RIDE-EASE – CAB BOOKING APP

A PROJECT REPORT

Submitted by

Yash Porwal 22BCE10332

Yuvraj Singh 22BCE10801

Bharti Jayprakash 22BCE11460

Lakshay Srivastava 22BCE10021

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRIKALAN, SEHORE

MADHYA PRADESH - 466114

APRIL 2025

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Project Title: RideEase Cab Booking App

The RideEase Cab Booking App is a comprehensive and modern solution designed to simplify the process of booking cabs. Built using the MERN stack (MongoDB, Express.js, React, and Node.js), this application provides users with an intuitive interface to book rides, track their trips in real-time, and make secure payments. The app is tailored to meet the needs of both passengers and drivers, ensuring a seamless and efficient transportation experience.

The primary goal of this project is to eliminate the challenges associated with traditional cab services by introducing features like real-time tracking, flexible ride options, driver ratings, and a user-friendly interface. By leveraging the power of modern web technologies, RideEase aims to redefine the way people commute.

1.2 Team Members and Roles

This project is a collaborative effort undertaken by a team of four members. Each member has been assigned specific responsibilities based on their expertise to ensure the successful completion of the project.

Yash Porwal – Frontend Developer (User Interface)

Yash is responsible for designing and developing the user interface of the RideEase application. His role involves translating UI/UX designs into interactive, responsive web pages using modern frontend technologies like React.js. He ensures that the app is user-friendly, visually appealing, and performs efficiently across all devices. Yash collaborates closely with backend developers to integrate APIs and optimize the overall user experience, making sure that users can easily navigate, book rides, and access all features seamlessly.

Yuvraj Singh – Backend Developer (User Management & Core Logic)

Yuvraj handles the core backend development for the RideEase platform. His responsibilities include creating and maintaining RESTful APIs, managing user authentication, ride booking logic, real-time driver tracking, and payment processing. Yuvraj ensures that the server-side infrastructure is secure, scalable, and reliable. He works with the database to store and retrieve user data, ride details, and transaction records, ensuring smooth and efficient backend operations that support the frontend interface.

Bharti Jayprakash – Backend Developer (Driver/Captain & Admin Management)

Bharti is responsible for developing the backend modules related to driver management, driver verification, ride assignment, and admin functionalities. Her role involves building secure APIs for driver registration, document verification, shift management, and earnings tracking. Bharti ensures that driver-related data is accurately stored and managed, and that real-time communication between drivers and the platform is seamless. She also works on implementing safety features like driver verification and trip sharing, contributing to the overall security and reliability of the system.

Lakshya Srivastava – Frontend Developer (Driver/Captain App)

Lakshya specialises in developing the driver/captain interface of RideEase. His responsibilities include creating an intuitive and responsive app for drivers using React Native or similar frameworks. Lakshya ensures that drivers can accept or reject ride requests, view real-time navigation, update their availability status, and track earnings. His focus is on delivering a smooth, easy-to-use experience that helps drivers manage their rides efficiently while maintaining high performance and responsiveness across devices.

CHAPTER 2

PROJECT OVERVIEW

2.1 Purpose and Goals

The RideEase Cab Booking App is designed to provide a seamless and efficient solution for urban transportation. Its primary purpose is to bridge the gap between passengers and drivers, offering a platform that simplifies the process of booking rides, tracking trips, and making secure payments. By leveraging modern technologies, the app aims to enhance the user experience while addressing common challenges associated with traditional taxi services.

The goals of RideEase include:

- **Convenience:** To enable users to book rides effortlessly through an intuitive interface.
- **Efficiency:** To optimize ride allocation and reduce wait times by connecting passengers with nearby drivers in real-time.
- **Transparency:** To provide detailed ride information, including estimated fares, driver details, and real-time tracking.
- **Security:** To ensure safe transactions and data protection for both passengers and drivers.
- **Scalability:** To create a robust platform capable of handling a growing user base and expanding features.

RideEase is not just a cab booking app; it is a comprehensive transportation solution that adapts to the needs of modern commuters, whether they are traveling for work, leisure, or emergencies.

2.2 Key Features and Functionalities

The RideEase Cab Booking App incorporates several key features that make it stand out in the competitive ride-hailing market:

1. User Registration and Authentication:

- Users can securely register accounts using email or phone numbers.
- Authentication ensures that only verified users can access the platform.

2. Ride Booking Options:

- Passengers can book rides by selecting pickup and destination locations.
- Flexible options allow users to schedule rides for immediate use or at a later time.

3. Real-Time GPS Tracking:

- The app integrates GPS technology to enable live tracking of rides.
- Both passengers and drivers can view estimated arrival times and current locations.

4. Driver information:

- Passengers can access detailed profiles of drivers, including names, ratings, vehicle details, and photos.
- This feature builds trust between users and drivers.

5. Payment Integration:

- Multiple payment options such as credit cards, e-wallets, and cash are supported.

- Secure payment gateways ensure hassle-free transactions.

6. Ride History:

- Users can view past bookings with details like pickup/drop-off locations, fare amounts, and timestamps.

- This feature helps users track their travel expenses.

7. Push Notifications:

- Notifications keep users informed about ride status updates such as driver arrival, ride start, payment confirmation, etc.

8. Admin Panel for Management:

- Administrators can monitor bookings, manage drivers/passengers, adjust pricing dynamically, and resolve complaints efficiently.

These features collectively ensure a smooth experience for passengers while empowering drivers with tools to manage their rides effectively.

2.3 Target Users

The RideEase Cab Booking App caters to three primary user groups:

1. Passengers (Users):

   Passengers are the main target audience who use the app to book rides conveniently. They benefit from features like real-time tracking, flexible payment options, driver ratings, and ride history. Whether commuting daily or traveling occasionally, RideEase ensures an effortless experience for all passengers.

2. Drivers:

   Drivers represent the second critical user base of RideEase. The app provides them with tools to

accept or decline ride requests, navigate routes using GPS, track earnings, and communicate with passengers seamlessly. By optimizing ride allocation and reducing idle time, RideEase helps drivers maximize their revenue while maintaining high service standards.

3. Administrators (Taxi Service Providers):

   Administrators oversee the platform's operations through an advanced admin panel. They manage driver registrations, monitor passenger activity, adjust pricing based on demand patterns (e.g., surge pricing), handle complaints or disputes, and generate analytics reports to improve service quality.

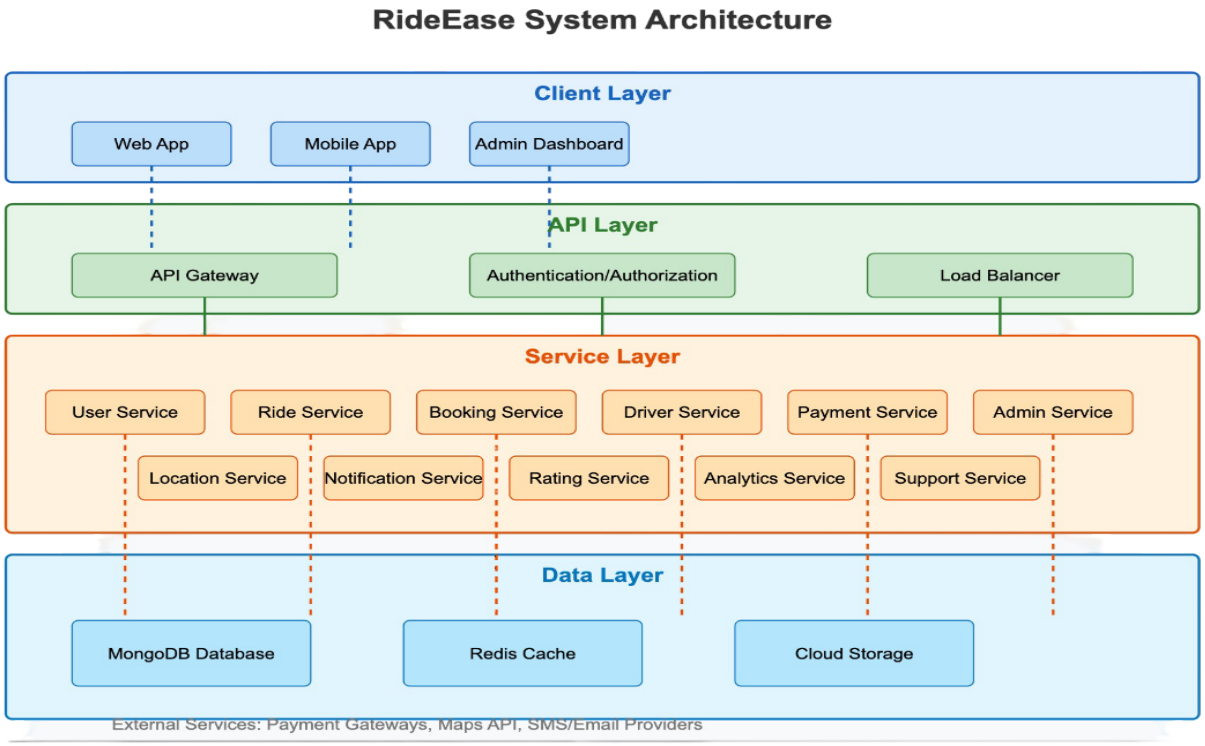
By addressing the unique needs of each user group-passengers seeking convenience, drivers aiming for efficiency, and administrators ensuring smooth operations-RideEase creates a balanced ecosystem that enhances urban mobility for all stakeholders.

CHAPTER 3

ARCHITECTURE

3.1 System Architecture Overview

The RideEase Cab Booking App is built using the MERN stack, which consists of MongoDB, Express.js, React, and Node.js. This architecture is designed to provide a seamless and scalable system for both passengers and drivers. The system follows a three-tier architecture: the frontend, backend, and database layers, all of which communicate through RESTful APIs. The frontend is responsible for rendering the user interface and handling user interactions. It communicates with the backend to send requests and receive responses. The backend, built using Node.js and Express.js, acts as the business logic layer, processing requests, managing data flow, and interacting with the database. MongoDB serves as the database layer, storing all data related to users, drivers, rides, and payments.



3.2 Frontend Architecture (React)

The frontend of RideEase is developed using React.js, a popular JavaScript library for building user interfaces. React's component-based architecture allows for reusable Ul elements, making development faster and more efficient. The frontend is designed to be responsive and user-friendly, ensuring a smooth experience across different devices.

