRANSACTION AND LOG MODULE

The Transaction and Log Module plays a critical role in ensuring transparency, accountability, and operational reliability within the Ride Share Platform. This module systematically records all significant system activities, maintaining a chronological audit trail of events performed by Riders, Drivers, and Admins.Key tracked operations include **ride creation**, **ride bookings**, **cancellations**, **status transitions** (available → booked → completed → cancelled), and **user authentication actions**. Each event is logged with essential metadata such as user identity, timestamp, previous and updated values, and the type of operation performed.

### REPORTING AND ANALYTICS MODULE

The Reporting and Analytics Module provides essential operational and performance insights for the Admin Dashboard within the Ride Share Platform. This module consolidates data from ride transactions, booking logs, and user activity to generate meaningful analytics that support informed decision-making. Key performance indicators include total rides created, total bookings, ride completion rates, cancellation patterns, and overall platform activity trends.Integrated with real-time transactional and log data, the module ensures that administrators always access up-to-date statistics.

### AVAILABILITY MANAGEMENT MODULE

The Availability Management Module ensures accurate and efficient handling of ride availability across the Ride Share Platform. It is designed to streamline administrative workflows while maintaining the reliability of booking operations. Administrators can manage ride availability through an intuitive Status Toggle feature that allows instant, one-click updates to a ride’s operational state—such as marking a ride as *available*, *booked*, *completed*, or *cancelled*. This minimizes manual errors and ensures that the system reflects real-time ride capacity.

# CHAPTER 4 SYSTEM TESTING

### INTRODUCTION

The goal of the software business worldwide has always been to offer software products of the highest quality with distinctive characteristics. The team however, cannot ensure these aspects without testing software components under a variety of anticipated and unforeseen circumstances. The process of detecting flaws in a developed product is called software testing. Additionally, it helps in the detection of flaws, gaps and missing requirements by determining whether the actual findings can be reconciled with the anticipated outcomes.

The last step before a product is introduced to the market is testing. It involves looking at, analyzing, observing, and rating several features of a product. Software testing is crucial because it allows any faults or errors in the software to be found early and fixed before the software product is delivered. Reliability, security and high performance are all ensured by thoroughly tested software, which also leads to saving time, cost effectiveness, and customer pleasure. Four phases are included in system testing.

* + - Unit testing
    - Module testing
    - Integration testing
    - Validation testing

### UNIT TESTING

**Unit testing** focuses on testing individual components or modules to ensure that each one performs its specific function as expected. In your **RIDE SHARE** project, this includes:

* **Authentication Module:** Testing ensures that only authorized users (Travelers or Admins) gain access by validating credentials and verifying roles.
* **Train and Schedule Management Module:** This requires testing all **CRUD (Create, Read, Update, Delete)** operations, verifying input data integrity, and confirming the functionality of status changes for trains and schedules.
* **Booking Management Module:** Testing the correct functionality of booking creation, passenger data capture, and confirmation that the system handles various booking statuses (confirmed, waitlisted, RAC).
* **Notification Module:** Testing confirms that the system successfully sends booking confirmation notifications.
* **Transaction and Log Module:** This involves testing the generation and accuracy of the activity log for events like booking creation and payment status updates.
* **Availability Management Module:** This involves testing that the one-click vacancy toggle correctly updates the availability status of a train or seat.

## MODULE TESTING

**Module testing** focuses on testing entire modules as single units, ensuring that all functionalities within a module interact correctly. This tests the internal cohesion of each major component of your system.

* **Authentication Module:** This test case ensures the complete user authentication workflow works, from sign-up and login to token generation and role assignment, as a single, coherent process.
* **Ride & Schedule Management Module:** Tests creating, updating, and managing rides to ensure all ride-related operations work together smoothly.
* **Booking Management Module:** Ensures the full booking flow—selecting a ride, creating a booking, and updating ride status—functions as one cohesive process.
* **Notification Module:** This test verifies that the module correctly receives a booking status update and automatically dispatches a corresponding confirmation email or notification.
* **Log Module:** This test ensures that all critical events within the system—from a user logging in to a booking being created and paid for—are accurately captured and logged in the audit trail.
* **Reporting Module:** Ensures the system can gather data across modules and generate accurate administrative reports.
* **Availability Management Module :**Checks that status toggles (available/booked/completed/cancelled) update correctly and reflect across the entire system.

## INTEGRATION TESTING

**Integration testing** ensures that all modules work together seamlessly. This phase focuses on the "handshakes" between different parts of the system, verifying that data and functionality are passed correctly from one module to another.

* **User to Booking Integration:** Checks that an authenticated user can search available rides, select one, and successfully create a booking with their user ID correctly attached.
* **Booking to Payment Gateway Integration:** Verifies that when a booking is made or cancelled, the ride status updates consistently across the system (available → booked → cancelled/completed).
* **Admin Dashboard to Ride Management Integration:** Ensures that when an admin creates, edits, or cancels a ride, the updates are saved in the database and reflected instantly in the user-facing ride listings.
* **Booking to Availability Integration:** Confirms that a successful booking updates ride availability in real time and that all users see the correct, updated status.
* **Logging to Reporting Integration:** Checks that logs generated from user actions (login, booking, cancellations, status changes) are accurately used by the Reporting Module to produce correct analytics.

## 4.5 VALIDATION TESTING

**Validation testing** checks that the software meets all specified requirements and functions as intended in real-world conditions from the user's perspective. It's the final check to ensure the product meets user needs.

* **End-to-End Traveler Journey:** A complete test from a rider’s viewpoint—searching for available rides, selecting a ride, booking it, and confirming that the ride status updates accordingly and appears in the rider’s “My Rides” section.
* **Admin Workflow Validation:** Checks the full admin process, including logging into the dashboard, creating a new ride, modifying ride details, changing ride status, and ensuring the newly created or updated ride appears correctly for all users in the Available Rides list.
* **Error and Edge Case Validation:** Ensures the system properly handles edge cases such as booking a ride that becomes unavailable, attempting to cancel a completed ride, incorrect login details, or users trying to access restricted admin operations without authorization.
* **Performance and Load Testing:** Validates system stability under high user activity, ensuring that multiple riders can simultaneously search, book, and cancel rides without slowdowns, data conflicts, or system failures.
* **Status Transition Validation:** Tests that ride status changes (available → booked → completed → cancelled) behave accurately, update consistently across all modules, and remain synchronized for all users in real time.

