
Implementation Document

for

Fleet

Version 1.0

Prepared by

Group: 11

Habeeb Ramith Kumar	230432
Reddi Pallavi	230850
Pasala Bosu Akil Teja	230742
Shashi Bhidodiya	230742
Manam Amara Gayathri	230624
Sai Prabhav	230060
Koneti Karthik	230568
Poorvie Sadagopan	230759
Pittala Sruthi	230751
Jyothika Seru	230946

Group Name: Null Pointers

hramith23@iitk.ac.in
rpallavi23@iitk.ac.in
pbosuateja23@iitk.ac.in
shashib23@iitk.ac.in
mamara23@iitk.ac.in
addulasr23@iitk.ac.in
konetik23@iitk.ac.in
poorvies23@iitk.ac.in
sruthip23@iitk.ac.in
serujy23@iitk.ac.in

Course: CS253

Mentor TA: Souvik Mukherjee

Date: 28/3/2025



CONTENTS	II
REVISIONS	II
1 IMPLEMENTATION DETAILS.....	1
2 CODEBASE.....	5
3 COMPLETENESS.....	10
APPENDIX A - GROUP LOG	12

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
V1.0	All group members	First version of the implementation document	28/03/2025

Implementation Details

Fleet is a comprehensive web-based vehicle rental software built using the MERN stack, which leverages MongoDB, Express.js, React.js, and Node.js. This technology stack was chosen for its ability to seamlessly handle the complex requirements of our rental system.

We have used **Model-View-Controller** architecture for our software. This architecture is well-suited for the software since there are multiple ways to interact with the same data presented in different views. Also, the data has to be changed independent of the view.

The code structure reflects the architectural pattern used. The three components constituting the software are:

1. Model: Consists of MongoDB schemas for objects of various classes. The classes have been described in more detail in the Design document. The code for this component ('models' folder) is located in the "Fleet/Backend/" directory. Each file within the folder contains the schema for a particular class.

2. View: Controls what the user sees and how he/she can interact with the software. The code for this component rests in the 'src' folder within the "Fleet/Frontend/" directory. Each JavaScript file within the folder contains code that controls the view presented to the user. A corresponding CSS file, which controls the design of the webpage, has been included in each JavaScript file. In addition, we have the components folder (which has files containing react components that handle user interaction) and the assets folder (which has other necessary files (like images)).

3. Controller: Consists of business logic that makes the software perform the desired tasks. The code for this component ('controllers' folder) is located in the "Fleet/Backend/" directory. Each file within the folder contains react components dealing with a specific functionality.

Frontend :

Languages Used: HTML, CSS, JavaScript

Framework/Library Used: React.js and Express

Build Tool: Vite

Justification-

1. HTML

Provides the structural foundation of web pages, enabling developers to define elements such as headings, paragraphs, images, and links. It is essential for accessibility and search engine optimization.

2. CSS

Enhances the visual appeal of web pages by controlling colors, fonts, layouts, and animations. It ensures consistent design across pages and supports responsive design for compatibility with various devices.

3. JavaScript

JavaScript adds interactivity to websites, enabling dynamic features like form validation, animations, and real-time updates. It is universally supported by browsers and integrates well with third-party libraries for enhanced functionality.

4. React.js

- React is chosen for its **modular structure**, which makes code maintenance easier and more flexible.
- The **virtual DOM** feature optimizes rendering performance, making applications faster and more responsive.
- React's **reusable components** save development time by allowing developers to use the same code for similar features across the application.
- It supports SEO-friendly fast rendering, which is crucial for modern web applications.

5. Vite

- Vite is a modern build tool that offers **lightning-fast development server start times** due to its native ES module support.
- It uses **Hot Module Replacement (HMR)** to provide instant updates during development without requiring a full page reload.
- Vite is optimized for modern JavaScript frameworks like React and ensures faster builds compared to traditional tools like Webpack.

