**Full Stack Development with MERN**

**Project Documentation format**

**1. Introduction**

* **Project Title:** Cab Booking App
* **Team Members:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sno.** | **Name** | **Registered Email** | **Work** |
| 1 | Nandam Saketh Ram | sakethram.22bce9935@vitapstudent.ac.in | Project + Reports |
| 2 | Harshavardhan | harshavardhan.cr2021@vitstudent.ac.in | Phase 1 + Phase 2 |
| 3 | Nishanth | nishanth.p2021@vitstudent.ac.in | - |
| 4 | Vishnu | vishnu.m2022@vitstudent.ac.in | - |

**2. Project Overview**

**Purpose:**

The purpose of this project is to develop a Cab booking system that facilitates ride booking for users and efficient management of rides and cabs for admins. The goal is to create a user-friendly platform where users can easily register, log in, and book rides using a responsive web interface. The admin panel allows administrators to manage users, cabs, and bookings effectively, ensuring the smooth operation of the service.

**Features:**

**1.** **User Registration and Login:**

Users can create an account by providing necessary details.

Registered users can log in to access the platform.

**2.** **Admin Login:**

Admins can log in to the admin panel using their credentials.

**3.** **Landing Page Navigation:**

A user-friendly landing page with navigation to different sections such as Home, About, Ride, and Donate.

**4.** **Ride Booking Process:**

Users can search for rides, view available options, and book rides by entering pickup and drop-off details.

**5.** **My Bookings Page:**

Users can view, edit, and delete their bookings on the MyBookings page.

A table displaying booking details with edit and delete options.

**6.**  **All Bookings Page (Admin):**

Admins can view, edit, and delete all bookings in the system.

A table displaying all bookings with edit and delete options for administrators.

**7.** **Admin Dashboard:**

Dashboards displaying relevant details and statistics for users and admins, such as the number of users, cabs, and bookings.

**8.** **Logout Functionality:**

Secure logout functionality for both users and admins, redirecting them to the login page.

**9.** **Responsive Design and Layout:**

The application is designed to be responsive, ensuring a good user experience on various devices and screen sizes.