# CHAPTER 5 RESULTS

The Full-Stack Ride Share Platform demonstrates the successful development and deployment of a modern, feature-rich application designed to simplify ride sharing and enhance the overall commuting experience for riders and drivers. Through secure JWT-based authentication, users can safely register, log in, and access role-specific features, ensuring that all personal information, booking activity, and ride history remain protected. Riders can efficiently search for available rides, apply filters such as origin and destination, and view detailed ride information in real time. The system seamlessly supports the core booking workflow, enabling riders to reserve rides, receive instant confirmation, and track upcoming or completed journeys through an intuitive dashboard. Drivers can create rides, manage schedules, update ride status, and maintain control over their availability with minimal effort.Administrators benefit from a dedicated, role-based Admin Dashboard that provides full visibility and control over ride management. Admins can create, edit, monitor, and cancel rides, while real-time ride status tracking ensures transparency and operational accuracy. The Transaction and Audit Log Module captures all key activities—including ride creation, bookings, cancellations, and status transitions—ensuring accountability and system integrity across all user roles. The Reporting and Analytics Module delivers meaningful insights such as total rides, booking counts, cancellation rates, and user activity trends, supporting data-driven decision-making and operational planning. The platform’s search, sort, and filter tools enable users and admins to quickly locate specific rides or track past activity, even within large datasets. Its responsive and user-friendly interface ensures smooth navigation across devices, reducing manual workload and improving the efficiency of everyday operations.

# CHAPTER 6 CONCLUSION AND FUTURE WORK

The Full-Stack Ride Share Platform has successfully demonstrated a secure, scalable, and user-friendly solution for coordinating shared travel between riders and drivers. By integrating core functionalities such as ride discovery, real-time availability tracking, booking management, and dedicated administrative controls into a unified system, the application significantly enhances operational efficiency, transparency, and user experience. Key modules—including Authentication, Ride and Schedule Management, Booking Management, Transaction and Log Tracking, and Reporting & Analytics—work together seamlessly to enable users to book rides effortlessly, drivers to manage their schedules with ease, and administrators to oversee platform activity with accuracy and control. The responsive interface, real-time status updates, and structured data handling ensure that users and admins can manage their tasks efficiently while maintaining complete visibility of ongoing operations.The future development of the Ride Share Platform will focus on enhancing system intelligence, user engagement, and operational automation. Planned improvements include integrating a live GPS-based ride tracking system to enable real-time location updates, implementing dynamic pricing models based on ride demand and time-of-day factors, and adding a driver and rider rating and review system to improve trust and service quality. Additional enhancements will include multilingual and multi-currency support to accommodate a broader user base, in-app chat or emergency contact features for safer communication, and a rewards or loyalty program to increase user retention. From an administrative standpoint, future versions will expand role-based access controls, introduce advanced predictive analytics for identifying peak ride demand patterns, and incorporate automated fraud detection mechanisms to strengthen platform security. Together, these upgrades will further elevate the platform’s usability, reliability, and ability to support large-scale real-world ride-sharing operations.

# APPENDIX 1 CODING

#### Main.jsx :

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import { AuthProvider } from './contexts/AuthContext';

import Navbar from './components/Navbar';

import ProtectedRoute from './components/ProtectedRoute';

import Home from './pages/Home';

import BookRide from './pages/BookRide';

import MyRides from './pages/MyRides';

import Profile from './pages/Profile';

import CreateRide from './pages/CreateRide';

import AvailableRides from './pages/AvailableRides';

import Dashboard from './pages/Dashboard';

import Login from './pages/Login';

import Signup from './pages/Signup';

import Signin from './pages/Signin';

import AdminRides from './pages/AdminRides';

function App() {

  return (

    <AuthProvider>

      <Router>

        <Navbar />

        <div style={{ maxWidth: 800, margin: '2rem auto', padding: '1rem' }}>

          <Routes>

            <Route path="/" element={<Home />} />

            <Route path="/signin" element={<Signin />} />

            <Route path="/signup" element={<Signup />} />

            <Route path="/login" element={<Login />} />

            <Route path="/dashboard" element={

              <ProtectedRoute>

                <Dashboard />

              </ProtectedRoute>

            } />

            <Route path="/book" element={

              <ProtectedRoute requireRole="rider">

                <BookRide />

              </ProtectedRoute>

            } />

            <Route path="/available" element={

              <ProtectedRoute>

                <AvailableRides />

              </ProtectedRoute>

            } />

            <Route path="/create" element={

              <ProtectedRoute requireRole="driver">

                <CreateRide />

              </ProtectedRoute>

            } />

            <Route path="/rides" element={

              <ProtectedRoute>

                <MyRides />

              </ProtectedRoute>

            } />

            <Route path="/profile" element={

              <ProtectedRoute>

                <Profile />

              </ProtectedRoute>

            } />

            <Route path="/admin/rides" element={

              <ProtectedRoute requireRole="admin">

                <AdminRides />

              </ProtectedRoute>

            } />

          </Routes>

        </div>

      </Router>

    </AuthProvider>

  );

}

export default App;

AdminRiders.jsx:

import React, { useEffect, useState } from 'react';

import { useAuth } from '../contexts/AuthContext';

import API from '../api';

const AdminRides = () => {

  const { user } = useAuth();

  const [rides, setRides] = useState([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState('');

  const [start, setStart] = useState('');

  const [destination, setDestination] = useState('');

  const [status, setStatus] = useState('');

  const [isSearching, setIsSearching] = useState(false);

  useEffect(() => {

    fetchRides();

  }, []);

  const fetchRides = async (query, signal) => {

    try {

      const response = await API.get('/rides', { params: query || undefined, signal });

      setRides(response.data);

    } catch (error) {

      setError('Failed to fetch rides');

      console.error('Error fetching rides:', error);

    } finally {

      setLoading(false);

    }

  };

  const onSearch = async () => {

    setIsSearching(true);

    setError('');

    const controller = new AbortController();

    try {

      const params = {};

      if (start.trim()) params.origin = start.trim();

      if (destination.trim()) params.destination = destination.trim();

      if (status.trim()) params.status = status.trim();

      await fetchRides(params, controller.signal);

    } finally {

      setIsSearching(false);

    }

  };

  const formatDate = (dateString) => new Date(dateString).toLocaleString();

  if (!user || user.role !== 'admin') {

    return <div className="alert alert-danger">Access denied. Admins only.</div>;

  }

  if (loading) {

    return <div className="d-flex justify-content-center"><div className="spinner-border" role="status"></div></div>;

  }

  return (

    <div className="container">

      <h2>All Rides (Admin)</h2>

      <div className="row g-2 mb-3">

        <div className="col-12 col-md-4">

          <input

            type="text"

            className="form-control"

            placeholder="Start (origin)"

            value={start}

            onChange={(e) => setStart(e.target.value)}

          />

        </div>

        <div className="col-12 col-md-4">

          <input

            type="text"

            className="form-control"

            placeholder="Destination"

            value={destination}

            onChange={(e) => setDestination(e.target.value)}

          />

        </div>

        <div className="col-12 col-md-2">

          <select className="form-select" value={status} onChange={(e) => setStatus(e.target.value)}>

            <option value="">All Statuses</option>

            <option value="available">Available</option>

            <option value="booked">Booked</option>

            <option value="completed">Completed</option>

            <option value="cancelled">Cancelled</option>

          </select>

        </div>

        <div className="col-12 col-md-2 d-grid">

          <button className="btn btn-primary" onClick={onSearch} disabled={isSearching}>

            {isSearching ? 'Searching...' : 'Search'}

          </button>

        </div>

      </div>

      {error && (

        <div className="alert alert-danger" role="alert">

          {error}

        </div>

      )}

      {rides.length === 0 ? (

        <div className="alert alert-info" role="alert">

          No rides found.