3.2.1 Component Structure

The component structure in React is organized hierarchically to ensure clarity and reusability. At the top level are layout components such as Header, Footer, and Sidebar, which provide a consistent structure across all pages. Below these are core page components like LoginPage, BookingPage, TrackingPage, and HistoryPage. Each page component is further divided into smaller functional components to handle specific tasks. For example, the BookingPage contains components for selecting pickup and drop-off locations, viewing available rides, and confirming bookings. This hierarchical structure ensures that each component has a single responsibility, making it easier to debug and maintain. Components are also designed to be reusable across different parts of the application. For instance, a Button component used in the login page can also be reused in the booking confirmation page.

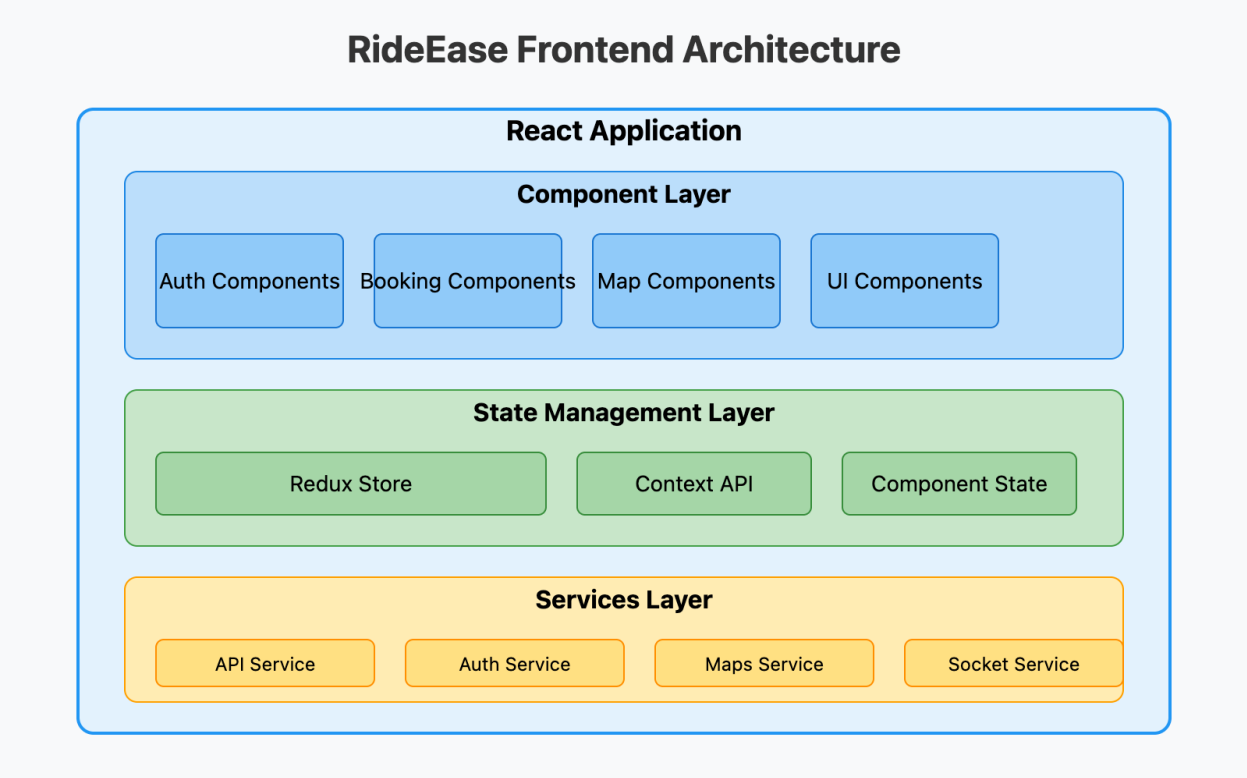
3.2.2 State Management

State management in RideEase is handled using Redux Toolkit, which centralizes application state in a single store. This approach simplifies data flow between components by allowing them to access shared state directly from the store rather than passing props down through multiple levels of components. Redux slices are used to manage different aspects of the application's state. For example, an authSlice manages user authentication data such as login status and tokens, while a bookingSlice tracks ride details like pickup location, destination, fare estimates, and driver information. By organizing state into slices, the application maintains a clear separation of concerns.

3.2.3 Ul Libraries

The frontend leverages Material-Ul for styling and pre-built components such as buttons, modals, grids, and forms. Material-Ul ensures that the application has a professional look while reducing development time by providing ready-to-use design elements. For navigation between pages, React Router is used to implement client-side routing seamlessly. Additionally, Mapbox GL JS is integrated into the frontend to

provide real-time maps for ride tracking and route visualization.



3.3 Backend Architecture (Node.js and Express.js)

The backend of RideEase is developed using Node.js with Express.js as the web framework. It serves as the core application logic layer that processes client requests, interacts with the database, and returns appropriate responses through RESTful APIs.

3.3.1 API Gateway

The API Gateway acts as the central entry point for all client requests coming from the frontend or other external systems (e.g., mobile apps). It routes these requests to specific microservices based on their functionality-for example; authentication requests are forwarded to the Authentication Service while booking-related requests are directed to the Booking Service.

Express.js routers are used to define routes specific to each functionality in a modular manner.

For instance:

a. /api/auth handles authentication-related endpoints such as login or signup.

b. /api/booking manages ride booking endpoints like creating or canceling bookings. This modular routing approach ensures that each service operates independently while maintaining a cohesive structure.

3.3.2 Authentication Service

User authentication in RideEase is implemented using JSON Web Tokens (JWT). When users log in or register successfully, a JWT token is generated containing their unique identifier (e.g., user ID). This token is then sent back to the client and stored securely (e.g., in local storage). For every subsequent request requiring authentication (e.g., accessing ride history), this token is sent along with the request headers for verification.

Password security is ensured using bcrypt hashing before storing passwords in MongoDB This prevents plaintext passwords from being exposed even if there's unauthorized access to the database.

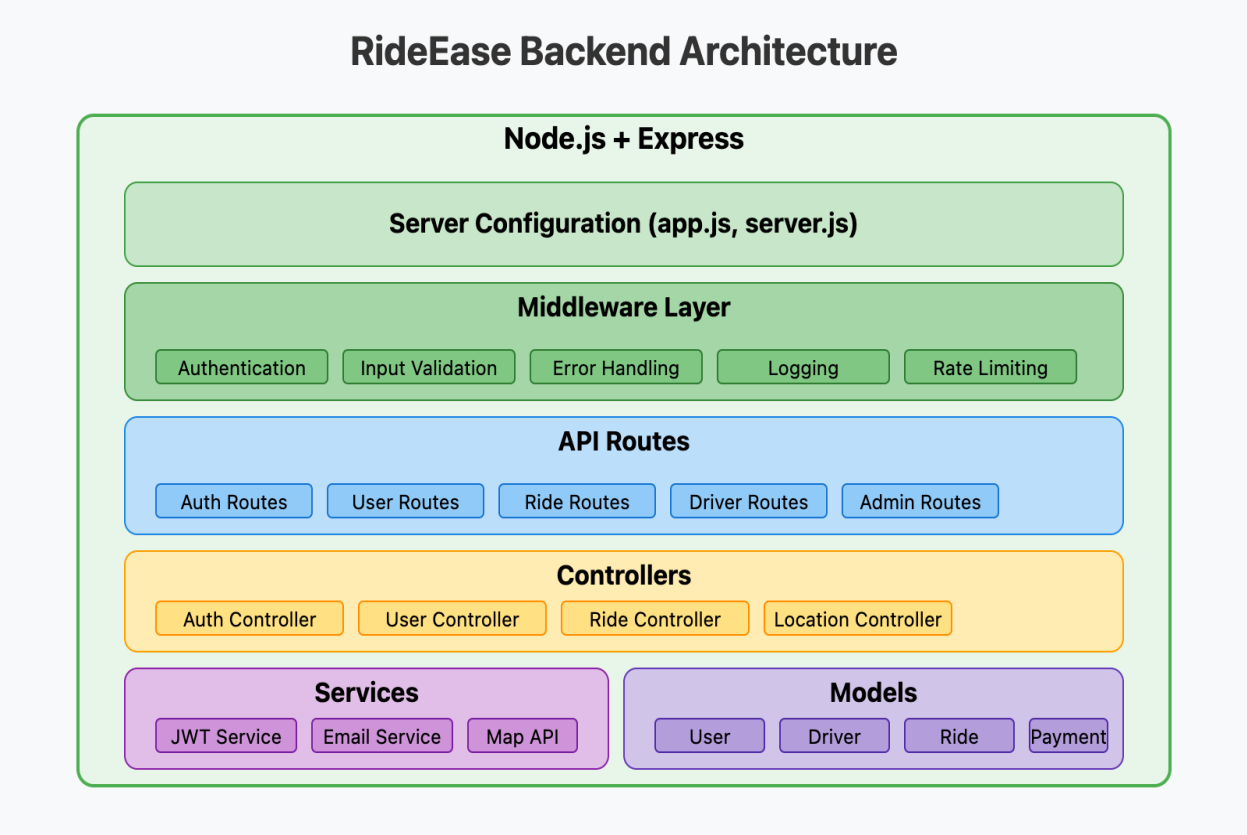
3.3.3 Booking Service

The Booking Service handles all operations related to ride bookings-from validating pickup/drop- off locations to assigning drivers based on proximity or availability status. It integrates with third- party APls like Mapbox for calculating optimal routes between locations.
When a user confirms a booking request:
1. The service checks for available drivers near their pickup location.
2. A driver is assigned based on factors like distance from pickup point or driver rating.
3. Fare estimates are calculated dynamically based on distance traveled or time taken.

3.3.4 Driver Management Service

This service manages driver-related functionalities such as updating driver availability status or fetching nearby drivers during ride allocation processes.

Real-time updates regarding driver location are pushed via WebSocket connections so passengers can track their rides accurately during transit periods.



3.4 Database Architecture (MongoDB)

MongoDB serves as RideEase's database layer due to its flexibility in handling unstructured data formats required by modern applications like cab booking systems.