- Its simplicity and flexibility make it ideal for projects requiring quick iteration cycles.

Backend

Runtime Environment: Node.js

Framework : Express

Database Management System : MongoDB

Justification

1. Node.js :

- Node.js uses a single-threaded event loop with non-blocking I/O calls, making it highly efficient in handling thousands of simultaneous connections without consuming excessive system resources.
- Node.js comes with a built-in package manager called npm, which is the largest ecosystem of open-source libraries in the world.
- Npm allows developers to easily install, manage, and share reusable code packages, making it quick and convenient to add functionality to Node.js applications.
- It provides numerous libraries and reusable templates.
- It allows developers to use JavaScript on both frontend and backend, improving team efficiency and reducing development costs.
- Node.js is lightweight, scalable, and ideal for applications requiring high concurrency.

2. Express :

- It simplifies the process of building web applications and APIs in Node.js by providing a minimalistic and flexible set of features.
- It simplifies the process of building web applications and APIs in Node.js by providing a minimalistic and flexible set of features.
- It provides a simple and intuitive way to define routes for handling different HTTP requests (GET, POST, PUT, DELETE, etc.) on various URLs.
- Express simplifies error handling by providing middleware that can catch and process errors, making it easier to manage and debug applications.

3. MongoDB:

- MongoDB's **document-oriented model** offers flexibility in schema design, allowing unstructured or semi-structured data to be stored easily without predefined schemas.

- Its **JSON-like BSON format** simplifies data manipulation and integrates seamlessly with JavaScript-based applications like Node.js.
- MongoDB supports horizontal scalability through sharding, making it suitable for handling large volumes of data or traffic spikes in modern web applications.
- The database provides powerful querying capabilities, including geospatial queries and aggregation pipelines, enabling complex data operations efficiently.

Summary of Benefits

The chosen MERN Stack is ideal for the project due to its scalability, performance efficiency, flexibility in handling evolving requirements, and seamless integration between frontend and backend components. This combination empowers us to build fast, interactive, and data-driven applications effectively.

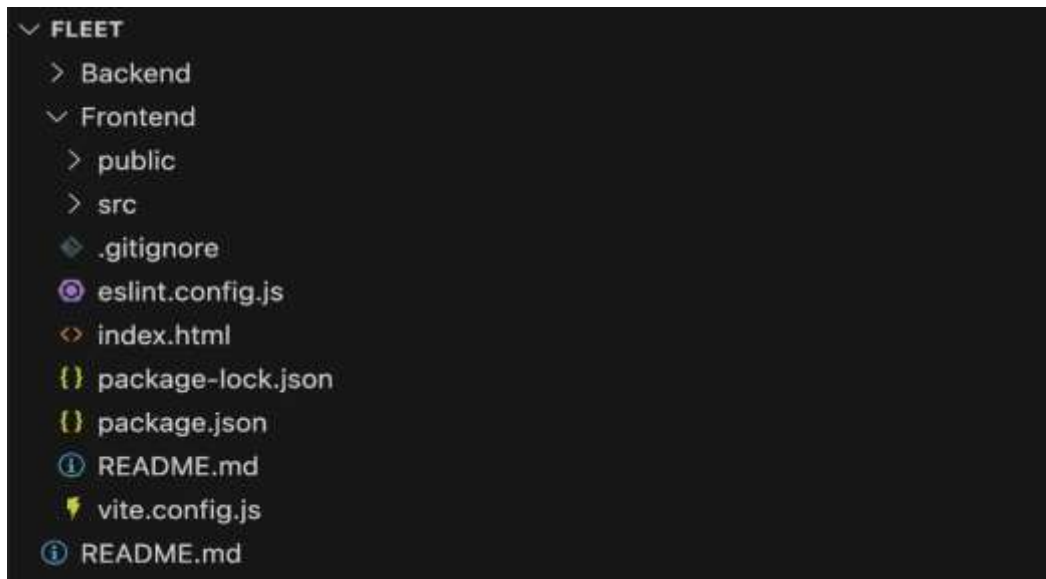
Codebase

GitHub repository - [Fleet](#)