        </div>

      ) : (

        <div className="row">

          {rides.map((ride) => (

            <div key={ride.\_id} className="col-md-6 mb-3">

              <div className="card">

                <div className="card-body">

                  <h5 className="card-title">{ride.origin} → {ride.destination}</h5>

                  <p className="card-text">

                    <strong>Driver:</strong> {ride.driverId?.name || 'Unknown'}<br/>

                    <strong>Rider:</strong> {ride.riderId?.name || 'N/A'}<br/>

                    <strong>Time:</strong> {formatDate(ride.time)}<br/>

                    <strong>Status:</strong>

                    <span className={`badge ms-1 ${

                      ride.status === 'available' ? 'bg-success' :

                      ride.status === 'booked' ? 'bg-warning' :

                      ride.status === 'completed' ? 'bg-secondary' : 'bg-danger'

                    }`}>

                      {ride.status}

                    </span>

                  </p>

                </div>

              </div>

            </div>

          ))}

        </div>

      )}

    </div>

  );

};

export default AdminRides;

Bookride.jsx:

import React, { useState, useEffect } from 'react';

import { useAuth } from '../contexts/AuthContext';

import API from '../api';

const BookRide = () => {

  const [availableRides, setAvailableRides] = useState([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState('');

  const [success, setSuccess] = useState('');

  const [filterOrigin, setFilterOrigin] = useState('');

  const [filterDestination, setFilterDestination] = useState('');

  const [sortBy, setSortBy] = useState('time');

  const { user } = useAuth();

  useEffect(() => {

    fetchAvailableRides();

  }, []);

  const fetchAvailableRides = async () => {

    try {

      const response = await API.get('/rides/available');

      setAvailableRides(response.data);

    } catch (error) {

      setError('Failed to fetch available rides');

      console.error('Error fetching rides:', error);

    } finally {

      setLoading(false);

    }

  };

  const handleBookRide = async (rideId, driverName) => {

    if (!window.confirm(`Are you sure you want to book this ride with ${driverName}?`)) {

      return;

    }

    try {

      await API.post(`/rides/book/${rideId}`);

      setSuccess('Ride booked successfully! You can view it in "My Rides".');

      fetchAvailableRides(); // Refresh the list

      setTimeout(() => setSuccess(''), 5000);

    } catch (error) {

      setError(error.response?.data?.error || 'Failed to book ride');

      setTimeout(() => setError(''), 5000);

    }

  };

  const formatDate = (dateString) => {

    return new Date(dateString).toLocaleString();

  };

  const formatTime = (dateString) => {

    return new Date(dateString).toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });

  };

  const formatDateOnly = (dateString) => {

    return new Date(dateString).toLocaleDateString();

  };

  const filteredRides = availableRides.filter(ride => {

    const originMatch = ride.origin.toLowerCase().includes(filterOrigin.toLowerCase());

    const destinationMatch = ride.destination.toLowerCase().includes(filterDestination.toLowerCase());

    return originMatch && destinationMatch;

  });

  const sortedRides = [...filteredRides].sort((a, b) => {

    switch (sortBy) {

      case 'time':

        return new Date(a.time) - new Date(b.time);

      case 'origin':

        return a.origin.localeCompare(b.origin);

      case 'destination':

        return a.destination.localeCompare(b.destination);

      default:

        return 0;

    }

  });

  const getTimeUntilRide = (dateString) => {

    const now = new Date();

    const rideTime = new Date(dateString);

    const diffMs = rideTime - now;

    const diffHours = Math.floor(diffMs / (1000 \* 60 \* 60));

    const diffMinutes = Math.floor((diffMs % (1000 \* 60 \* 60)) / (1000 \* 60));

    if (diffMs < 0) return 'Past';

    if (diffHours < 1) return `${diffMinutes} minutes`;

    if (diffHours < 24) return `${diffHours} hours`;

    return `${Math.floor(diffHours / 24)} days`;

  };

  if (loading) {

    return (

      <div className="d-flex justify-content-center align-items-center" style={{ minHeight: '400px' }}>

        <div className="text-center">

          <div className="spinner-border text-primary" role="status"></div>

          <p className="mt-2">Loading available rides...</p>

        </div>

      </div>

    );

  }

  return (

    <div className="container">

      <div className="row">

        <div className="col-12">

          <h2 className="mb-4">🚗 Book a Ride</h2>

          <p className="text-muted mb-4">Find and book rides from drivers in your area</p>

        </div>

      </div>

      {/\* Success/Error Messages \*/}

      {success && (

        <div className="alert alert-success alert-dismissible fade show" role="alert">

          <i className="bi bi-check-circle me-2"></i>

          {success}

          <button type="button" className="btn-close" onClick={() => setSuccess('')}></button>

        </div>

      )}

      {error && (

        <div className="alert alert-danger alert-dismissible fade show" role="alert">

          <i className="bi bi-exclamation-triangle me-2"></i>

          {error}

          <button type="button" className="btn-close" onClick={() => setError('')}></button>

        </div>

      )}

      {/\* Filters and Search \*/}

      <div className="row mb-4">

        <div className="col-md-3 mb-2">

          <label htmlFor="filterOrigin" className="form-label">From</label>

          <input

            type="text"

            className="form-control"

            id="filterOrigin"

            placeholder="Enter origin..."

            value={filterOrigin}

            onChange={(e) => setFilterOrigin(e.target.value)}

          />

        </div>

        <div className="col-md-3 mb-2">

          <label htmlFor="filterDestination" className="form-label">To</label>

          <input

            type="text"

            className="form-control"

            id="filterDestination"

            placeholder="Enter destination..."

            value={filterDestination}

            onChange={(e) => setFilterDestination(e.target.value)}

          />

        </div>

        <div className="col-md-3 mb-2">

          <label htmlFor="sortBy" className="form-label">Sort by</label>

          <select

            className="form-select"

            id="sortBy"

            value={sortBy}

            onChange={(e) => setSortBy(e.target.value)}

          >

            <option value="time">Time</option>

            <option value="origin">Origin</option>

            <option value="destination">Destination</option>

          </select>

        </div>

        <div className="col-md-3 mb-2 d-flex align-items-end">

          <button

            className="btn btn-outline-secondary w-100"

            onClick={() => {

              setFilterOrigin('');

              setFilterDestination('');

              setSortBy('time');