3.4.1 Schema Design

The database schema consists of multiple collections:

- A users collection stores details about passengers including their name email address password hash etc.
- A drivers collection tracks information about registered drivers including vehicle details ratings availability status etc
- A rides collection logs trip-specific information such as pickup/drop-off coordinates timestamps fare amounts assigned driver IDs etc

- A payments collection records transaction details associated with completed trips

3.4.2 Data Relationships

Relationships between entities are modeled using references:

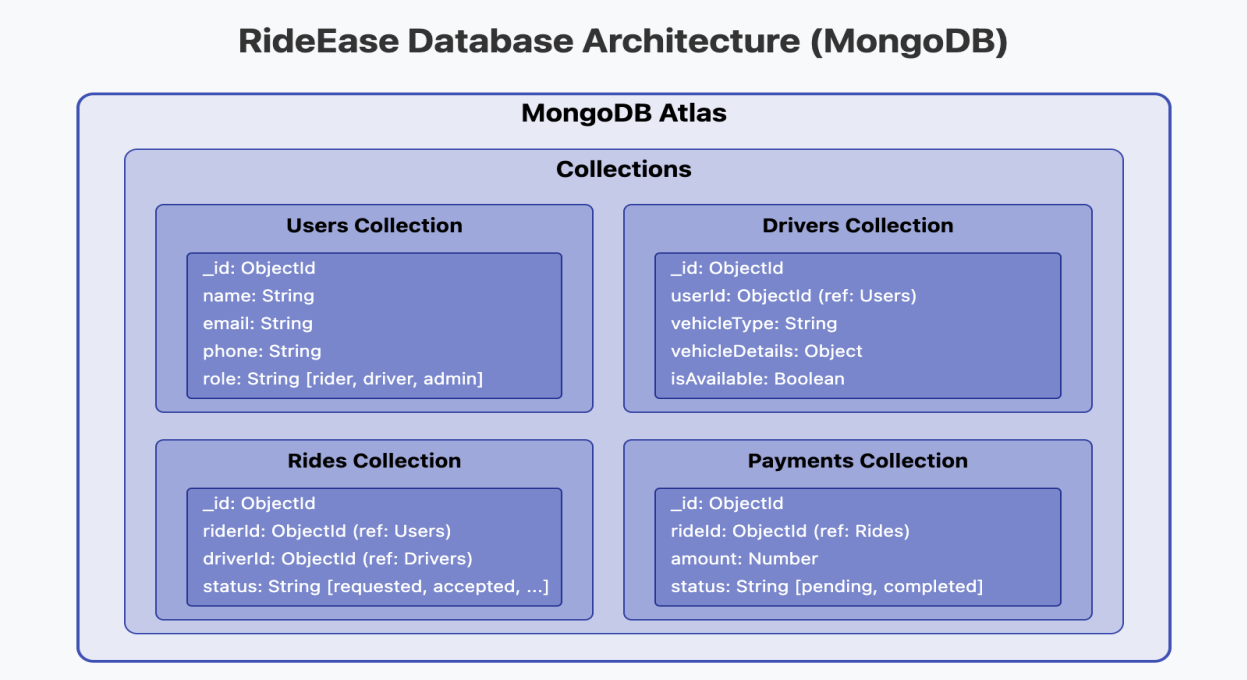
- Each ride document references both its passenger (userID) & assigned driver (driverID)
- Payment documents embed transaction-specific fields directly within respective ride documents ensuring faster query performance during payment history lookups

3.4.3 Indexing Strategy

Indexes play critical roles optimizing query performance especially when dealing large-scale datasets common scenarios include:

1) Geospatial indexes applied fields storing pickup/drop coordinates enabling efficient searches nearby drivers

2) Compound indexes combining frequently queried fields timestamps user IDs facilitating rapid retrieval historical trip records



CHAPTER 4

SETUP INSTRUCTIONS

4.1 Prerequisites and Dependencies

Before setting up the RideEase Cab Booking App, developers need to ensure that their systems meet the necessary prerequisites and have the required dependencies installed. The application is built on the MERN stack, which requires specific software components to function properly.

First, developers need to have Nodejs installed on their systems. Node.js version 16.0.0 or higher is recommended to ensure compatibility with all the dependencies used in the project. Node.js comes with nom (Node Package Manager), which is essential for installing and managing the project dependencies.

MongoDB is another critical prerequisite. Developers can either install MongoDB locally on their machines or use MongoDB Atlas, a cloud-based database service. For local development, MongoDB Community Edition version 5.0 or higher is recommended. If using MongoDB Atlas, developers will need to create an account and set up a cluster.

Git is required for version control and for cloning the repository from GitHub or any other version control platform where the project is hosted. Additionally, a code editor such as Visual Studio Code, Sublime Text, or WebStorm is recommended for efficient development.

The backend of RideEase depends on several pm packages, as listed in the package.json file:

- Express.js (v4.21.1): A web application framework for Node.js, used to build the API.
- Mongoose (v8.8.2): An Object Data Modeling (ODM) library for MongoDB and Node.js.
- bcrypt (v5.1.1): A library for hashing passwords to ensure security.
- jsonwebtoken (v9.0.2): Used for implementing JWT-based authentication.

- dotenv (v16.4.5): Loads environment variables from a env file into process.env.
- cors (v2.8.5): Enables Cross-Origin Resource Sharing (CORS).
- cookie-parser (v1.4.7): Parses Cookie header and populates req.cookies with an object keyed by the cookie names.
- express-validator (v7.2.0): A set of express.js middlewares for validation sanitization.
- axios (v1.7.7): A promise-based HTTP client for making API requests.

The frontend of RideEase is built using React and depends on several packages:

- React (v18.3.1) and React DOM (v18.3.1): Core libraries for building the user interface.
- React Router DOM (v6.28.0): For handling routing within the React application.
- Axios (v1.7.7): For making HTTP requests to the backend API.
- @react-google-maps/api (v2.20.3): For integrating Google Maps into the application.
- socket.io-client (v4.8.1): For real-time communication between the client and server.
- GSAP (v3.12.5) and @gsap/react (v2.1.1): For creating animations.
- remixicon (v4.5.0): For using icons in the Ul.

Development dependencies include:

- Vite (v5.4.10): A build tool that provides a faster development experience.
- ESLint (v9.13.0): For linting JavaScript code.
- Tailwind CSS (v3.4.15): A utility-first CSS framework for styling.

- PostCSS (v8.4.49) and Autoprefixer (v10.4.20): For processing CSS.

4.2 Environment Setup

Setting up the development environment for RideEase involves configuring both the local development machine and the project-specific environment variables.

First, developers need to install Node.js and npm. They can download the latest LTS version from the official Node.js website (https://nodejs.org/) and follow the installation instructions for their operating system. After installation, they can verify the installation by running the following commands in the terminal:

- node --version
- npm –version

Next, they need to set up MongoDB. For local development, they can download MongoDB Community Edition from the official MongoDB website (https://www.mongodb.com/try/download/community) and follow the installation instructions. Alternatively, they can set up a MongoDB Atlas cluster by creating an account on the MongoDB Atlas website (https://www.mongodb.com/cloud/atlas) and following the setup wizard to create a new cluster.

For the project itself, developers need to clone the repository from the version control system. Assuming the project is hosted on GitHub, they can use the following command:

- git clone https://github.com/yourusername/rideease.git
- cd rideease

The project follows a monorepo structure with separate directories for the frontend and backend. Developers need to set up both parts of the application.

4.3 Installation Steps

After cloning the repository, developers need to install the dependencies for both the frontend and backend parts of the application. The installation process involves using npm to install all the required packages listed in the respective package.json files.

For the backend:

- cd backend
- npm install

This command installs all the dependencies listed in the backend's package.json file, including Express.js, Mongoose, bcrypt, jsonwebtoken, and others.

For the frontend:

- cd ../frontend
- npm install

This command installs all the dependencies listed in the frontend's package.json file, including React, React Router DOM, Axios, and others.

After installing the dependencies, developers need to build the frontend application for production:

- npm run build

This command creates a production-ready build of the frontend application, which can be served by the backend server.

4.4 Configuration (.env setup)

Both the frontend and backend parts of RideEase require environment variables to be set up correctly. These variables include database connection strings, API keys, JWT secrets, and other configuration parameters that should not be hardcoded into the source code. For the backend, developers need to create

a env file in the backend directory with the following variables:

For the backend, developers need to create a env file in the backend directory with the following variables:

```
# Server Configuration
PORT=5000
NODE_ENV=development

# MongoDB Connection
MONGODB_URI=mongodb://localhost:27017/rideease
# or for MongoDB Atlas
# MONGODB_URI=mongodb+srv://<username>:<password>@cluster0.mongodb.net/rideease

# JWT Configuration
JWT_SECRET=your_jwt_secret_key
JWT_EXPIRES_IN=7d

# Google Maps API Key
GOOGLE_MAPS_API_KEY=your_google_maps_api_key

# Payment Gateway Configuration (if applicable)
STRIPE_SECRET_KEY=your_stripe_secret_key
```

For the frontend, developers need to create a .env file in the frontend directory with the following variables:

```
VITE_API_URL=http://localhost:5000/api
VITE_GOOGLE_MAPS_API_KEY=your_google_maps_api_key
VITE_SOCKET_URL=http://localhost:5000
```

It's important to note that environment variables in the frontend should be prefixed with VITE_ for Vite to expose them to the client-side code.

After setting up the environment variables, developers can start the development servers:

For the backend:

- cd backend
- npm start

For the frontend:

- cd frontend
- npm run dev

The backend server will run on the port specified in the env file (default: 5000), and the frontend development server will run on port 3000 by default. Developers can access the application by navigating to http://localhost:3000 in their web browser.

By following these setup instructions, developers can quickly get the RideEase Cab Booking App up and running in their local development environment, ready for further development or testing.

CHAPTER 5

FOLDER STRUCTURE

5.1 Client: Structure of the React Frontend

The frontend of the RideEase project is organized to maximize modularity, maintainability, and scalability, following modern React best practices. The main directory is named frontend, and its core logic resides in the src folder.

Within src, the **components** directory contains all reusable UI elements and feature-specific components. For example, CaptainDetails.jsx, ConfirmRidePopup.jsx, FinishRide.jsx, and LiveTracking.jsx each represent encapsulated pieces of functionality, such as displaying captain (driver) information, handling ride confirmations, finalizing rides, and providing live ride tracking, respectively. This modular approach allows for easy updates and code reuse across different parts of the application.

The **context** directory is used for React Context providers, which manage global state and shared logic. Files like CaptainContext.jsx, SocketContext.jsx, and UserContext.jsx provide context for captain/driver data, real-time socket connections, and user authentication or preferences. This structure enables efficient state management and cleaner component hierarchies.