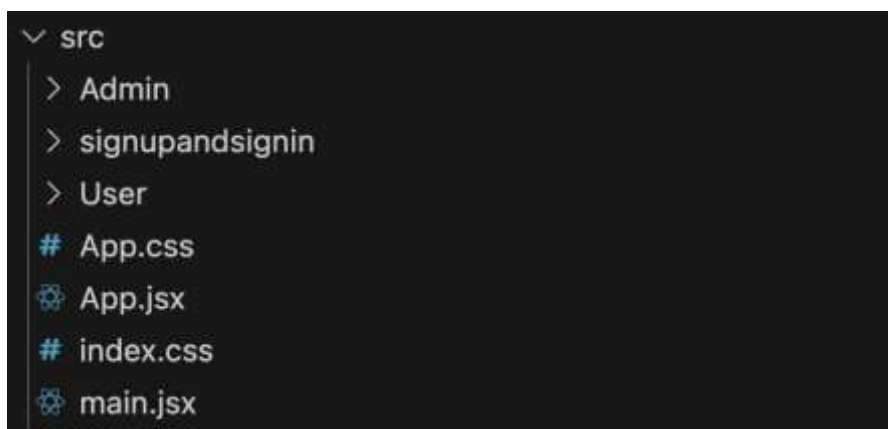
Codebase Navigation

Frontend

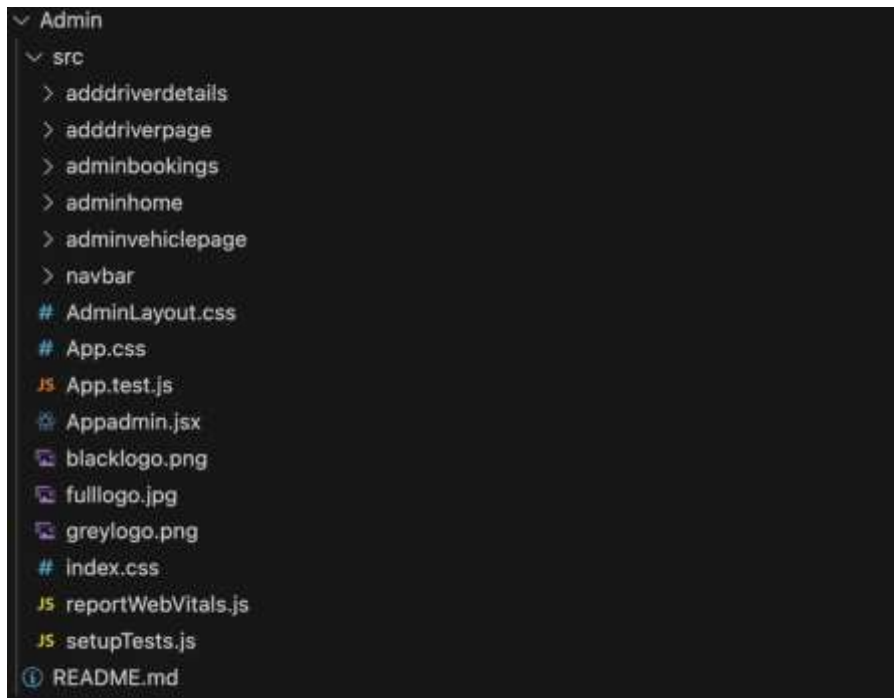
The frontend code of the webpage is present in “Fleet/Frontend/”. The src folder contains all the React source code files. The public folder contains the visual content that the user sees on the webpage.



