System Design Document

for

VEHICLE BOOKING SERVICE

Version 1.0

Prepared by

Group Name: DBMS_CS04_GRP1

Sahil Muhammed	B230089CS	sahil_b230089cs@nitc.ac.in
Sreehari Sanjeev	B230574CS	sreehari_b230567cs@nitc.ac.in
Nishanth K Nizar	B230471CS	nishanth_b230471cs@nitc.ac.in
Mohammed Omar	B230426CS	mohammed_b230426cs@nitc.ac.in

Instructor: Professor Prabhu Mohandas

Course: CS2011E DBMS

Date: 09/03/2025

TABLE OF CONTENTS

1 Introduction

1.1 Document Purpose

The purpose of the System Design document is to give the reader a descriptive overview of the design choices, system architecture, data flow, security considerations and deployment strategies involved in the production of the vehicle booking service.

1.2 Scope of the Document

The vehicle booking service is a web-based platform that allows passengers to book hassle free rides while also ensuring fair pay for the drivers. The system includes an ETA of the driver, ability to rate/review, seamless payment facility, emergency SOS and much more. The system relies on Next.js for the front-end, Node.js with Express.js for the back-end and MySQL for the database.

1.3 Intended Audience and document overview

The System design document is intended for the development team, project managers, marketing staff, users, testers and Professor Prabhu Mohandas. This document aims to deliver an informative rundown through the design and architectural conveniences we have opted for when creating our web-app..

1.4 Definitions, Acronyms and Abbreviations

- **DBMS:** Database Management System
- **UI/UX:** User Interface/User Experience
- **ETA:** Estimated Time of Arrival
- **SQL:** Structured Query Language

1.5 Reference and Acknowledgements

- IEEE Standard for Software Requirement Specifications
- React.js and Node.js Official Documentation
- MySQL Documentation

2 System Overview

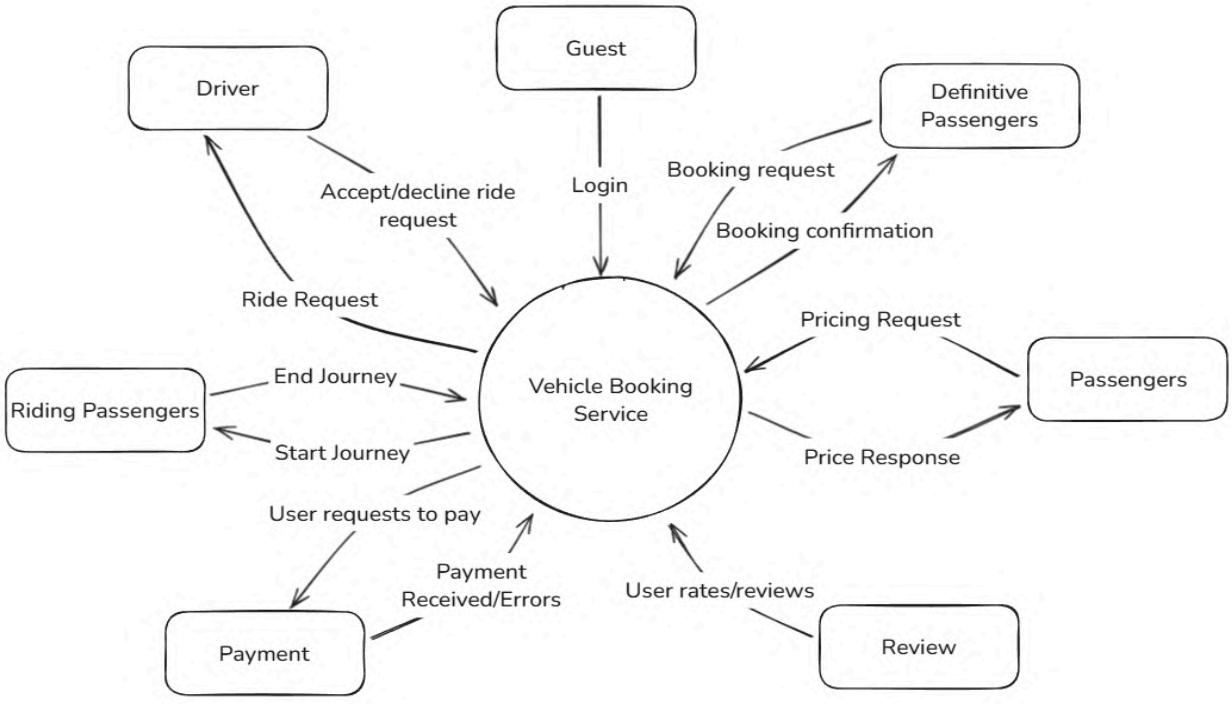
2.1 High-Level Architecture

The system is built following the client-server architecture. The following are the components that play a critical role in our production:

- **Frontend:** Next.js (React.js)
- **Backend:** Node.js with Express.js
- **Database:** MySQL
- **Third-Party Integrations:** Google Maps API, Razorpay API
- **Deployment:** Vercel for frontend, Azure for backend

2.2 System Context Design

A diagram to show the various interactions between the drivers, the passengers and the server.