The **pages** directory contains the main page components that correspond to the application's routes. Examples include Home.jsx for the landing page, UserLogin.jsx and CaptainLogin.jsx for authentication interfaces, and Riding.jsx for the active ride experience. Each page acts as a container, assembling the necessary components and contexts to deliver complete user or driver flows.

The root of the src directory also includes App.jsx, the main application component where routes and global providers are configured. This ensures a single entry point for rendering the entire React app.

Outside of src, the **public** directory holds static assets such as images and SVGs, including the project logo (rideease-logo.png) and Vite's default SVG. The project root contains configuration files like index.html, package.json (for dependency management), vite.config.js (for Vite bundler settings), and tailwind.config.js (for Tailwind CSS customization).

This structure ensures that the frontend codebase is clean, organized, and ready for collaborative development and future expansion.

5.2 Server: Organization of the Node.js Backend

The backend of RideEase is structured for clarity, modularity, and extensibility, following industry standards for Node.js and Express applications. The main directory is named backend, with the core logic encapsulated in the src folder.

Within src, the **controllers** directory contains business logic for handling HTTP requests. Files such as user.controller.js, captain.controller.js, maps.controller.js, and ride.controller.js are responsible for processing incoming requests, interacting with services and models, and sending appropriate responses for each domain area (users, captains, maps, rides).

The **models** directory defines the Mongoose schemas and data models for MongoDB. Files like user.model.js, captain.model.js, ride.model.js, and blackListToken.model.js represent the structure and validation rules for each data entity, ensuring data integrity and consistency.

The **routes** directory maps API endpoints to their corresponding controller functions. For example, user.routes.js, captain.routes.js, maps.routes.js, and ride.routes.js define the RESTful routes for each major feature, keeping the routing logic separate from business logic.

The **services** directory encapsulates complex business processes and external integrations. For instance, maps.service.js might handle communication with mapping APIs, while ride.service.js manages ride allocation and status updates.

The **middleware** directory includes reusable middleware functions such as auth.middleware.js for authentication and authorization, ensuring that protected routes are only accessible to authorized users.

The **utils** directory provides utility classes and functions, such as ApiError.js and ApiResponse.js for standardized error and response handling, and asyncHandler.js for managing asynchronous route handlers.

Database connection logic resides in the **db** directory, specifically in db.js, which establishes and manages the connection to MongoDB.

At the root of the backend, app.js sets up the Express application, including middleware, routes, and error

handling. server.js serves as the main entry point for starting the backend server, while socket.js handles real-time communication via WebSockets. The backend also includes package.json for dependency management and README.endpoints.md for API documentation.

This backend structure supports clean separation of concerns, scalability, and ease of maintenance, allowing different team members to work independently on various features without conflict.

CHAPTER 6

RUNNING THE APPLICATION

Running the RideEase Cab Booking App involves starting both the frontend and backend servers. This chapter provides detailed instructions on how to run the application in different environments, including development, testing, and production.

Development Environment Setup

Before running the application, ensure that you have completed all the installation steps outlined in Chapter 4. This includes installing all dependencies and configuring the environment variables correctly.

To run the application in development mode, you need to start both the frontend and backend servers. The development mode provides features like hot reloading, which automatically refreshes the application when code changes are detected, making the development process more efficient.

Starting the Backend Server:

Navigate to the backend directory in your terminal and run the following command:

```
cd backend
 nodemon start
or
node start server.js
```

This command starts the Node.js server using nodemon, which automatically restarts the server when file changes are detected. The server will run on the port specified in your .env file (default is 5000).

You should see output similar to the following, indicating that the server is running successfully:

```
[nodemon] 2.0.22 - monitoring for changes...
 Server running on port 4000
 Connected to MongoDB
```

Starting the Frontend Server:

In a new terminal window, navigate to the frontend directory and run the following command:

```
cd frontend
 npm run dev
```

This command starts the Vite development server, which serves the React application. The server typically runs on port 3000 by default, but Vite may choose a different port if 3000 is already in use.

You should see output similar to the following:

```
 VITE v5.4.10  ready in 1234 ms

➜ Local:   http://localhost:5173/
➜ Network: http://192.168.1.x:5173/
```

After running these commands, you can access the application by opening your browser and navigating to http://localhost:5173.

Production Deployment

For production deployment, you need to build the frontend application and configure the backend to serve the static files.

Building the Frontend:

```
cd frontend
 npm run build
```

This command creates a production-ready build of the React application in the dist directory. The build process optimizes the code for performance, including minification and tree-shaking to reduce the bundle size.

Configuring the Backend to Serve Static Files:

In your app.js or server.js file in the backend directory, add the following code to serve the static files

from the frontend build:

```
const path = require('path');

// Serve static files from the React frontend app
app.use(express.static(path.join(__dirname, '../frontend/dist')));

// Anything that doesn't match the above, send back index.html
app.get('*', (req, res) => {
 res.sendFile(path.join(__dirname, '../frontend/dist/index.html'));
});
```

Starting the Production Server:

```
cd backend
npm start
```

This command starts the Node.js server in production mode. In a production environment, you might want to use a process manager like PM2 to ensure that your application stays running:

```
npm install -g pm2
pm2 start server.js
```

Environment-Specific Configurations

The application uses different configurations based on the environment it's running in. The NODE_ENV environment variable determines which configuration to use:

- `development`: Used during local development.
- `test`: Used when running tests.
- `production`: Used in the production environment.

You can set the `NODE_ENV` variable when starting the servers:

```
# For development
NODE_ENV=development npm run dev
```

```
# For production
NODE_ENV=production npm start
```

Troubleshooting Common Issues

1. **Port Already in Use:**
   If you see an error like "Port 4000 is already in use", you can either:
   a. Close the application using that port.
   b. Change the port in your .env file.
2. **Database Connection Issues:**
   If the server fails to connect to MongoDB, check your connection string in the .env file and ensure that MongoDB is running.
3. **Missing Dependencies:**
   If you encounter errors about missing modules, run `npm install` in both the frontend and backend directories to ensure all dependencies are installed.
4. **Environment Variables Not Loaded:**
   Ensure that your .env files are properly set up in both the frontend and backend directories. For the frontend, remember that environment variables must be prefixed with `VITE_` to be accessible in the client-side code.

By following these instructions, you should be able to run the RideEase Cab Booking App successfully in both development and production environments.

CHAPTER 7

API DOCUMENTATION

The RideEase Cab Booking App exposes a comprehensive set of RESTful API endpoints that enable communication between the frontend and backend components. This documentation provides detailed information about the API endpoints available in the RideEase application. The APl is organized into four main sections: User, Captain, Maps, and Rides.

Authentication

Most endpoints require JWT authentication. Include the token in the request header:

Authorization: Bearer <jwt_token>

User Endpoints

1. Register User

Creates a new user account in the system.

Endpoint: POST /user/register

Request Body:

{

"fullname": {

"firstname": "Yuvraj",

"lastname": "Singh"

"email": "yuvraj@example.com",

"password": "password123"

Response: Returns user details and authentication token.

2. Login User

Authenticates existing user and provides access token.

Endpoint: POST / user/login

Request Body:

"email": "yuvraj@example.com",

"password": "password123"

Response: Returns user details and authentication token.

3. Get User Profile

Retrieves the profile information of the logged-in user.

Endpoint: GET /user/user-profile

Response: Returns user profile details.

4. User Logout

Invalidates the current user session.

Endpoint: GET /user/logout

Response: Confirms successful logout.

Captain Endpoints

1. Register Captain

Creates a new captain account with vehicle details.

Endpoint: POST / captain/register

Request Body:

"fullname": {

"firstname": "Bharti",

"lastname": "Jayprakash"

5,

"email": "bharti@example.com",

"password": "password123"

"vehicle": {

"color": "Black",

"plate": "ABC123",

"capacity": 4,

"vehicleType": "car"

Response: Returns captain details and authentication token.

2. Captain Login

Authenticates existing captain and provides access token.

Endpoint: POST / captain/login

Request Body:

{

"email": "bharti@example.com",

"password": "password123"

}

Response: Returns captain details and authentication token.

3. Get Captain Profile

Retrieves the profile information of the logged in captain.

Endpoint: GET / captain/profile

Response: Returns captain profile details.

4. Captain Logout

Invalidates the current captain session.

Endpoint: GET / captain/logout

Response: Confirms successful logout.

Maps Endpoints

1. Get Location Coordinates

Converts address to geographical coordinates.

Endpoint: GET /maps/get-coordinates?address=Example+Address

Response: Returns latitude and longitude.

2. Get Distance and Time

Calculates distance and estimated time between two locations.

Endpoint: GET /maps/get-distance-time?origin=Start+Location&destination=End+Location

Response: Returns distance and duration information.

3. Get Location Suggestions

Provides autocomplete suggestions for location search.

Endpoint: GET /maps/get-suggestions?input=search_text

Response: Returns array of location suggestions.

4. Get Ride Fare

Calculates estimated fare for different vehicle types.

Endpoint: GET /maps/qet-fare?pickup=Start+Location&destination=End+Location

Response: Returns fare estimates for different vehicle types.

Ride Endpoints

1. Create New Ride

Initiates a new ride request.

Endpoint: POST /rides/create

Request Body:

{

"pickup": "Start Location",

"destination": "End Location",

"vehicleType": "car"

}

Response: Returns ride details.

2. Confirm Ride

Allows captain to accept and confirm a ride.

Endpoint: POST /rides/confirm

Request Body:

{

"rideId": "ride_id_here"

}

Response: Returns confirmed ride details.

3. Verify OTP

Verifies ride start OTP provided by user.

Endpoint: POST /rides/verify-otp

Request Body:

{

"rideld": "ride_id_here",

"otp": "1234"

}

Response: Returns OTP verification status.

4. End Ride

Completes an ongoing ride.

Endpoint: POST /rides/end-ride

Request Body:

"rideId": "ride_id_here"

}

Response: Returns completed ride details.

Error Handling

All endpoints return appropriate HTTP status codes:

• 200: Success

• 400: Bad Request

• 401: Unauthorized

• 403: Forbidden

• 404: Not Found

• 500: Internal Server Error

Rate Limiting

API requests are limited to 100 requests per IP per hour.