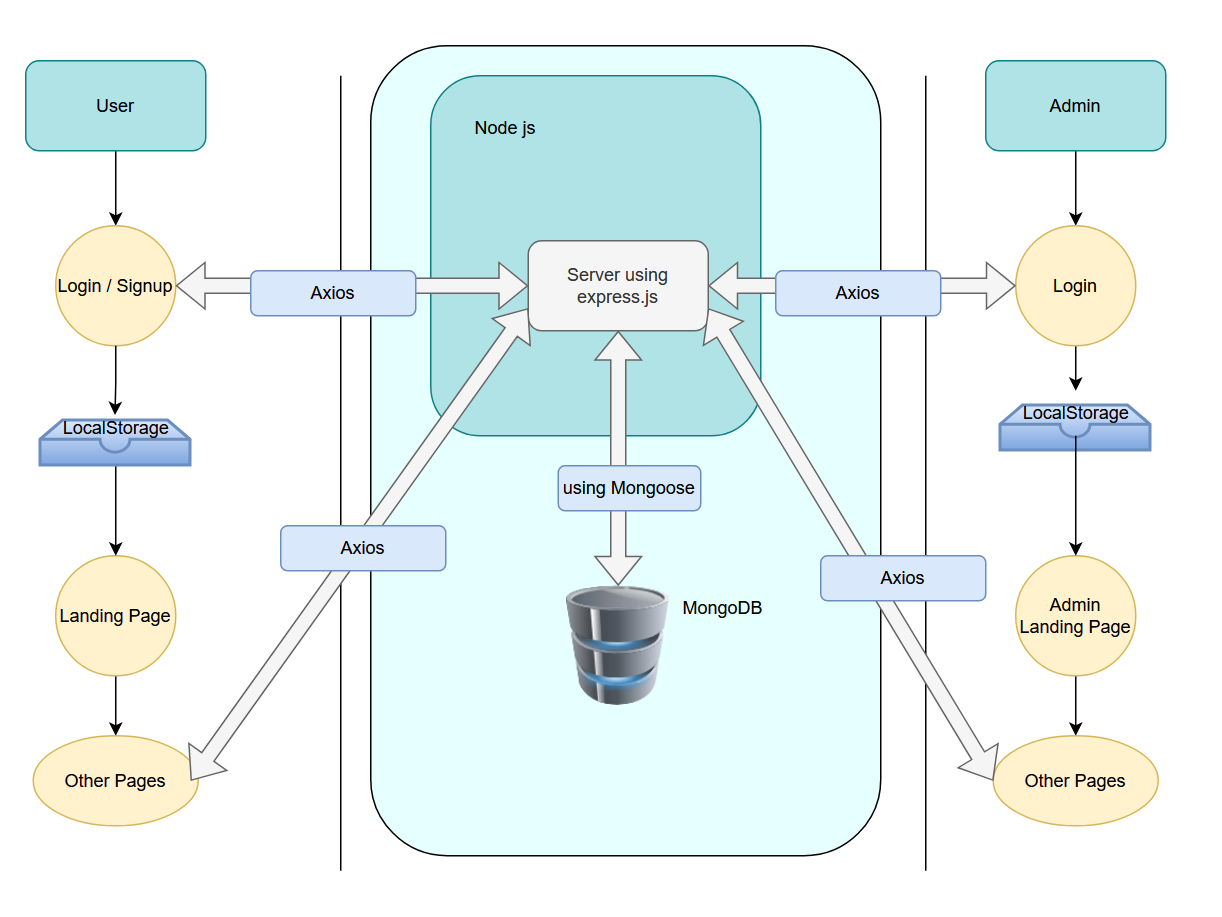
**10.** **Backend API Integration:**

Accurate retrieval and update of data through backend API integration, ensuring smooth communication between the frontend and backend.

**11.** **Admin Functionalities:**

Admins can manage users, cabs, and bookings efficiently through the admin panel. Other Features include adding, editing, and deleting cabs, as well as viewing user and booking details.

**3. Architecture**



Using React.js

Using React.js

* **Frontend:** Describe the frontend architecture using React.

**Technologies Used**:

* + **Frontend Framework**: React.js
  + **UI Framework/Libraries**: Bootstrap, Bootstrap Icons
  + **API Libraries**: Axios

**Main Folders**:

* + src/components: Contains reusable UI components.
  + src/pages: Includes different pages, categorized into User and Admin sub-folders.

**Key Components**

**1. App.js**: Handles routing and the main application layout.

* + **Routes**: Includes normal routes (accessible by anyone) and protected routes (accessible only by authenticated users).

**2. Components Folder**: Houses reusable components used across the application.

**3. Pages Folder**: Contains pages for both users and admins, including a prompt for choosing user/admin mode.

**Routing**

Managed using React Router DOM for navigation between different pages.

**User Interface (UI) Design**

* + 1. Clean and modern aesthetic focusing on usability and responsiveness.
    2. Features a navbar, user-friendly form layout, and visually appealing components like cards for displaying rides and cabs.
    3. Implemented using React and Bootstrap.
* **Backend:** Outline the backend architecture using Node.js and Express.js.

**1. Web Server**

**Technologies Used**:

* + **Backend Framework**: Node.js with Express.js

**Project Structure**

**Main Directories and Files**:

* + controllers: Handles different operations for admins, bookings, cabs, and users.
  + middleware: Manages file uploads and seeds initial admin data.
  + models: Defines MongoDB schemas for different collections.
  + routers: Maps HTTP requests to the appropriate controller functions.
  + Root Files: .env, app.js, package.json, package-lock.json

**2. Key Directories and Files**

* + - **Controllers**: adminControllers.js, bookingControllers.js, cabControllers.js, userControllers.js
    - **Middleware**: seedAdmin.js, upload.js
    - **Models**: admin.js, booking.js, cab.js, user.js
    - **Routers**: adminRoutes.js, bookingRoutes.js, cabRoutes.js, userRoutes.js