3 System Architecture

3.1 Architectural Design:

- The system follows a multi-layered architectural design:
    - **Presentation Layer:** This is the layer that handles the user's interaction with the web-app. This is the layer responsible for what the user 'sees' when interacting with the web-app. Buttons and User Forms are available for the user to interact with. Implemented using Next.js.
    - **Application Layer:** This is the layer that handles the backend logic. The requests made by the user need to be served. These requests could be of various forms and are served using proper API routes and backend logic linked to the database.
    - **Database Layer:** The database layer is self-explanatory and serves the function of giving the user the ability to store relevant details with us.

3.2 Subsystems Description:

- The system is divided into subsystems. Following the principle of compartmentalization ensures that a malfunction in any one of the subsystems won't compromise the entire system.
    - **User Management:** Handles users, be it passengers, drivers or the admin; from the functionality they can access the roles they assume, the user management subsystem handles everything.
    - **Ride Booking Module:** This module handles the actual ride booking functionality of our service. Handling an excessive number of passenger or stagnant drivers, this subsystem aims to connect the driver and the passenger as early as possible.
    - **Driver Management:** Once the ride is booked, it needs to be confirmed, and this is dependent on the number of drivers available in that area at that particular time. The driver management subsystem aims to solve these issues with ease and ensure that no passenger is left devoid of a driver.
    - **Payment Processing:** The passenger having availed the transportation, must obviously pay the driver a fare for the transit. This involves a third party service (Razorpay) that has been integrated into our service as a payment partner. When a passenger wishes to pay a driver, they simply use the Razorpay API which seamlessly ensures bank to bank UPI transfers.
    - **Admin Panel:** The admin panel exists solely for the developers to regulate and monitor activity in the web-app itself. From addressing downtimes and fixing bugs to resolving customer complaints and ensuring regulatory compliance, the admin panel

exists as a platform for the developers to guarantee as minimal downtime as possible.

3.3 Technology Stack:

We have opted for a comprehensive technology stack of the following components, each performing a unique and key duty in the functioning of our web-app.

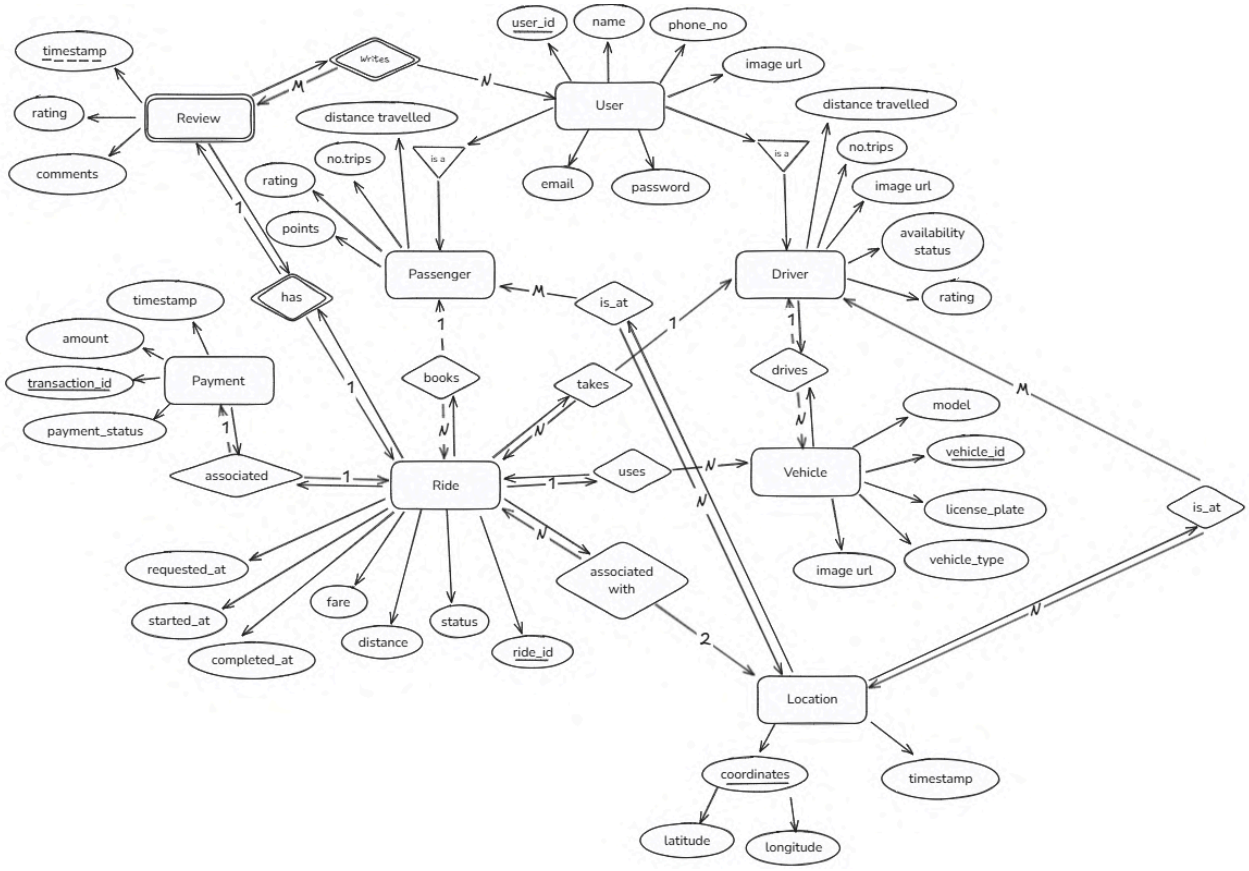
Component	Technology
Frontend	Next.js
Backend	Node.js with Express.js
Database	MySQL
Payment Gateway	Razorpay API
Mapping	Google Maps API
Deployment	Vecel and Azure