Security

• All endpoints use HTTP

• Authentication required for protected routes

• Input validation implemented

• CORS enabled

CHAPTER 8

AUTHENTICATION

The RideEase Cab Booking App implements a robust authentication and authorization system to ensure secure access to the application's features and protect user data. The system utilizes a combination of session-based authentication and JSON Web Tokens (JWT) to manage user sessions and authorize access to protected resources.

Session-Based Authentication

RideEase primarily employs session-based authentication for its web application. This method is chosen for its simplicity, immediate invalidation capabilities, and suitability for browser-based interactions. Here's how it works:

1. **User Login**: When a user submits their credentials (username/email and password) through the login form, the server validates these credentials against the stored user information in the database.

2. **Session Creation**: Upon successful validation, the server creates a new session for the user. This session contains essential information such as the user's ID, role (e.g., customer, driver, or admin), and any other relevant data needed for the application's functionality.

3. **Session Storage**: The session data is stored server-side, typically in a database like MongoDB or in-memory storage solutions like Redis for faster access. This approach enhances security by keeping sensitive session data off the client-side.

4. **Session ID**: A unique session ID is generated and sent back to the client. This ID is stored as an HTTP-only cookie in the user's browser, making it inaccessible to client-side scripts and thus more secure against XSS attacks.

5. **Subsequent Requests**: For each subsequent request, the browser automatically includes the session ID cookie. The server uses this ID to retrieve the corresponding session data and authenticate the user.

6. **Session Expiration**: Sessions are configured with an expiration time (e.g., 30 minutes of inactivity) to enhance security. After expiration, users are required to re-authenticate.

JSON Web Tokens (JWT) for API Authentication

While session-based authentication is used for the web interface, RideEase also implements JWT for its API, particularly useful for mobile app integration and third-party service interactions. Here's how JWT authentication is implemented:

1. **Token Generation**: Upon successful authentication, the server generates a JWT containing the user's ID, role, and other necessary claims. This token is signed using a secret key known only to the server.

2. **Token Storage**: The JWT is sent back to the client, which stores it securely (e.g., in local storage for web apps or secure storage for mobile apps).

3. **Authorization Header**: For API requests, the client includes the JWT in the Authorization header of each request, typically in the format: `Authorization: Bearer <token>`.

4. **Token Verification**: The server verifies the JWT's signature and extracts the user information for each request, authorizing access to protected resources based on the user's role and permissions.

5. **Token Expiration**: JWTs are configured with an expiration time, after which they become invalid, requiring the user to obtain a new token.

Authorization

Authorization in RideEase is role-based, with different access levels for customers, drivers, and administrators:

1. **Middleware**: Custom middleware functions are implemented to check the user's role and permissions before allowing access to specific routes or resources.

2. **Role-Based Access Control (RBAC)**: Different roles (customer, driver, admin) are assigned different sets of permissions. For example:

    a. Customers can book rides, view their ride history, and manage their profiles.

b. Drivers can accept ride requests, update their availability, and view their earnings.

c. Administrators have access to user management, ride analytics, and system configuration options.

3. **Route Protection**: Express.js routes are protected using middleware that checks the user's authentication status and role before processing the request.

Security Measures

To enhance the security of the authentication system, RideEase implements several additional measures:

1. **Password Hashing**: User passwords are hashed using bcrypt before storage, ensuring that even if the database is compromised, actual passwords remain secure.

2. **HTTP**: All communications between the client and server are encrypted using HTTP to prevent man-in-the-middle attacks and data interception.

3. **CSRF Protection**: For session-based authentication, CSRF tokens are implemented to prevent cross-site request forgery attacks.

4. **Rate Limiting**: API endpoints, especially those related to authentication, are rate-limited to prevent brute-force attacks.

5. **Two-Factor Authentication (2FA)**: An optional 2FA system is implemented for users who require an extra layer of security, especially useful for administrative accounts.

6. **Secure Cookie Settings**: Session cookies are set with the Secure flag (only transmitted over HTTP) and HttpOnly flag (inaccessible to client-side scripts) to enhance security.

By combining session-based authentication for web interfaces and JWT for API access, RideEase provides a flexible, secure, and efficient authentication and authorization system. This dual approach allows for seamless integration with various client types while maintaining high security standards and user-friendly experiences.

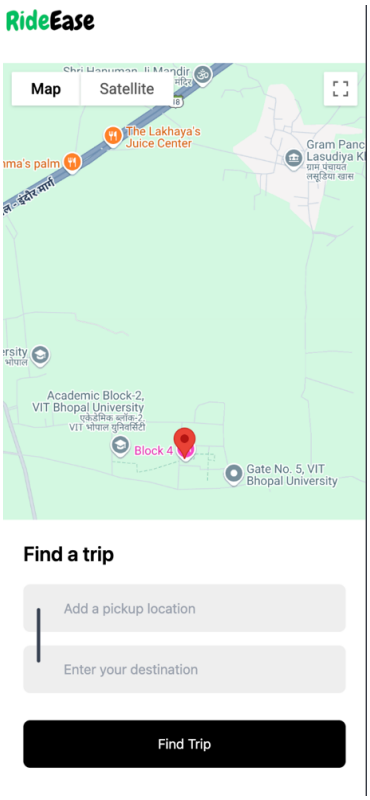
CHAPTER 9

USER INTERFACE

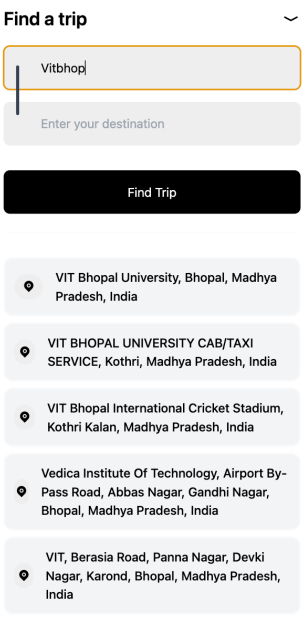
The RideEase Cab Booking App features a modern, intuitive, and user-friendly interface designed to provide a seamless experience for both passengers and drivers. This chapter showcases the key UI components and features of the application through a series of screenshots and descriptions.

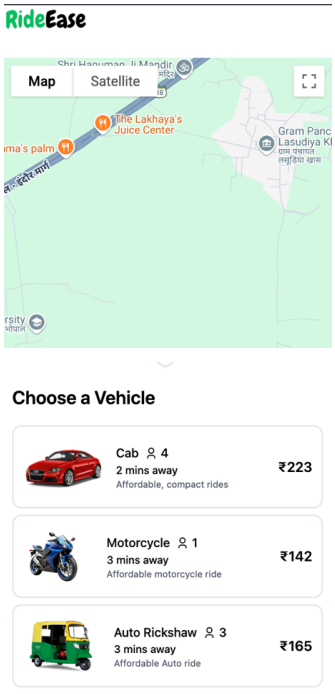
1. Start Page

2. Home Screen

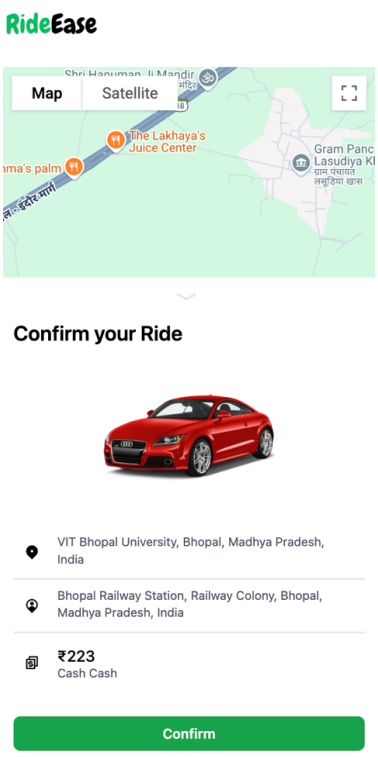


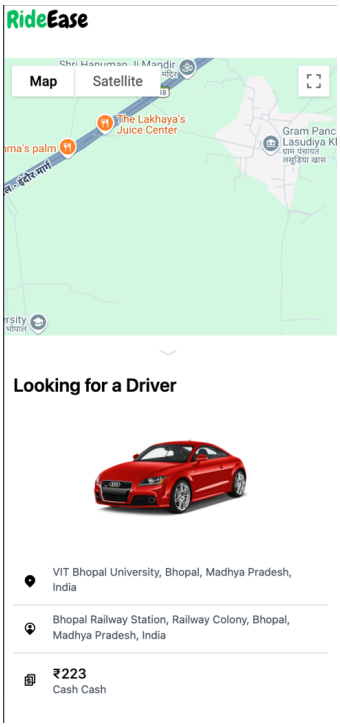
3. Ride Booking Interface



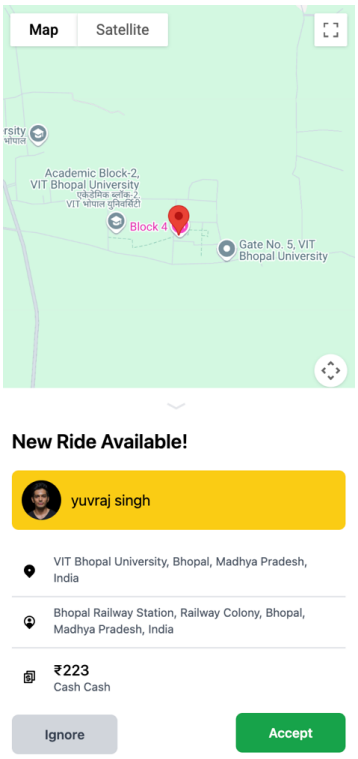


3





4. Driver Selection



CHAPTER 10

TESTING

The RideEase Cab Booking App implements a comprehensive testing strategy to ensure reliability, performance, and security across all components. This chapter details the testing methodologies, tools, and processes employed throughout the development lifecycle.

Testing Strategy Overview

The testing approach for RideEase follows a multi-layered strategy that encompasses various testing types:

1. **Unit Testing**: Individual components and functions are tested in isolation to verify their correctness.
2. **Integration Testing**: Interactions between different modules and services are tested to ensure they work together as expected.
3. **End-to-End Testing**: Complete user flows are tested from start to finish to validate the entire application behavior.
4. **Performance Testing**: The application is tested under various load conditions to ensure it can handle expected user traffic.
5. **Security Testing**: Vulnerabilities and potential security issues are identified and addressed.

This comprehensive approach ensures that all aspects of the application are thoroughly tested before deployment.