            }}

          >

            Clear Filters

          </button>

        </div>

      </div>

      {/\* Results Count \*/}

      <div className="row mb-3">

        <div className="col-12">

          <p className="text-muted">

            {filteredRides.length} ride{filteredRides.length !== 1 ? 's' : ''} available

            {filterOrigin || filterDestination ? ' (filtered)' : ''}

          </p>

        </div>

      </div>

      {sortedRides.length === 0 ? (

        <div className="text-center py-5">

        <div className="mb-3">

            <i className="bi bi-car-front" style={{ fontSize: '3rem', color: '#6c757d' }}></i>

          </div>

          <h4 className="text-muted">No rides available</h4>

          <p className="text-muted">

            {filterOrigin || filterDestination

              ? 'No rides match your search criteria. Try adjusting your filters.'

              : 'No available rides at the moment. Check back later or create a ride request!'

            }

          </p>

          <button className="btn btn-primary" onClick={fetchAvailableRides}>

            <i className="bi bi-arrow-clockwise me-2"></i>

            Refresh

          </button>

        </div>

      ) : (

        <div className="row">

          {sortedRides.map((ride) => (

            <div key={ride.\_id} className="col-12 col-md-6 col-lg-4 mb-4">

              <div className="card h-100 shadow-sm">

                <div className="card-header bg-primary text-white">

                  <h5 className="card-title mb-0">

                    <i className="bi bi-geo-alt me-2"></i>

                    {ride.origin} → {ride.destination}

                  </h5>

                </div>

                <div className="card-body">

                  <div className="row mb-3">

                    <div className="col-6">

                      <small className="text-muted">Driver</small>

                      <p className="mb-0 fw-bold">{ride.driverId?.name || 'Unknown'}</p>

                    </div>

                    <div className="col-6">

                      <small className="text-muted">Status</small>

                      <p className="mb-0">

                        <span className="badge bg-success">Available</span>

                      </p>

                    </div>

                  </div>

                  <div className="row mb-3">

                    <div className="col-6">

                      <small className="text-muted">Driver Mobile</small>

                      <p className="mb-0 fw-bold">{ride.driverId?.mobile || 'N/A'}</p>

                    </div>

                    <div className="col-6">

                      <small className="text-muted">Contact</small>

                      <p className="mb-0">

                        <a href={`tel:${ride.driverId?.mobile}`} className="btn btn-sm btn-outline-primary">

                          <i className="bi bi-telephone me-1"></i>Call

                        </a>

                      </p>

                    </div>

                  </div>

                  <div className="row mb-3">

                    <div className="col-6">

                      <small className="text-muted">Date</small>

                      <p className="mb-0">{formatDateOnly(ride.time)}</p>

                    </div>

                    <div className="col-6">

                      <small className="text-muted">Time</small>

                      <p className="mb-0 fw-bold">{formatTime(ride.time)}</p>

                    </div>

        </div>

        <div className="mb-3">

                    <small className="text-muted">Departure in</small>

                    <p className="mb-0">

                      <span className={`badge ${getTimeUntilRide(ride.time) === 'Past' ? 'bg-danger' : 'bg-info'}`}>

                        {getTimeUntilRide(ride.time)}

                      </span>

                    </p>

                  </div>

                  <div className="d-grid">

                    <button

                      className="btn btn-primary btn-lg"