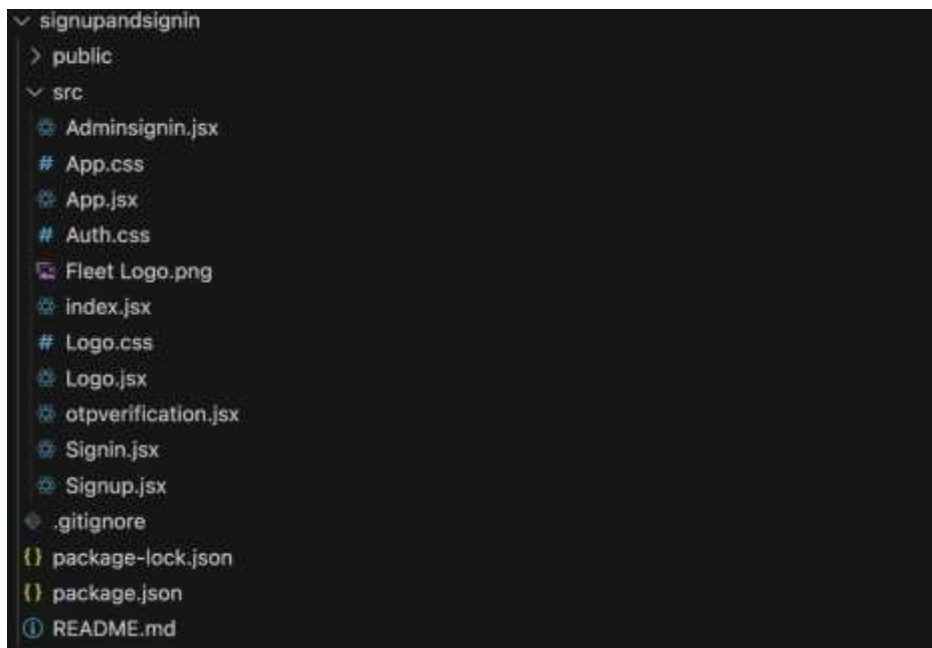
The src folder further contains three folders namely Admin, signupandsignin and User.