4 Detailed Design

4.1 Database Design (ER diagram)

- The system will use an ER diagram to model the entities and relationships in our database.The SQL schema that will be used is as follows:
    - User
    - Ride
    - Payment
    - Review
    - Vehicle
    - Location

- The ER diagram illustrated below has been drawn using excalidraw.com. We have used double arrows to represent total participation. The arrows were simply used to draw the diagrams more efficiently, they do not have the same significance as the ones which were discussed in class.



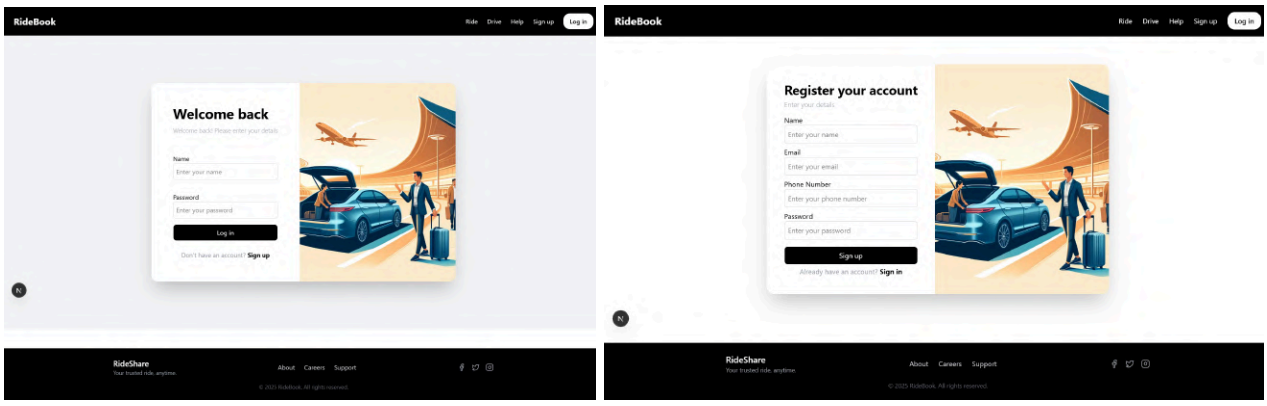
4.2 API Design

Endpoint	Method	Description
/signup	POST	Registers a new user
/login	POST	Authenticates a user who's trying to login
/user/profile	GET	Retrieves the details of a logged in user
/user/update	PUT	Updates the details of a logged in user
/rides/eta	POST	Retrieves the ETA of a driver
/driver/availability	PUT	Updates the availability of a driver
/rides/accept	POST	Enables a driver to accept a ride request
/rides/book	POST	Enables a passenger to make a ride request
/rides/reject	POST	Enables a driver to reject a ride request
/payment/process	POST	Processes the payments using the Razorpay endpoint