                      onClick={() => handleBookRide(ride.\_id, ride.driverId?.name)}

                    >

                      <i className="bi bi-bookmark-plus me-2"></i>

                      Book This Ride

                    </button>

                  </div>

                </div>

                <div className="card-footer text-muted">

                  <small>

                    <i className="bi bi-clock me-1"></i>

                    Created {formatDate(ride.createdAt)}

                  </small>

                </div>

              </div>

            </div>

          ))}

        </div>

      )}

      {/\* Refresh Button \*/}

      <div className="row mt-4">

        <div className="col-12 text-center">

          <button className="btn btn-outline-primary" onClick={fetchAvailableRides}>

            <i className="bi bi-arrow-clockwise me-2"></i>

            Refresh Rides

          </button>

        </div>

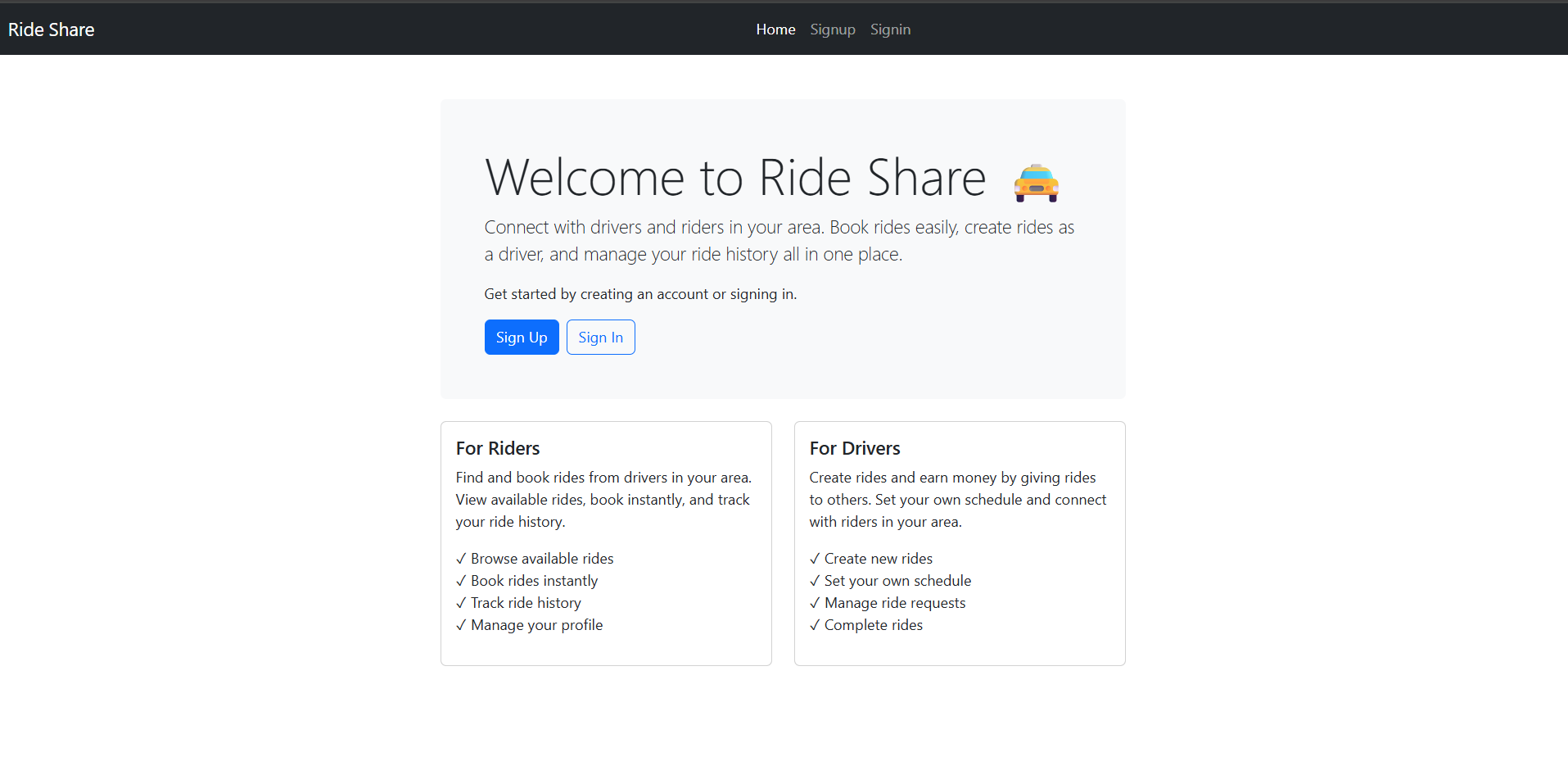
        </div>

    </div>

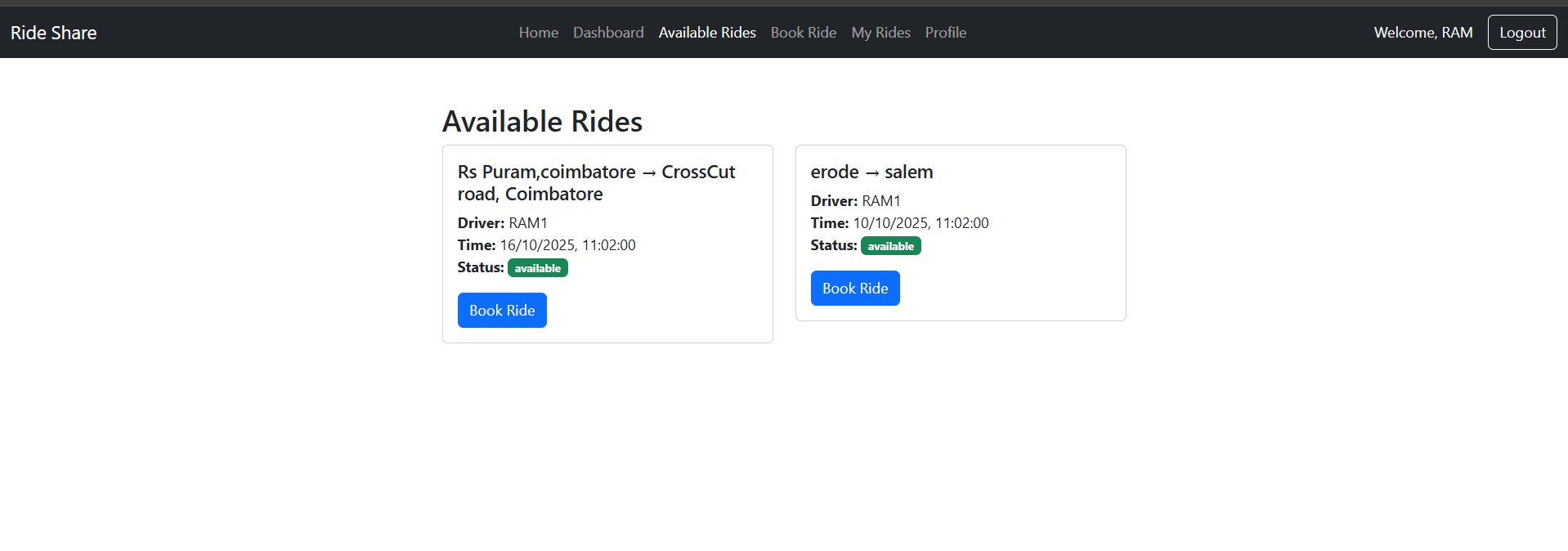
  );};

export default BookRide;