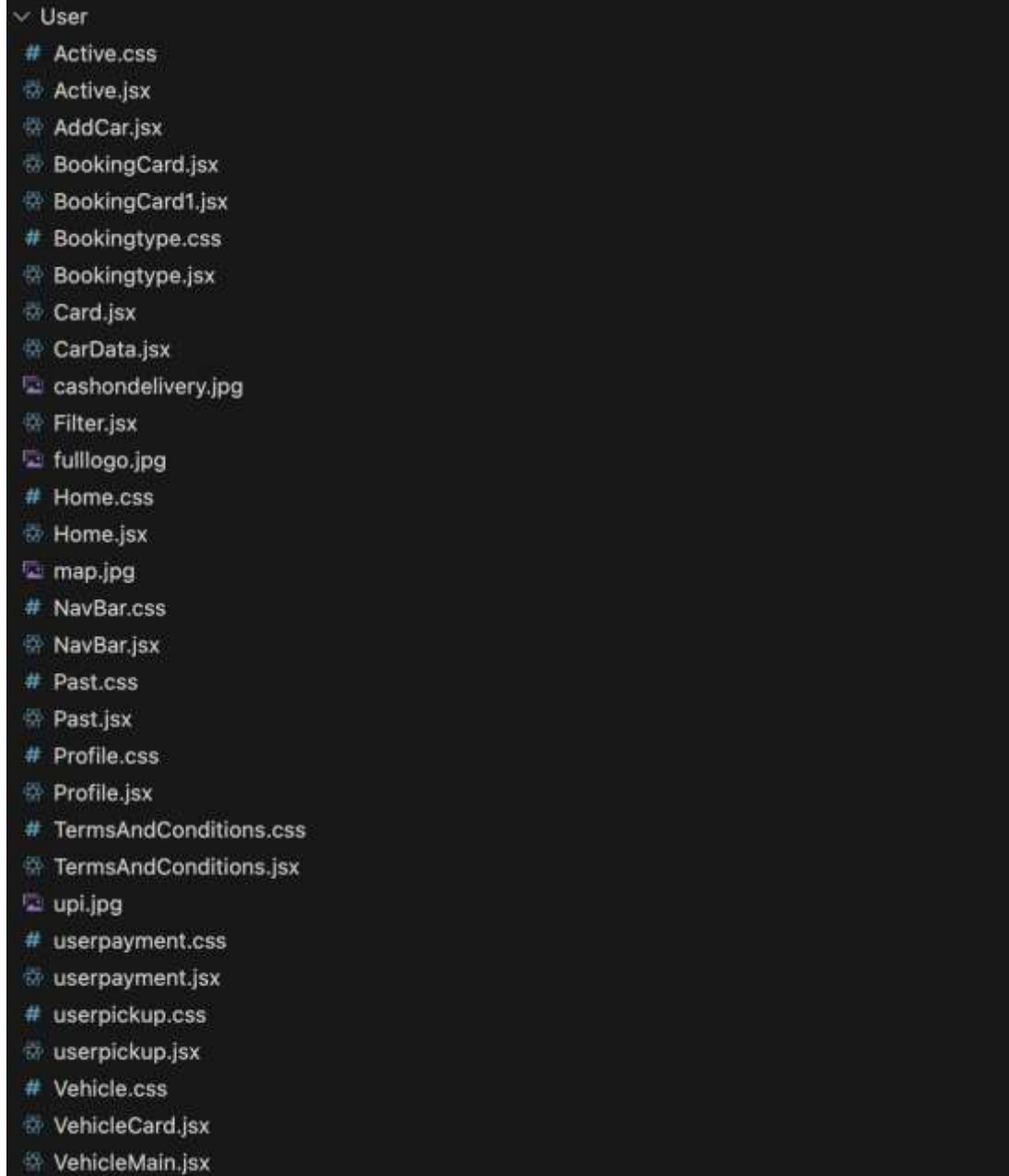
Admin folder contains the separate codes in its src for the admin's interface. For instance, it covers files for the interfaces where admin can see/add vehicles, see the booking in the calendar or can see/add drivers.



Then there is a signupandsignin folder which has files which will handle of the sign-in interface for admin and user along with the sign-up webpage for new users.