Testing Tools and Frameworks

Frontend Testing

For the React frontend, we utilize:

1. **Jest**: As the primary testing framework for unit and integration tests. Jest provides a robust environment for testing React components with features like snapshot testing, mocking, and code coverage reports.

2. **React Testing Library**: This tool complements Jest by providing utilities for testing React components in a way that resembles how users interact with the application. It encourages testing behavior rather than implementation details.

3. **Cypress**: For end-to-end testing, Cypress allows us to automate browser testing and simulate user interactions across multiple pages and states of the application.

Example of a component test using Jest and React Testing Library:

```
import { render, screen, fireEvent } from '@testing-library/react';
 import BookingForm from './BookingForm';

 test('submits booking with correct location data', () => {
  render(<BookingForm />);

  // Fill in form fields
  fireEvent.change(screen.getByLabelText(/pickup location/i), {
   target: { value: '123 Main St' }
  });
  fireEvent.change(screen.getByLabelText(/destination/i), {
   target: { value: '456 Park Ave' }
  });

  // Submit form
  fireEvent.click(screen.getByRole('button', { name: /book now/i }));

  // Assert that the form submission handler was called with correct data
  expect(mockSubmitHandler).toHaveBeenCalledWith({
   pickup: '123 Main St',
   destination: '456 Park Ave'
  });
```

```
});
```

Backend Testing

For the Node.js/Express backend, we employ:

1. **Mocha**: A flexible testing framework that provides a structured approach to organizing test suites and cases.
2. **Chai**: An assertion library that pairs with Mocha to provide expressive language for writing test assertions.
3. **Supertest**: For API testing, Supertest allows us to make HTTP requests to our API endpoints and assert responses.
4. **Sinon**: Used for creating spies, stubs, and mocks to isolate units of code during testing.

Example of an API endpoint test:

```
const request = require('supertest');
 const { expect } = require('chai');
 const app = require('../app');
 const mongoose = require('mongoose');

 describe('Booking API', () => {
  before(async () => {
    // Connect to test database
    await mongoose.connect(process.env.TEST_MONGODB_URI);
  });

  after(async () => {
    // Disconnect after tests
    await mongoose.disconnect();
  });

  it('should create a new booking when valid data is provided', async () => {
    const response = await request(app)
```

```
    .post('/api/bookings')
    .set('Authorization', `Bearer ${testUserToken}`)
    .send({
      pickupLocation: {
        latitude: 37.7749,
        longitude: -122.4194,
        address: '123 Main St'
      },
      dropoffLocation: {
        latitude: 37.3382,
        longitude: -121.8863,
        address: '456 Park Ave'
      },
      cabType: 'standard'
    });

  expect(response.status).to.equal(201);
  expect(response.body).to.have.property('success', true);
  expect(response.body).to.have.property('booking');
  expect(response.body.booking).to.have.property('status', 'pending');
 });
});
```

Database Testing

For testing database operations:

1. **MongoDB Memory Server**: Creates an in-memory MongoDB instance for testing, eliminating the need for a separate test database.

2. **Mongoose**: Used in conjunction with testing frameworks to validate schema operations and database interactions.

Continuous Integration and Testing

RideEase implements continuous integration using:

1. **GitHub Actions**: Automated workflows are configured to run tests on every pull request and commit to the main branch.
2. **Jest Coverage Reports**: Code coverage reports are generated to identify areas of the codebase that lack sufficient test coverage.

Example GitHub Actions workflow configuration:

```
name: RideEase CI

on:
 push:
  branches: [ main ]
 pull_request:
  branches: [ main ]

jobs:
 test:
  runs-on: ubuntu-latest

  steps:
  - uses: actions/checkout@v2
  - name: Use Node.js
    uses: actions/setup-node@v2
    with:
      node-version: '16'
  - name: Install dependencies
    run: |
      cd frontend && npm install
      cd ../backend && npm install
  - name: Run frontend tests
    run: cd frontend && npm test -- --coverage
  - name: Run backend tests
```

```
run: cd backend && npm test
```

Performance Testing

To ensure the application can handle expected load:

1. **JMeter**: Used to simulate heavy traffic to the API endpoints and measure response times.
2. **Lighthouse**: Employed to analyze frontend performance, accessibility, and best practices.
3. **Artillery**: Helps create realistic load scenarios to test the application's scalability.

Security Testing

Security is a critical aspect of the RideEase application, particularly given the sensitive nature of user data and payment information:

1. **OWASP ZAP**: Used to scan for common security vulnerabilities such as XSS, CSRF, and SQL injection.
2. **npm audit**: Regularly run to identify and fix security vulnerabilities in dependencies.
3. **JWT Testing**: Custom tests to verify the security of authentication mechanisms.

Test Coverage and Reporting

The testing strategy aims for high test coverage across all critical components:

1. **Frontend Coverage Target**: 80% or higher for components and utility functions.
2. **Backend Coverage Target**: 90% or higher for controllers, services, and models.
3. **Critical Paths**: 100% coverage for authentication, payment processing, and booking flows.

Test reports are generated after each test run and include:

- Number of tests passed/failed
- Code coverage percentages
- Performance metrics

- Security vulnerabilities detected

Test Environments

RideEase maintains separate environments for testing:

1. **Development**: Where developers run local tests during feature development.
2. **Testing/Staging**: A dedicated environment that mirrors production for integration and end-to-end testing.
3. **Production**: The live environment, where smoke tests are run after deployments.

This multi-environment approach ensures that tests are conducted in conditions that closely resemble the production environment, reducing the risk of environment-specific issues.

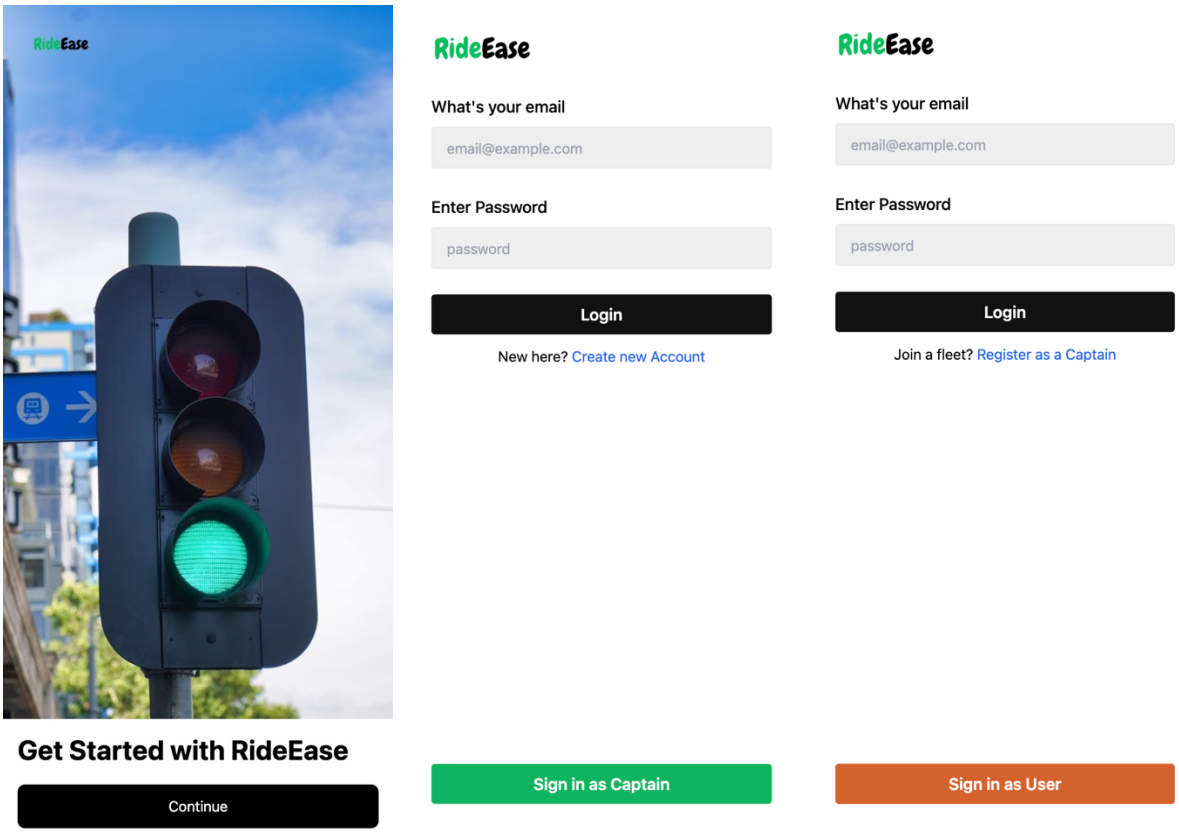
The comprehensive testing strategy implemented for RideEase ensures that the application is robust, secure, and provides a seamless experience for users. By combining various testing methodologies and tools, we can identify and address issues early in the development process, resulting in a higher-quality application.

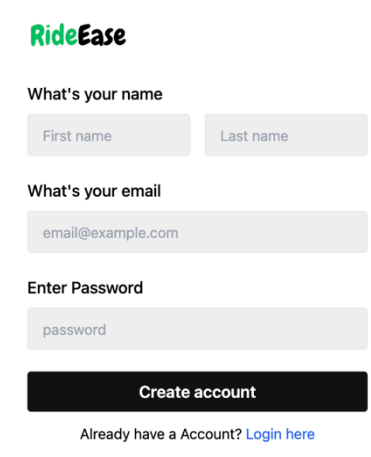
CHAPTER 11

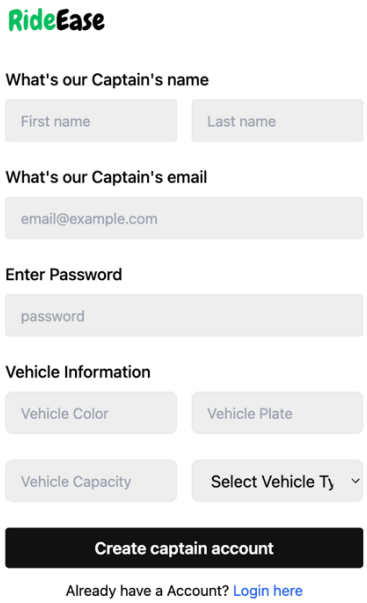
SCREENSHOTS AND DEMO

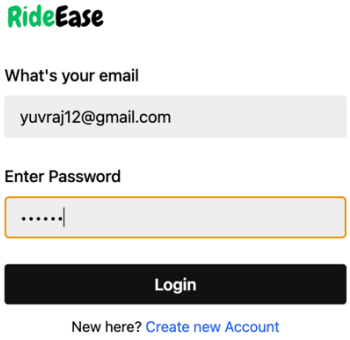
1.1 Project Title: RideEase Cab Booking App

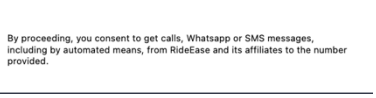
The RideEase Cab Booking App is a comprehensive and modern solution designed to simplify the process of booking cabs. Built using the MERN stack (MongoDB, Express.js, React, and Node.js), this application provides users with an intuitive interface to book rides, track their trips in real-time, and make secure payments. The app is tailored to meet the needs of both passengers and drivers, ensuring a seamless and efficient transportation experience.