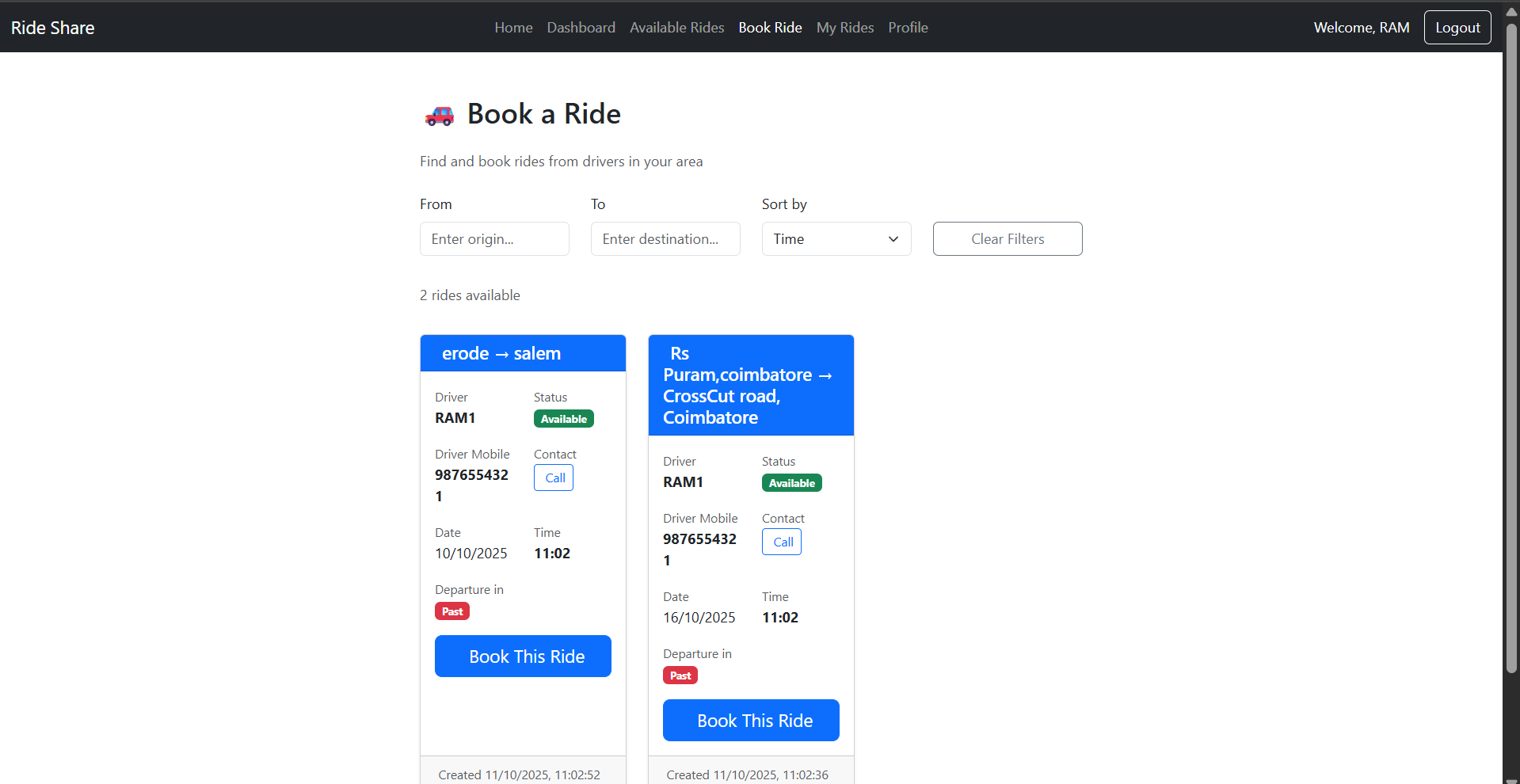
**APPENDIX2**

****

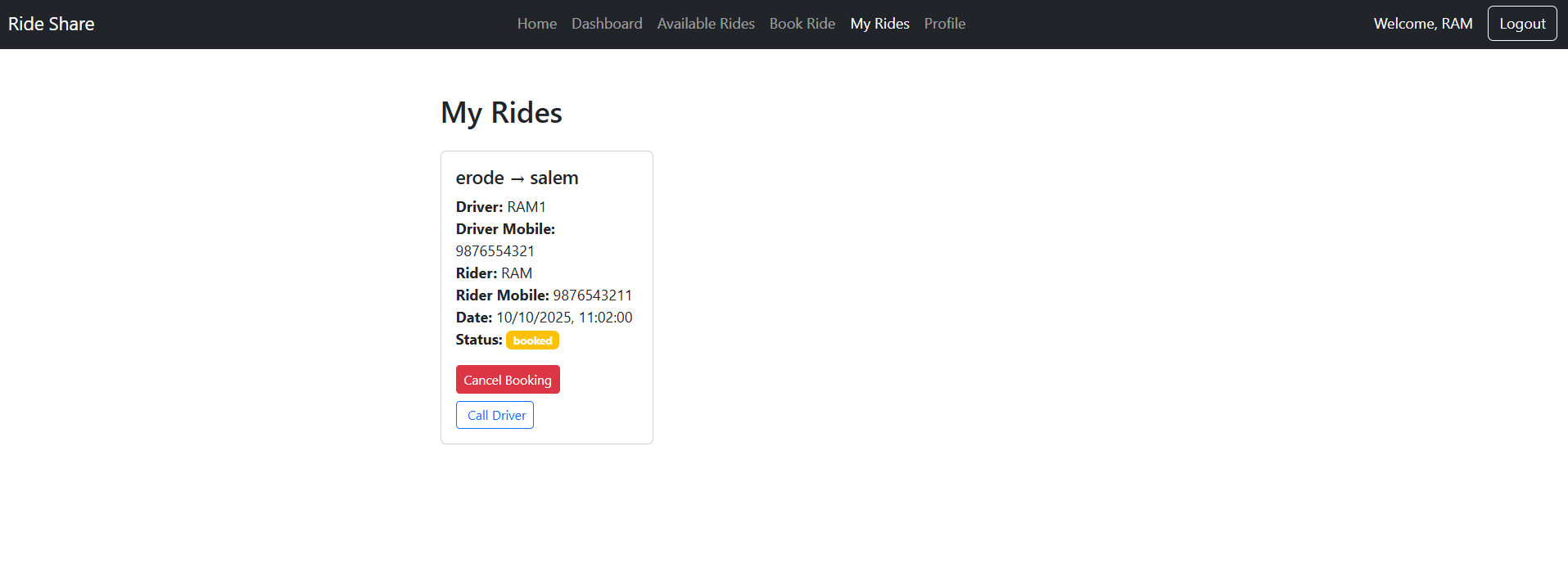
**Figure A2.1 Home page**



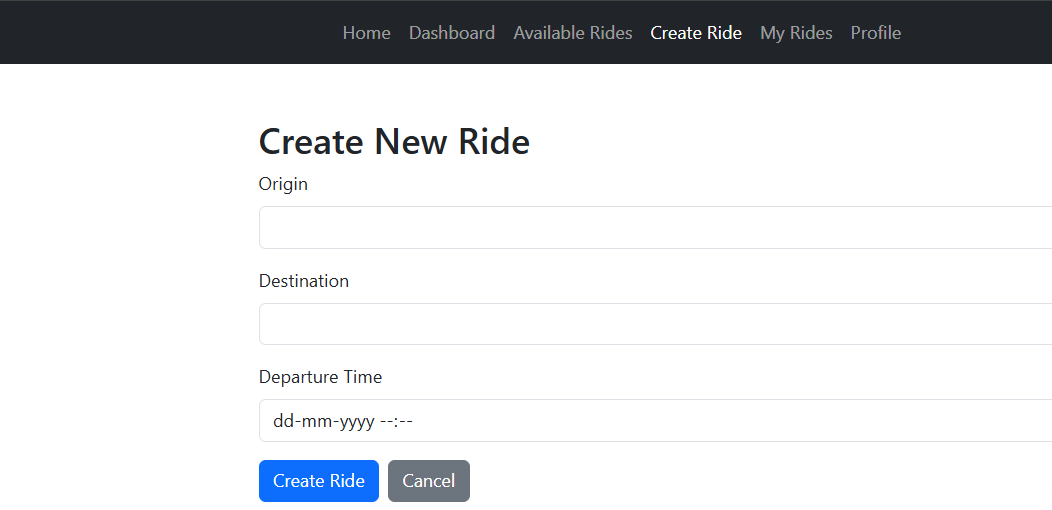
**Figure A2.2 Home page**

****

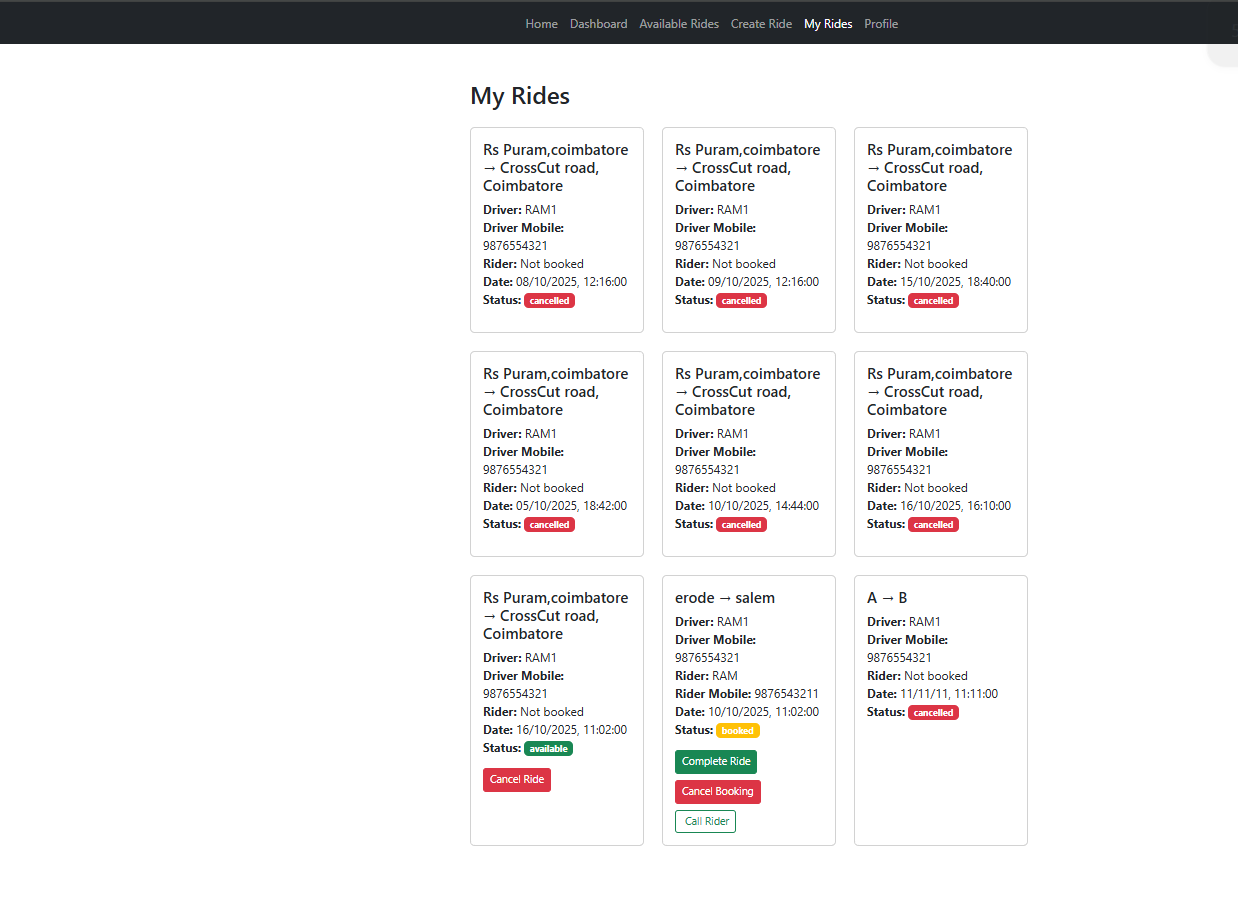
**Figure A2.3 Book ride page**



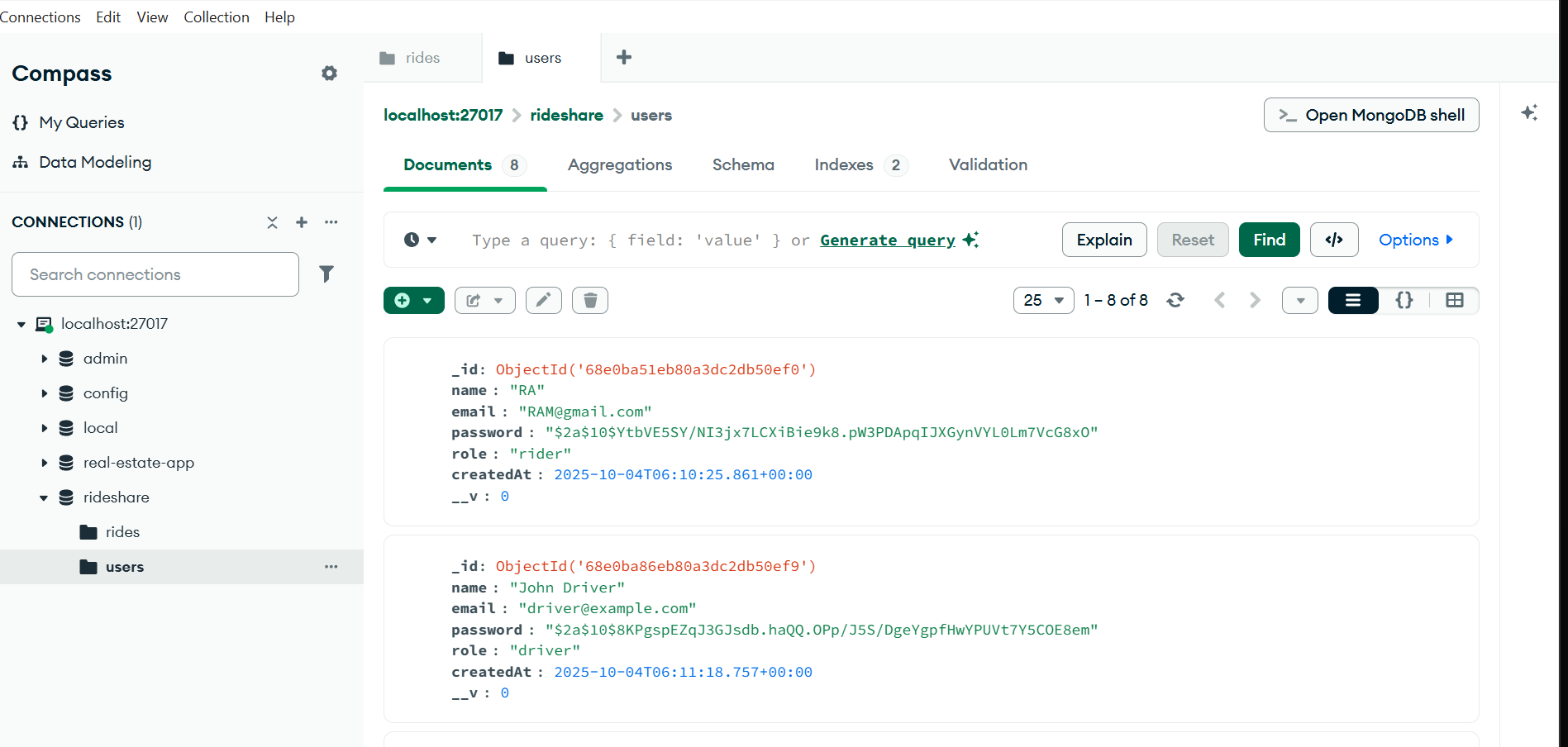
**Figure A2.4 Ride page**



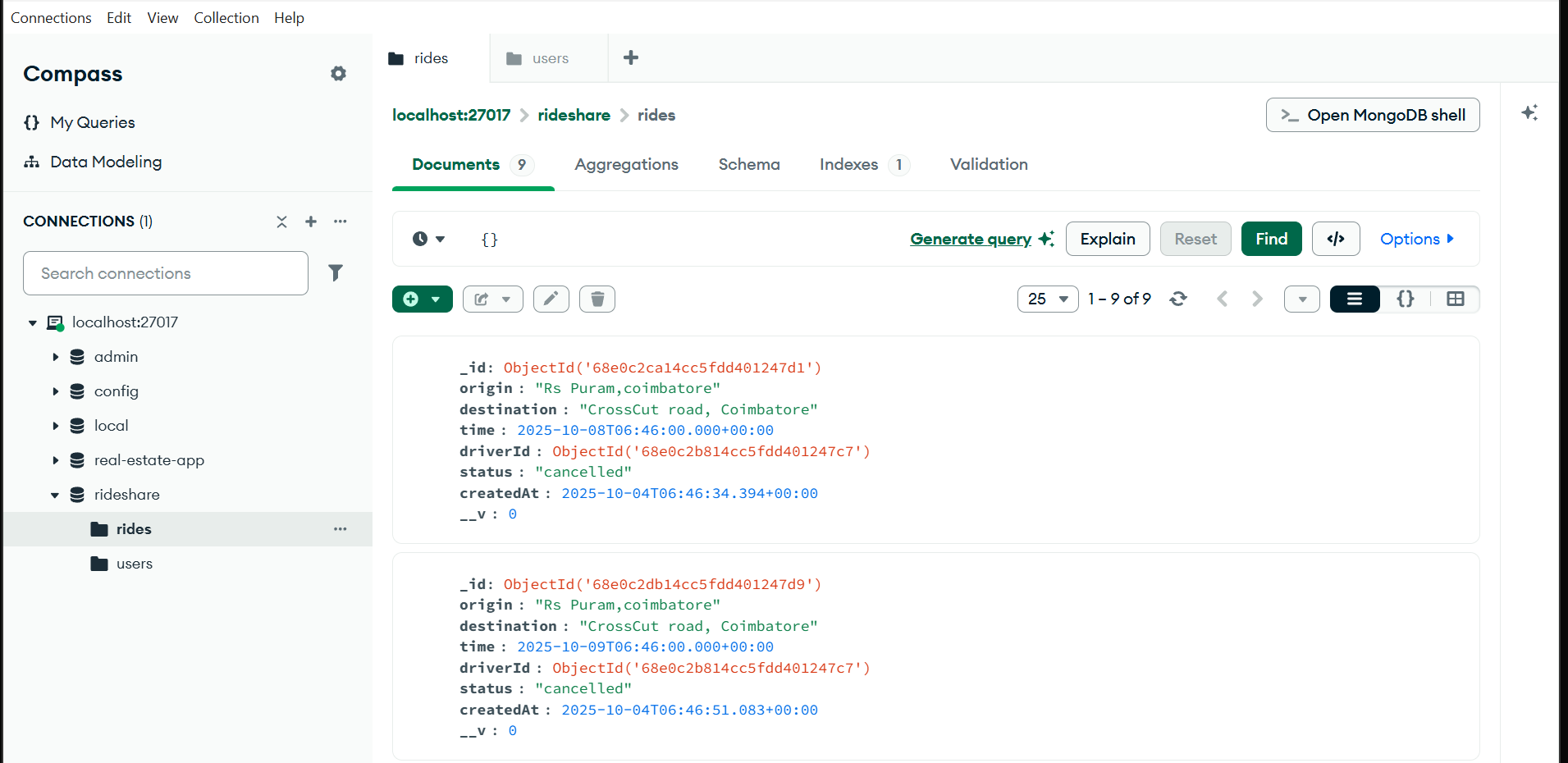
**Figure A2.5 Ride create page**



**Figure A2.6 History page**



**Figure A2.7 User Database**



**Figure A2.8 Driver database**

## REFERENCES

1. https://react.dev/learn
2. https:/[/www.mongodb.com/developer/languages/javascript/getting-started-with-](http://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-) [mongodb-](http://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-) and-mongoose/
3. **Sharma, P., & Nair, A. (2023).** Digital Transformation in Ride-Sharing Platforms for Enhanced Urban Mobility. International Journal of Trend in Scientific Research and Development (IJTSRD), 7(4), 310–318. Retrieved from <https://www.ijtsrd.com>.