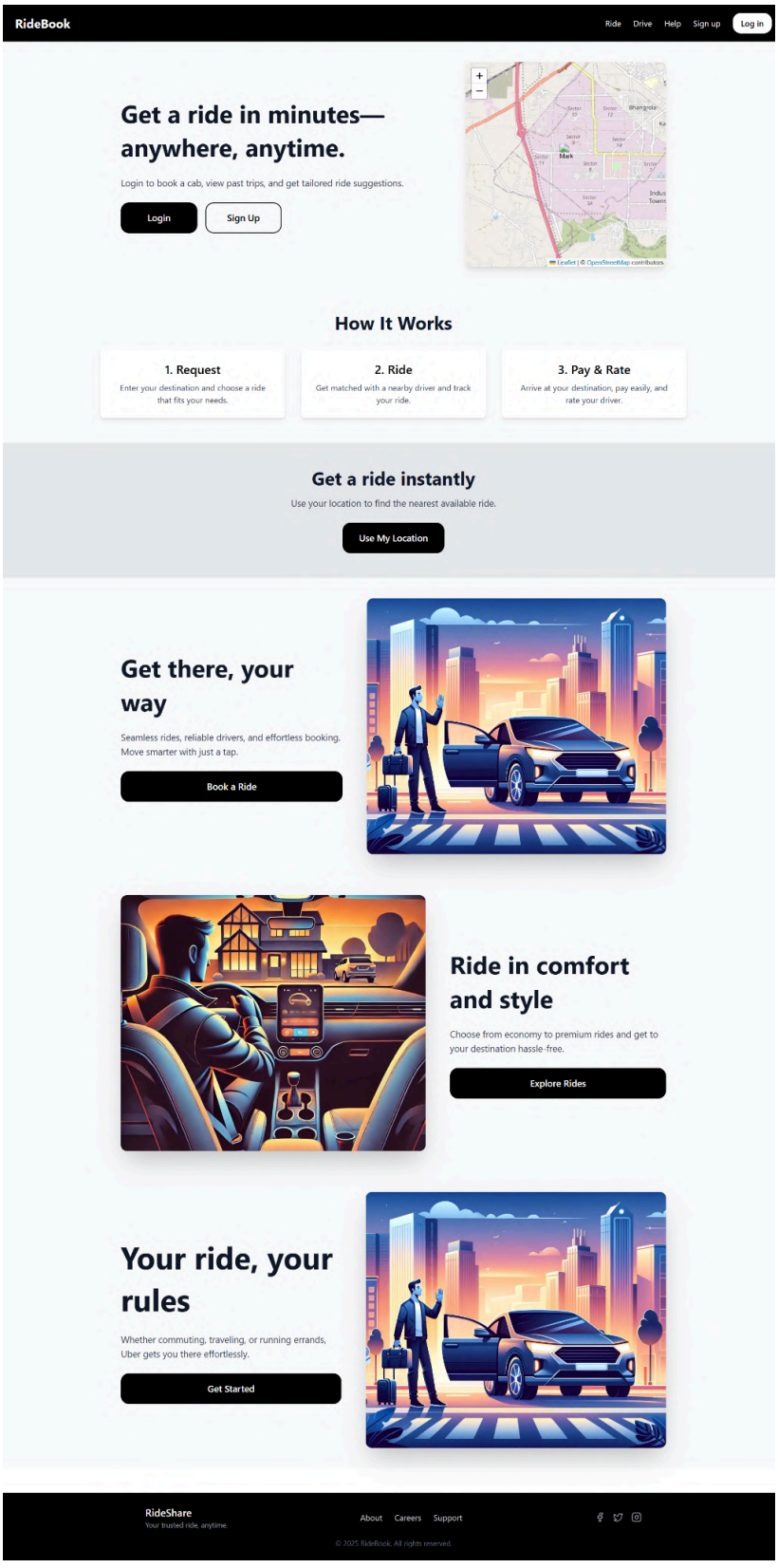
4.3 User Interface Design

The wireframe defined governs the user interface and aims to be as aesthetically pleasing as possible while also capturing the functionality of the web-app service.