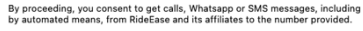



RideEase
What's your name
First name
Last name
What's your email
email@example.com
Enter Password
password
Create account
Already have a Account? Login here

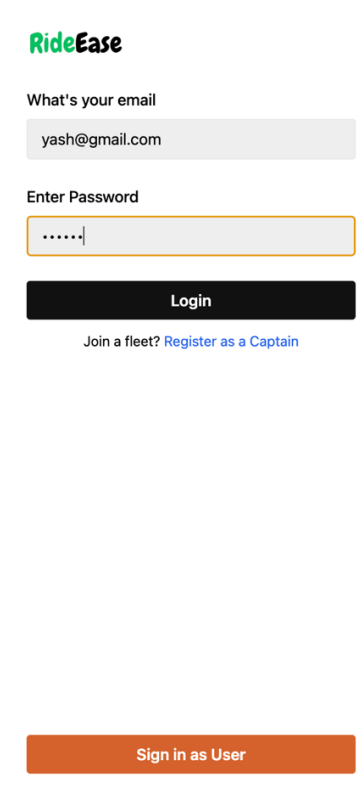

RideEase
What's our Captain's name
First name
Last name
What's our Captain's email
email@example.com
Enter Password
password
Vehicle Information
Vehicle Color
Vehicle Plate
Vehicle Capacity
Select Vehicle Ty
Create captain account
Already have a Account? Login here

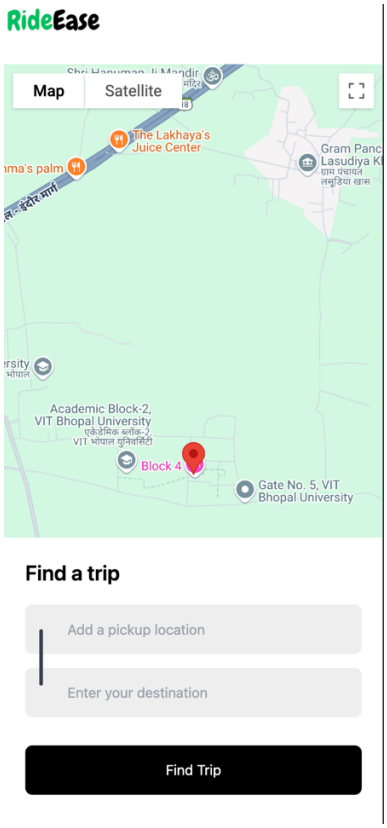

RideEase
What's your email
yuvraj12@gmail.com
Enter Password
Login
New here? Create new Account

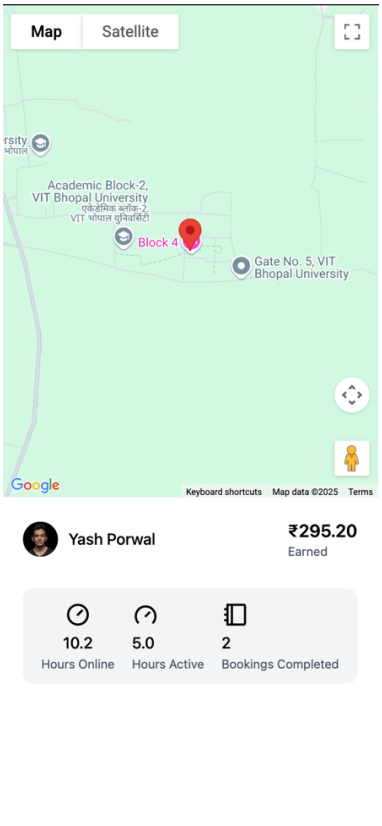

By proceeding, you consent to get calls, Whatsapp or SMS messages, including by automated means, from RideEase and its affiliates to the number provided.

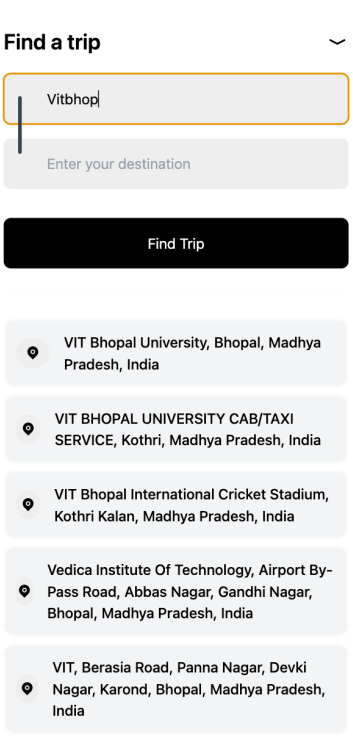

By proceeding, you consent to get calls, Whatsapp or SMS messages, including by automated means, from RideEase and its affiliates to the number provided.

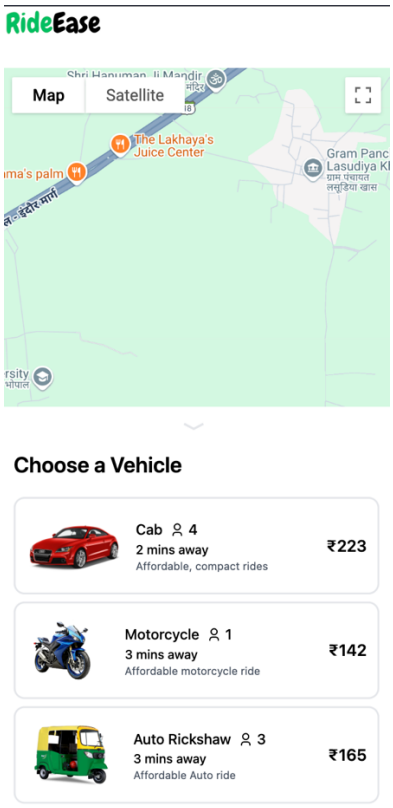

Sign in as Captain

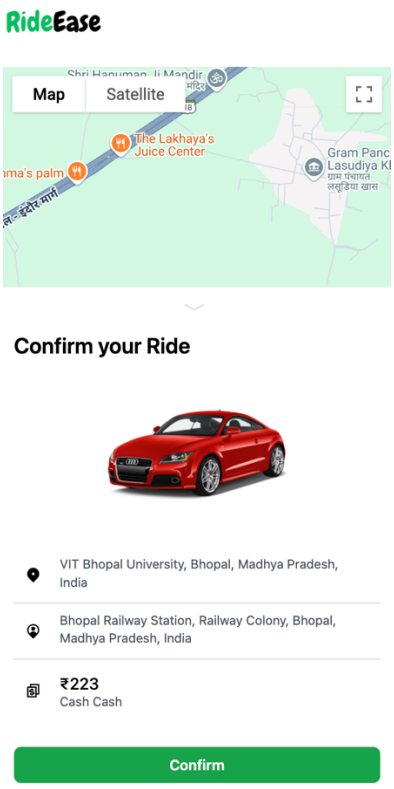

RideEase
What's your email
yash@gmail.com
Enter Password
Login
Join a fleet? Register as a Captain
Sign in as User

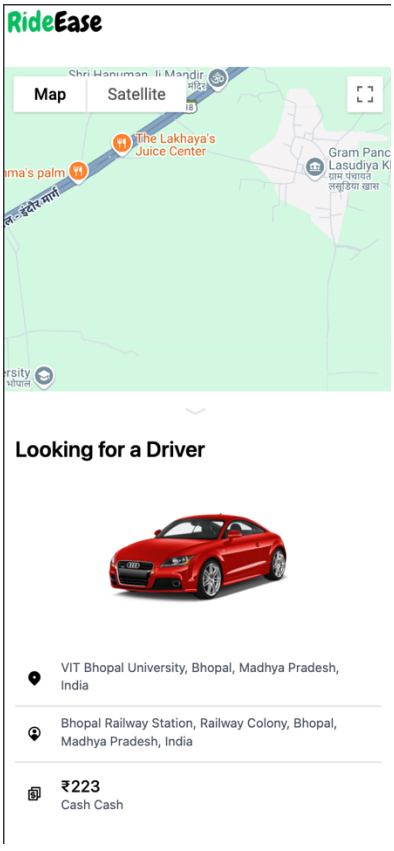

RideEase
Map
Satellite
The Lakhaya's Juice Center
Academic Block-2, VIT Bhopal University
Block 4
Gate No. 5, VIT Bhopal University
Find a trip
Add a pickup location
Enter your destination
Find Trip

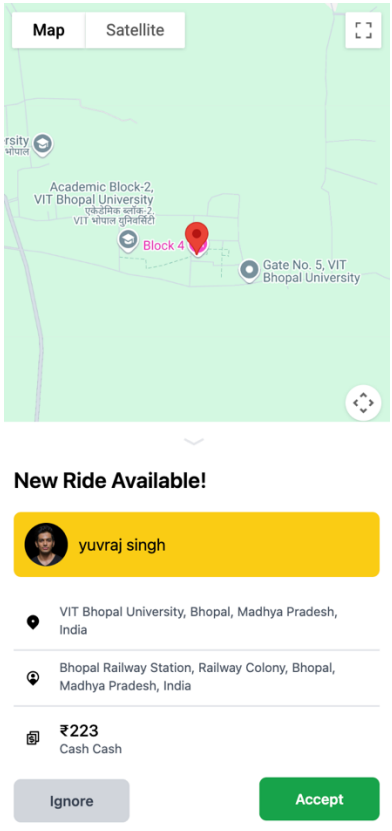

Map
Satellite
Academic Block-2, VIT Bhopal University
Block 4
Gate No. 5, VIT Bhopal University
Google
Yash Porwal
₹295.20
Earned
10.2
Hours Online
5.0
Hours Active
2
Bookings Completed