The User folder has the files used to handle the user interface. Basically, it covers all the webpages which a user will come across once logged into the website.



```

User
├── # Active.css
├── Active.jsx
├── AddCar.jsx
├── BookingCard.jsx
├── BookingCard1.jsx
├── # Bookingtype.css
├── Bookingtype.jsx
├── Card.jsx
├── CarData.jsx
├── cashondelivery.jpg
├── Filter.jsx
├── fulllogo.jpg
├── # Home.css
├── Home.jsx
├── map.jpg
├── # NavBar.css
├── NavBar.jsx
├── # Past.css
├── Past.jsx
├── # Profile.css
├── Profile.jsx
├── # TermsAndConditions.css
├── TermsAndConditions.jsx
├── upi.jpg
├── # userpayment.css
├── userpayment.jsx
├── # userpickup.css
├── userpickup.jsx
├── # Vehicle.css
├── VehicleCard.jsx
└── VehicleMain.jsx

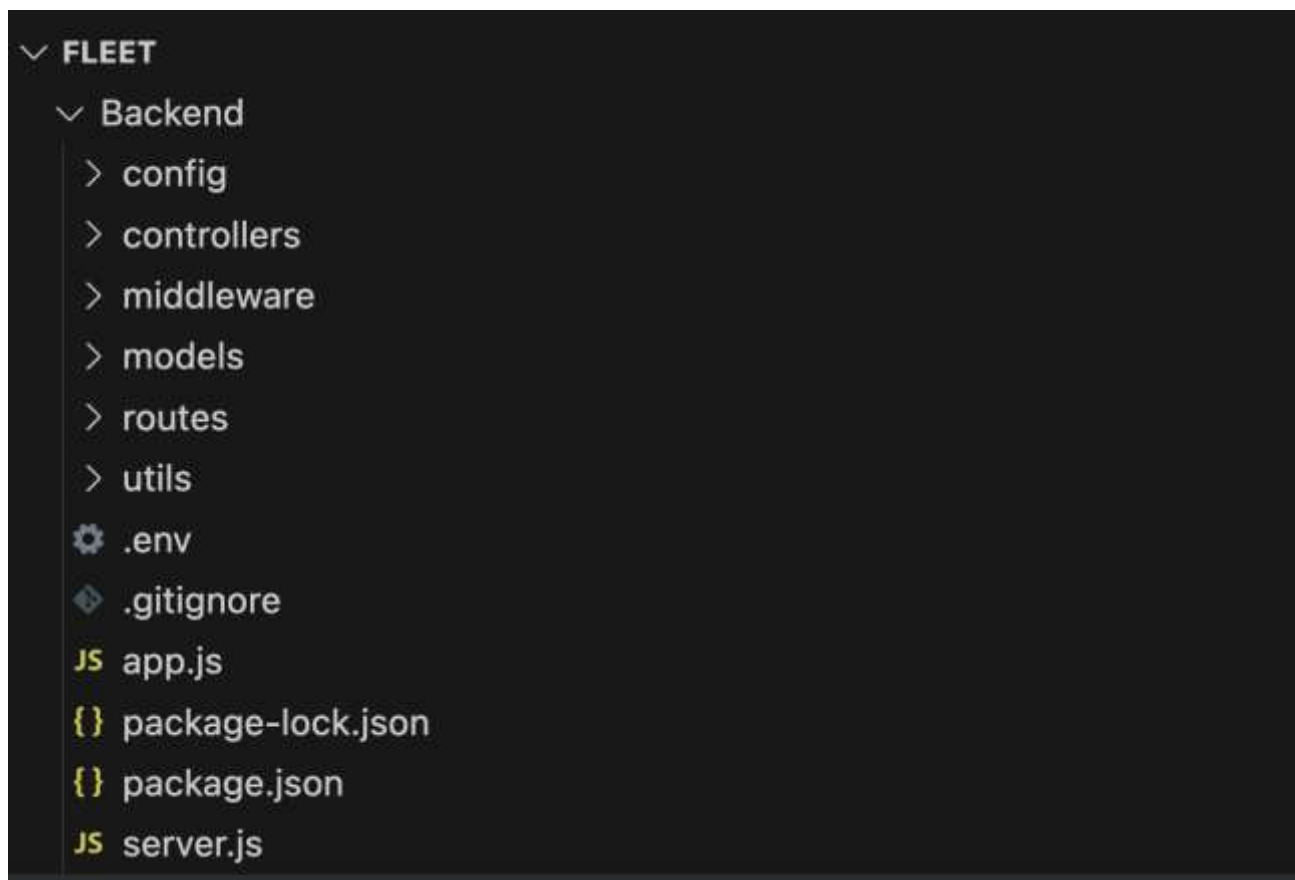
```

Backend

The backend code of the webpage is present in “Fleet/Backend/”.

The backend folder contains some very significant folders like controllers, middleware, model, routes and the server.js file.

The server.js file serves as the entry point, initializing the Express app, configuring middleware, connecting to the database, and starting the server. **Routes** define API endpoints and forward requests to **controllers**, which handle the business logic by processing requests and interacting with **models**. **Models** define the database schema and provide methods for fetching and storing data. **Middleware** functions process requests before they reach controllers, handling tasks like authentication, validation, and logging. This modular structure ensures a scalable, maintainable, and efficient backend.