**3. API Endpoints**

**User Authentication**: POST /login, POST /register

**Admin Authentication**: POST /adminlogin

**User Management**:

* + **For Users**: GET /getUser/, PUT /userEdit/

**Cab Management**:

* + **For Users**: GET /cabs, POST /cabs
  + **For Admins**: DELETE /deleteCar/

**Booking Management**:

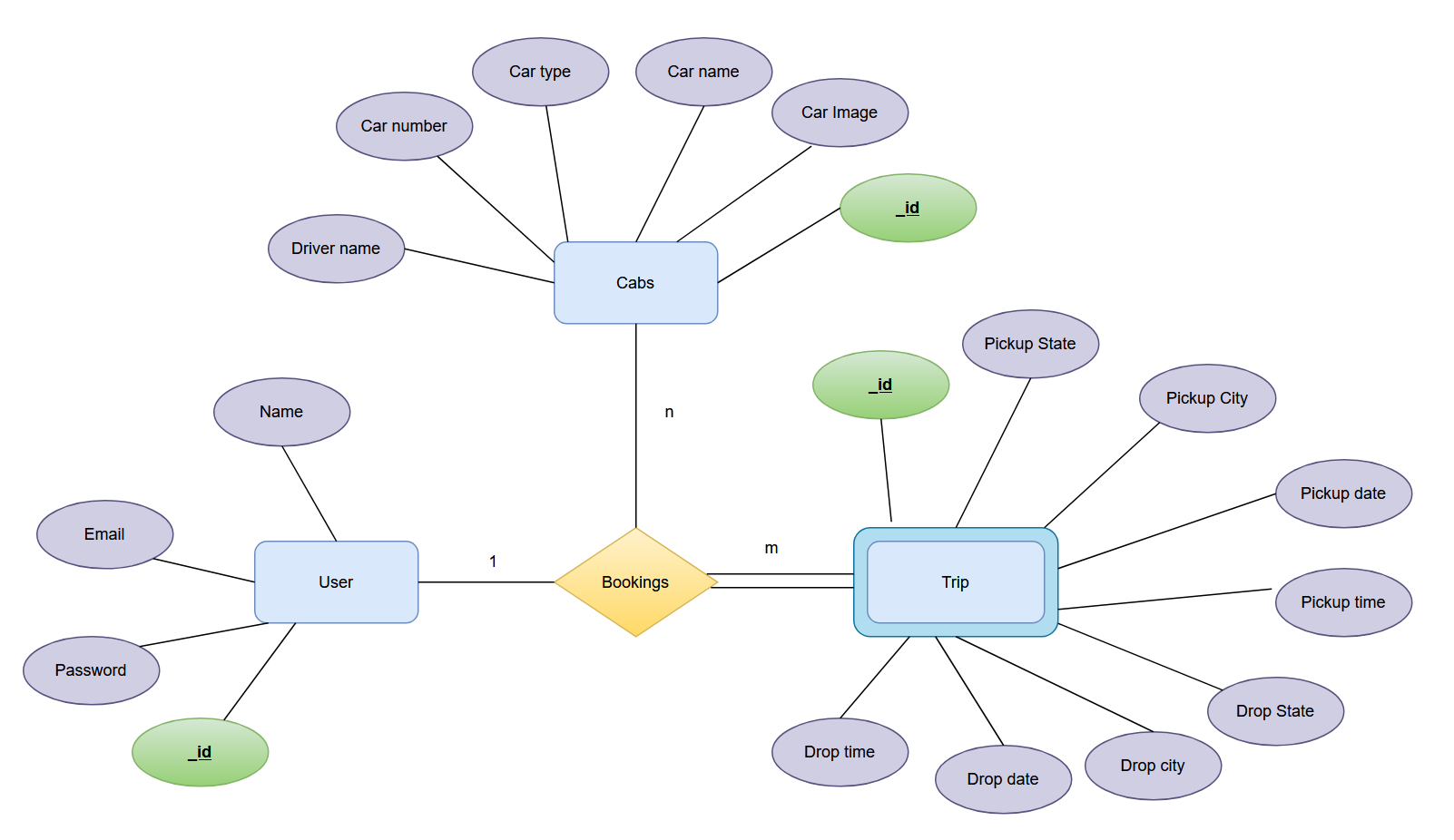
* + **For Users**: GET /getrides, POST /rides, GET /getrides/, DELETE /deleteride/
  + **For Admins**: GET /getAllUsers, DELETE /deleteUser/