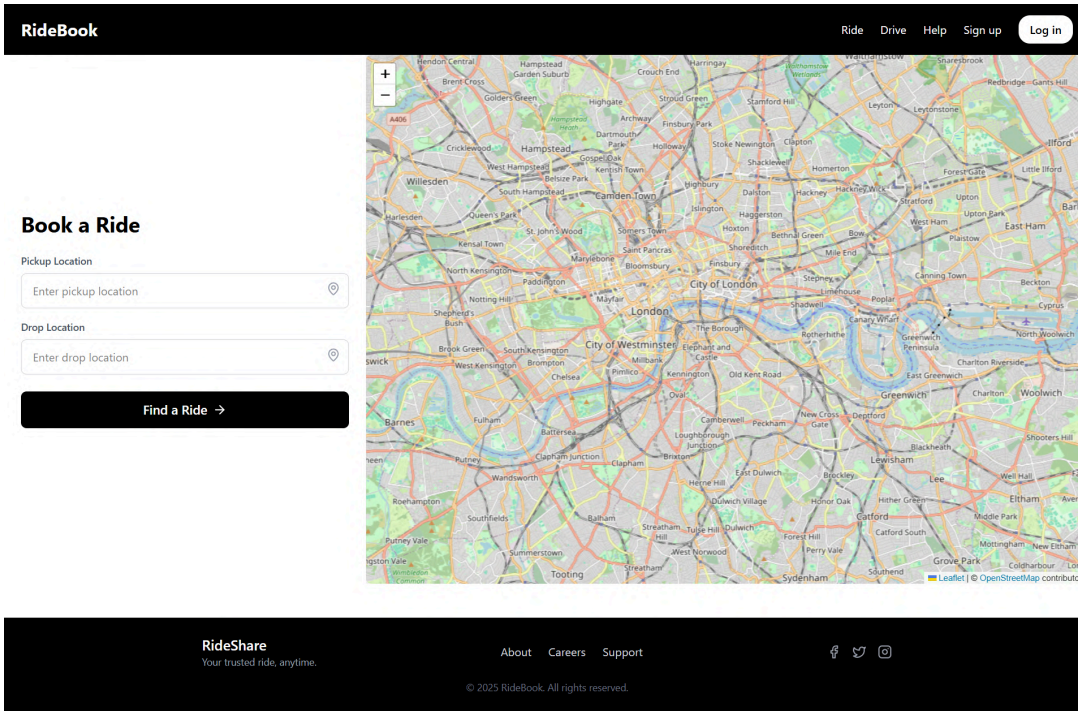
The Login and Sign-up Page:

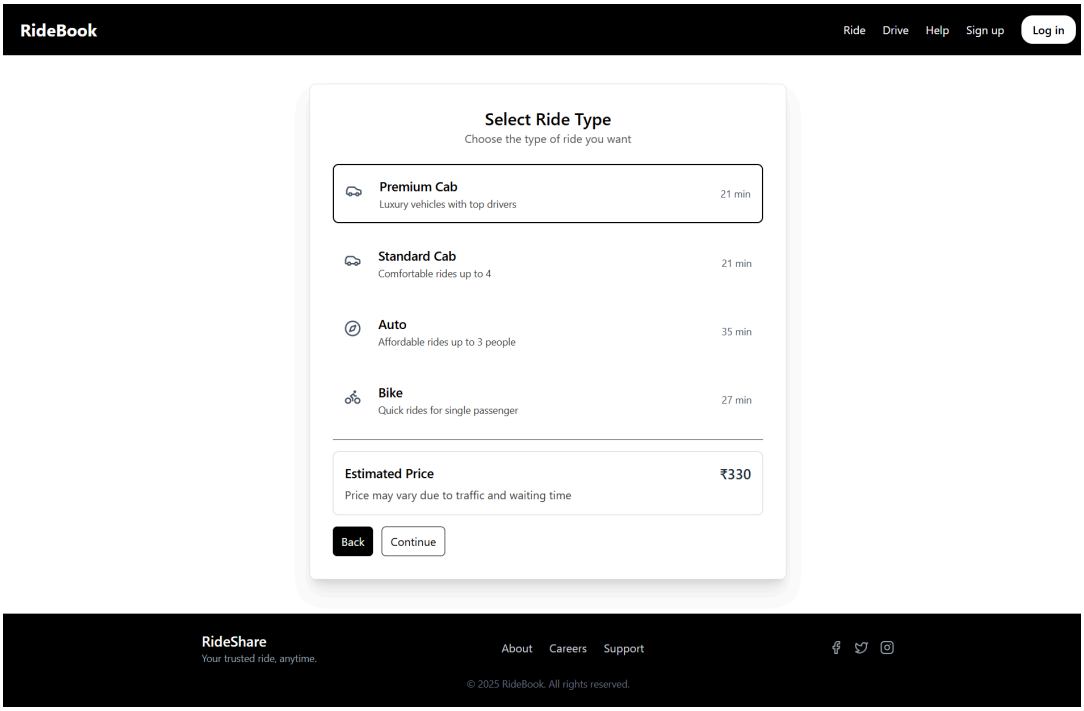


The Landing Page:

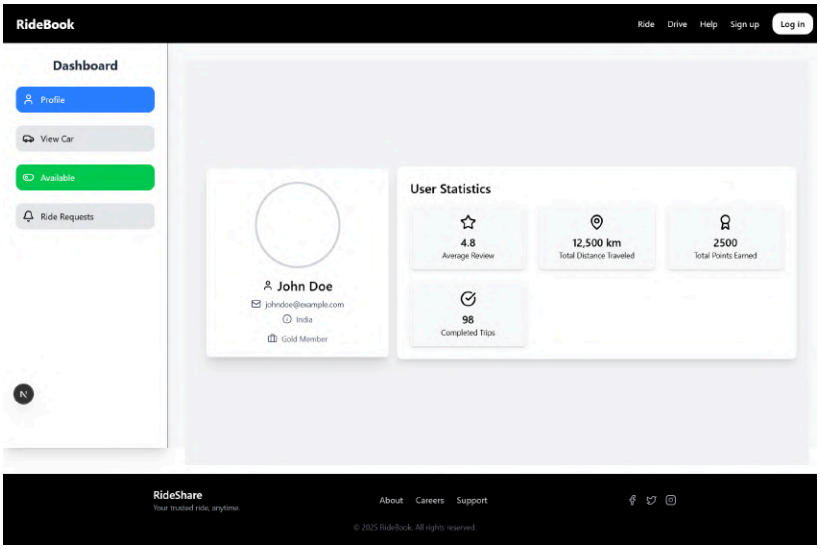


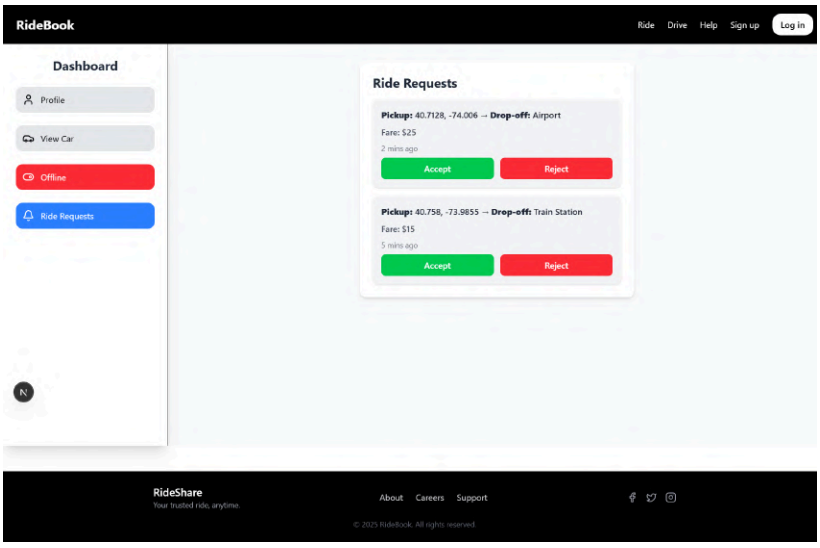
The Booking and Vehicle Selection Page:

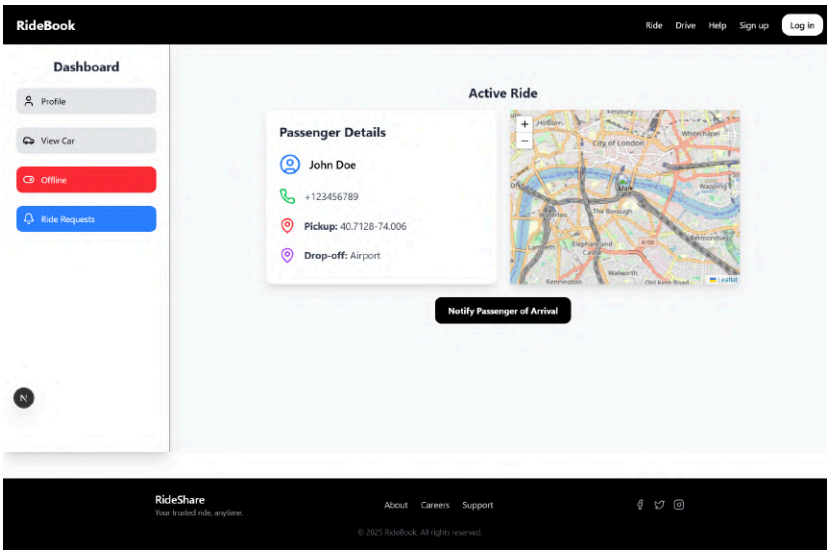




The Driver’s Dashboard:



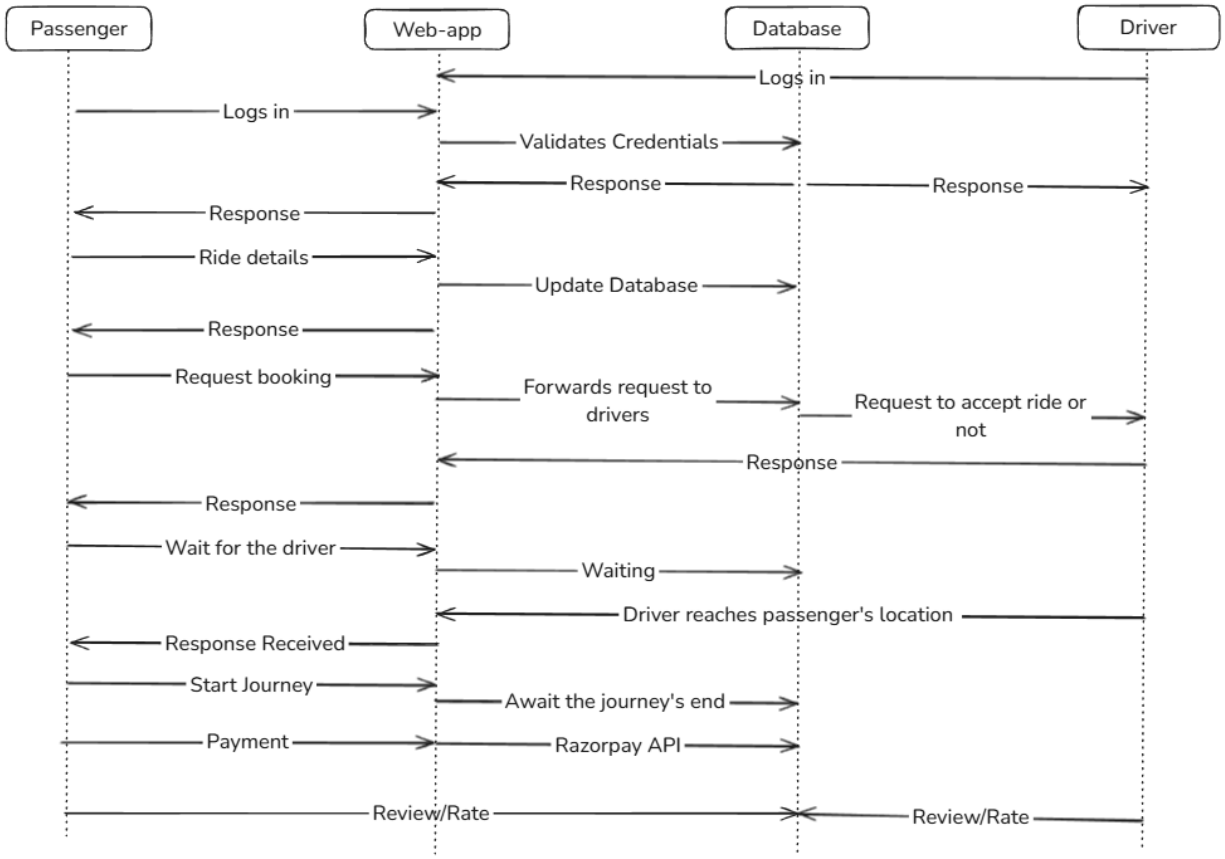




5 Data Flow and Interaction

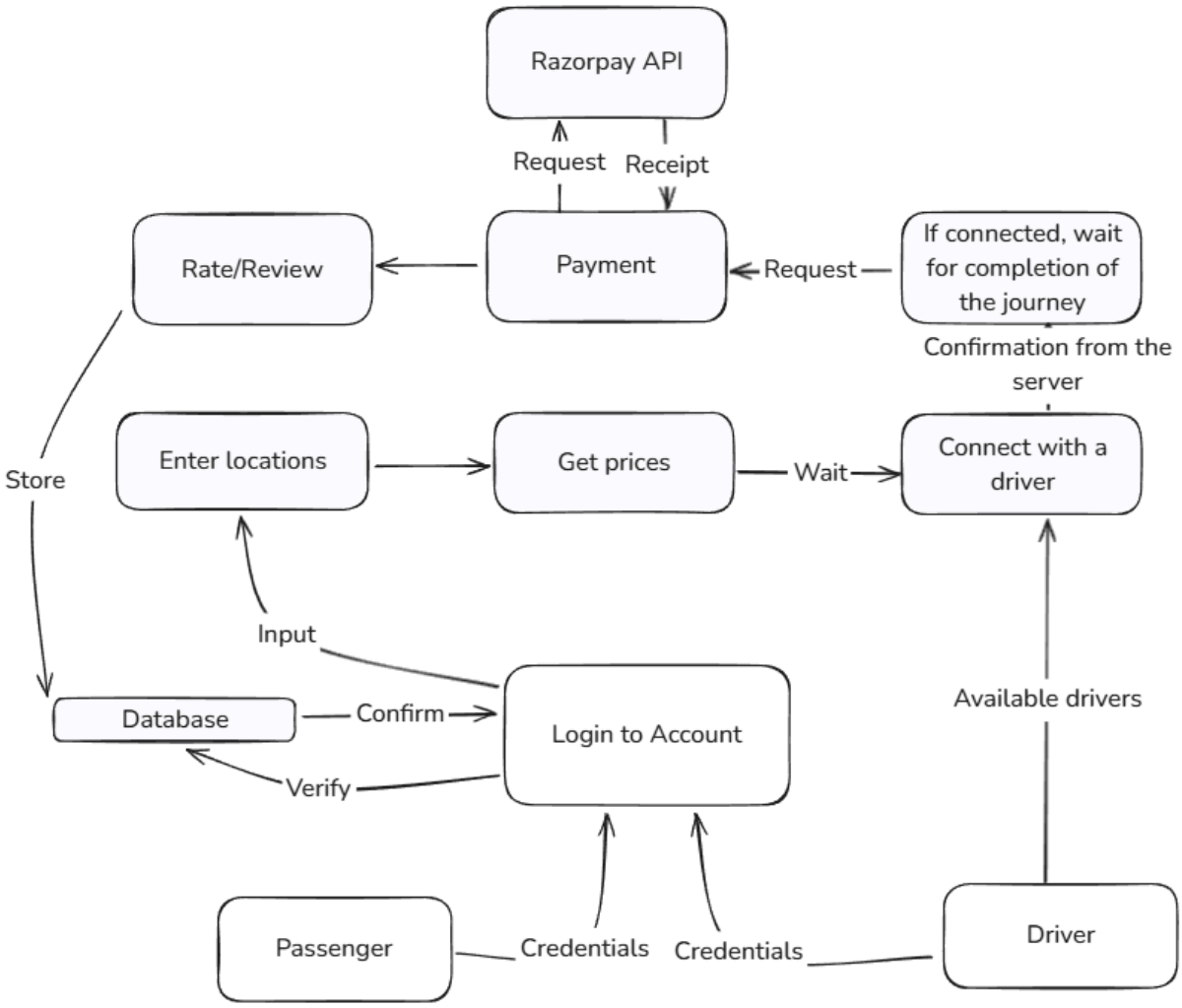
5.1 Sequence Diagram:

A Diagram illustrating the interactions between the various objects in a sequential manner. The diagram is meant to highlight the communications between these objects over time.



5.2 Data Flow Diagram

This diagram illustrates the flow of data within the system. A graph with nodes for events/states is used to describe the service.



6 Security and Performance Considerations

6.1 Authentication and Authorization

- **Role based access:**
  - Depending on who the user signs in as, the system redirects them to the relevant dashboard. Admins are authorized to have full control over the web-app whereas drivers and passengers have different dashboards/interfaces with limited access and control that they can interact with.
- **JWT-based authentication:**
  - Used for authentication and it does so by the exchange of information between the client and the server using tokens.
- **Protection against SQL Injections.**

6.2 Data & Protection

- **Password Protection:**
  - User passwords are hashed using an existing hashing algorithm as an extra measure taken against perpetrators.

6.3 Performance Requirements

Compartmentalization is key to ensuring there are no single points of failure. Even with the design choices we've made, ensuring that the user enjoys a seamless experience requires quick responses from the servers that serve these requests. We aim to have a response time of less than 3 seconds for the hefty API calls and ensure a proper reject/accept response from the drivers to the passengers within 5 minutes of the booking, at the latest.

7 Deployment

7.1 Hosting and Cloud Services

- **Frontend:** Deployed on Vercel
- **Backend:** Hosted on Azure
- **Database:** Hosted on Azure SQL