'controllers' and 'middleware' folders:

```

  ✓ controllers
    JS adminauth.js
    JS adminController.js
    JS authController.js
    JS bookingController.js
    JS driverController.js
    JS paymentController.js
    JS ratingController.js
    JS userController.js
    JS vehicleController.js
  ✓ middleware
    JS authMiddleware.js
    JS errorHandler.js
    JS validationMiddleware.js
```

'models' and 'routes' folders:

```

  ✓ models
    JS Administrator.js
    JS Booking.js
    JS Driver.js
    JS Feedback.js
    JS OTP.js
    JS User.js
    JS Vehicle.js
  ✓ routes
    JS adminRoutes.js
    JS authRoutes.js
    JS bookingRoutes.js
    JS driverRoutes.js
    JS paymentRoutes.js
    JS ratingRoutes.js
    JS userRoutes.js
    JS vehicleRoutes.js
```

Completeness

We were able to successfully implement nearly all of the features that we have mentioned in the SRS. We have listed them further in the sections below:

User sign in and sign up:

- We have successfully implemented the sign in option for the user using their email address and password.
- If a user is new, he/she can use the sign-up option to register as a new user on the website.
- We have implemented a separate login webpage for the admin using email address and password.

Providing a user-friendly interface:

- We have designed the website to be easy to use and navigate.
- The system allows user to search for vehicles using filters like vehicle type, availability, price and rating.

Vehicle listing:

- We have successfully implemented a system which displays detailed vehicle description including photos, driver details, availability, price, rating etc.
- The system allows the admin to update the vehicle details, vehicle list, availability and driver details in real time.
- The system allows the user to book the vehicle with or without driver.

Portal for admin/owner:

- The website has a separate interface for admin using which admin can manage the details of the vehicles.
- The system allows the admin to assign a driver to a vehicle.
- The system also provides the admin with a 'My Booking' section to view vehicle booking history and ongoing rentals.

Feedback/rating system:

- The system allows users to give ratings to the vehicle.
- The ratings of the vehicle get displayed on the vehicle search interface, which can help future users choose the vehicle for them accordingly.

Future improvements:

- A dedicated UPI payment gateway can be integrated into the website for vehicle bookings, enabling seamless, secure, and instant transactions for users in future versions.
- The feedback process can be enhanced by incorporating a dedicated comment section alongside the existing rating system. This will allow users to provide detailed reviews on the services offered by the vehicle and the platform, enabling more comprehensive feedback and improved service optimization.
- Future iterations of the user interface can be enhanced with a more visually engaging design and an intuitive, user-friendly experience, leveraging modern UI/UX principles for improved accessibility and responsiveness.
- In the future, we can enhance the user's driving license verification process by integrating Digilocker for secure and automated document authentication, ensuring real-time validation and compliance with regulatory standards.
- A dedicated driver interface can be developed, allowing drivers to update their availability and track their driving history. This data can be utilized for salary calculations, ensuring accurate payment processing and efficient workforce management.

Appendix A - Group Log

S.no	Date	Topic of discussion	Duration
1	15/2/2025	We discussed the process of developing our website along with the possible programming languages and frameworks to use.	1hr
2	1/3/2025	We finalized our programming language as JavaScript and decided to use React and Node.js.	1hr
3	2/3/2025	We created the rough initial UI of our website to get an idea of what we are making and what features to add.	1hr
4	3/3/2025	We made the teams for front-end and back-end of the project.	1hr
5	4/3/2025	We created the Git repository of our project and briefed each other with some Git commands.	1hr
6	5/3/2025	We distributed the frontend and backend work within the group and decided who will take responsibility for which part.	1hr
7	15/3/2025	Had a meeting to discuss and clear some doubts regarding the implementations.	1hr
8	20/3/2025	Again, had a meeting to check the progress of the project and to discuss if anyone needed any assistance in their parts.	1hr
9	23/3/2025	Decided to start making the implementation document side by side as the implementation proceeded. Discussed the template of the document.	1hr
10	26/3/2025	Meet to discuss some final small changes in the website.	1hr
11	28/3/2025	Final meet before implementation document submission.	1hr