4. **Rathod, K., & Deshmukh, V. (2022).** Development of a Web-Based Ride-Sharing Management System. Applied Information Technology and Computer Science, 4(2), 67–75. Retrieved from <https://publisher.uthm.edu.my>.
5. **Ali, M., & Prasad, S. (2021).** Design and Implementation of a Real-Time Ride Booking and Tracking System Using GPS and Web Technologies. Journal of Applied Information Technology, 5(1), 52–60. Retrieved from <https://javit.ppj.unp.ac.id>.
6. **Khan, R., & Mehta, T. (2021).** Performance Optimization of Real-Time Transportation Applications Using the MERN Stack. International Journal of Engineering Research & Technology (IJERT), 10(5), 690–698. Retrieved from <https://www.ijert.org>.
7. **Joshi, L., & Srivastava, K. (2021).** A Comprehensive Review of MERN Stack for Scalable Web and Mobile Applications. International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), 7(7), 210–219. Retrieved from <https://ijsrcseit.com>.
8. **RideShare Frontend (2024).** Retrieved from <https://ride-frontend-blush.vercel.app> .
9. https://github.com/RDharaneesh/Ride-share-service1.

**MAPPING TO SDG GOALS:**

The RideShare Application aligns with the United Nations Sustainable Development Goal (SDG) 9: **Industry, Innovation and Infrastructure**. This goal emphasizes strengthening digital infrastructure, promoting technological innovation, and building efficient systems that enhance service delivery and operational workflows. By offering an online platform for booking rides, scheduling pickups, tracking drivers, and managing trip history, the project contributes to the digital transformation of traditional transportation services. By developing an integrated digital system that connects riders with drivers for daily commuting, long-distance travel, and shared rides, the application supports improved coordination and streamlined management of transportation operations. Users can easily request rides, monitor driver location in real time, view fare estimates, and access details of previous trips, promoting organized mobility planning and reliable access to safe transportation services.