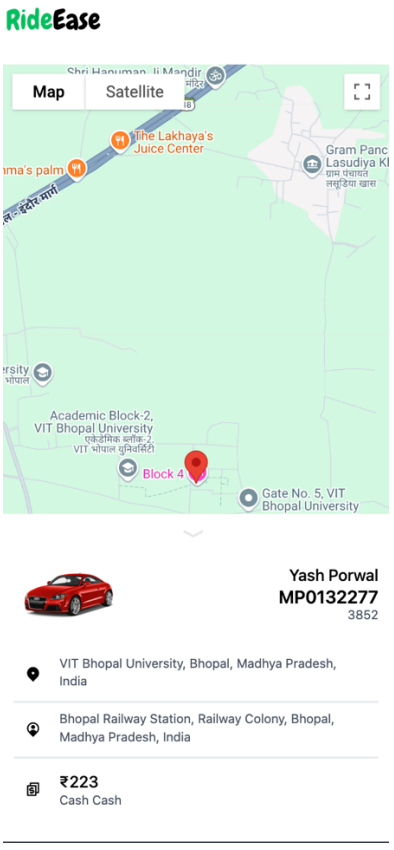

Find a trip
Vitbhop
Enter your destination
Find Trip
VIT Bhopal University, Bhopal, Madhya Pradesh, India
VIT BHOPAL UNIVERSITY CAB/TAXI SERVICE, Kothri, Madhya Pradesh, India
VIT Bhopal International Cricket Stadium, Kothri Kalan, Madhya Pradesh, India
Vedica Institute Of Technology, Airport By-Pass Road, Abbas Nagar, Gandhi Nagar, Bhopal, Madhya Pradesh, India
VIT, Berasia Road, Panna Nagar, Devki Nagar, Karond, Bhopal, Madhya Pradesh, India

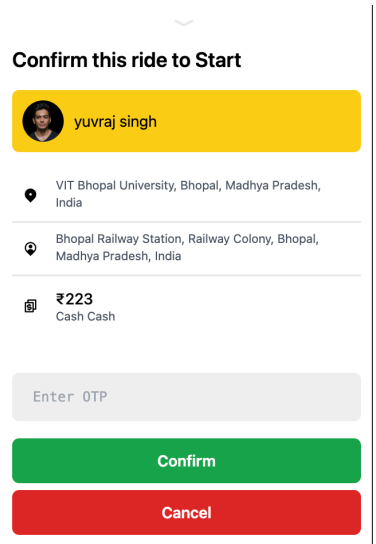

RideEase
Map
Satellite
The Lakhaya's Juice Center
Choose a Vehicle
Cab 4
2 mins away
Affordable, compact rides
₹223
Motorcycle 1
3 mins away
Affordable motorcycle ride
₹142
Auto Rickshaw 3
3 mins away
Affordable Auto ride
₹165

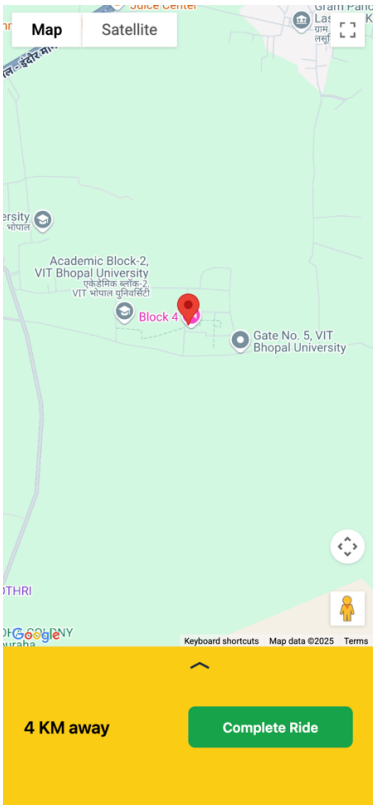

RideEase
Map
Satellite
The Lakhaya's Juice Center
Confirm your Ride
VIT Bhopal University, Bhopal, Madhya Pradesh, India
Bhopal Railway Station, Railway Colony, Bhopal, Madhya Pradesh, India
₹223
Cash Cash
Confirm

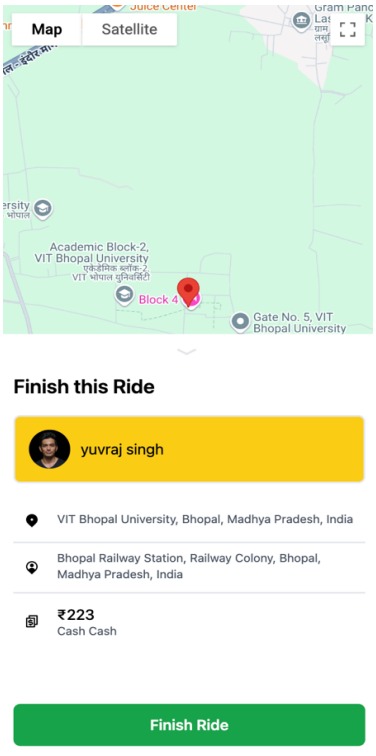

RideEase
Map
Satellite
The Lakhaya's Juice Center
Looking for a Driver
VIT Bhopal University, Bhopal, Madhya Pradesh, India
Bhopal Railway Station, Railway Colony, Bhopal, Madhya Pradesh, India
₹223
Cash Cash


Map
Satellite
Academic Block-2, VIT Bhopal University
Block 4
Gate No. 5, VIT Bhopal University
New Ride Available!
yuvraj singh
VIT Bhopal University, Bhopal, Madhya Pradesh, India
Bhopal Railway Station, Railway Colony, Bhopal, Madhya Pradesh, India
₹223
Cash Cash
Ignore
Accept


RideEase
Map
Satellite
The Lakhaya's Juice Center
Academic Block-2, VIT Bhopal University
Block 4
Gate No. 5, VIT Bhopal University
Yash Porwal
MP0132277
3852
VIT Bhopal University, Bhopal, Madhya Pradesh, India
Bhopal Railway Station, Railway Colony, Bhopal, Madhya Pradesh, India
₹223
Cash Cash







Demo Link:

https://drive.google.com/file/d/1vzQ9c5HjFkoGSSsGuXt5L2T1COzaLrSp/view?usp=sharing

CHAPTER 12

KNOWN ISSUES

While the RideEase Cab Booking App has undergone extensive testing and development, there are a few known issues that users and developers should be aware of:

1. **GPS Accuracy in Dense Urban Areas**:
   In areas with tall buildings or underground locations, GPS accuracy may be reduced, affecting precise pickup and drop-off locations. Users might need to manually adjust their location in such scenarios.

2. **Peak Hour Booking Delays**:
   During extremely high demand periods (e.g., New Year's Eve), there might be slight delays in driver assignment and increased waiting times. We're continuously optimizing our algorithms to mitigate this.

3. **Payment Gateway Timeouts**:
   Occasionally, during high traffic periods, payment gateway responses may time out. The system is designed to retry, but in rare cases, users might need to reattempt payment.

4. **Cross-Platform UI Inconsistencies**:
   Some UI elements may appear slightly different across various devices and operating systems. We're working on achieving better consistency across platforms.

5. **Real-time Tracking Lag**:
   In areas with poor network connectivity, real-time tracking of drivers may experience a slight lag. The system uses interpolation to estimate positions, but actual locations may vary.

6. **Language Support Limitations**:
   While the app supports multiple languages, some less common languages may have incomplete translations. We're actively working with translators to improve coverage.

7. **Push Notification Delays on Certain Devices**:

   Some Android devices with aggressive battery optimization may delay push notifications. We recommend users add RideEase to their battery optimization exception list.

8. **Rare Data Synchronization Issues**:

   In cases of sudden network drops, there might be temporary discrepancies between the app's displayed data and the server data. The app is designed to re-sync when connection is restored.

CHAPTER 13

FUTURE ENHANCEMENTS

RideEase is committed to continuous improvement and innovation. Here are some potential future features and enhancements we're considering:

1. **AI-Powered Route Optimization**:

   Implement machine learning algorithms to predict traffic patterns and suggest optimal routes, reducing travel time and costs.

2. **Augmented Reality (AR) for Pickup**:

   Develop an AR feature to help users and drivers locate each other more easily in crowded areas.

3. **Integration with Public Transit**:

   Offer multimodal transportation options by integrating public transit information, allowing users to combine cab rides with other forms of public transportation.

4. **Carpooling and Ride-Sharing Options**:

   Introduce a carpooling feature to match users with similar routes, reducing costs and environmental impact.

5. **Blockchain-Based Loyalty Program**

   Implement a blockchain-based loyalty system for more transparent and secure reward points management.

6. **Voice-Activated Booking**:

   Integrate with virtual assistants like Siri, Google Assistant, and Alexa for hands-free ride booking.

7. **Predictive Booking**:

   Use historical data and AI to predict user's ride needs and offer proactive booking suggestions.

8. **Enhanced Accessibility Features**:

   Improve app usability for users with disabilities, including voice navigation and screen reader optimizations.

9. **Carbon Footprint Tracking**:

   Allow users to track and offset their carbon footprint from rides, partnering with environmental organizations.

10. **In-App Chat Translation**:

    Implement real-time translation in the in-app chat to facilitate communication between users and drivers who speak different languages.

11. **Dynamic Pricing Based on Green Energy Usage**:

    Introduce variable pricing that offers discounts for rides in electric or hybrid vehicles to promote eco-friendly options.

12. **Virtual Queue System**:

    Develop a system for high-demand areas or events where users can join a virtual queue for rides, providing estimated wait times.

13. **Enhanced Driver Safety Features**:

    Implement advanced driver monitoring systems to detect fatigue or distracted driving, enhancing overall ride safety.

14. **Integration with Smart City Infrastructure**:

    Collaborate with smart city initiatives to integrate real-time traffic light data and parking availability information.

15. **Personalized Travel Insights**:

    Provide users with personalized analytics on their travel habits, costs, and environmental impact, with suggestions for optimization.

These future enhancements aim to not only improve the core functionality of RideEase but also to innovate in areas of sustainability, user experience, and integration with emerging technologies. As always, we will prioritize these enhancements based on user feedback, technological feasibility, and

alignment with our company's vision for urban mobility.