**4. Integration with Frontend**

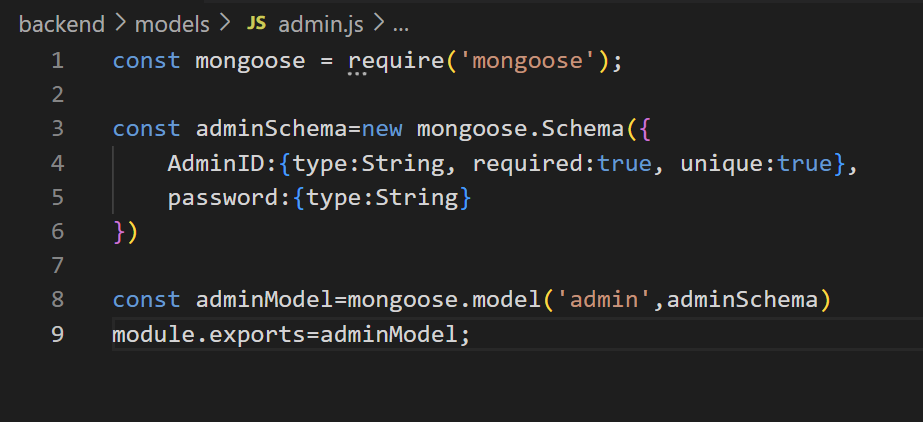
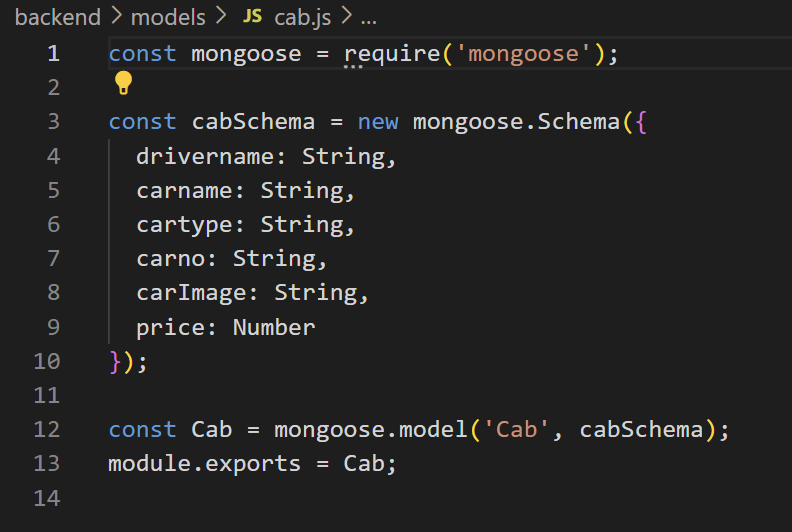
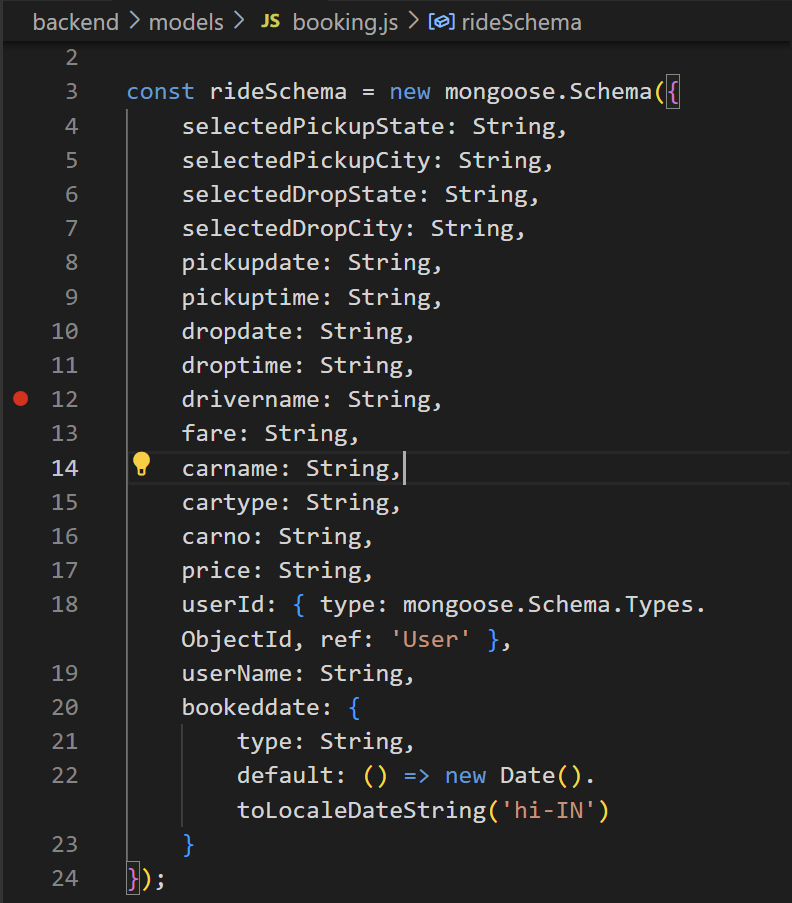
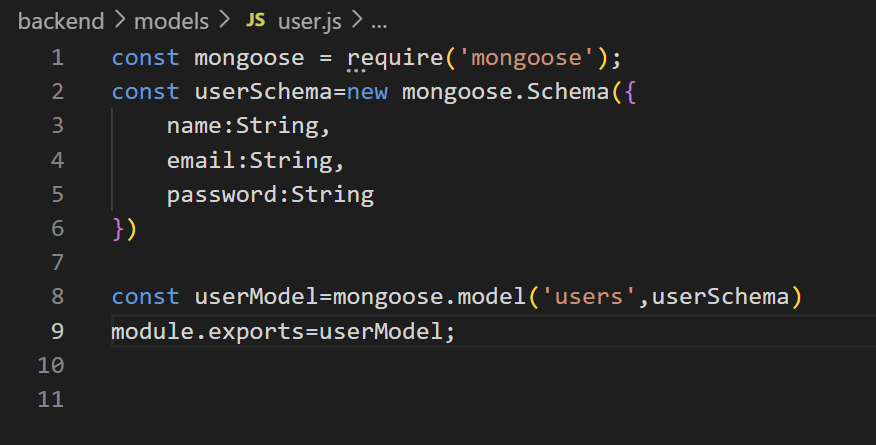
* + - The backend communicates with the frontend via RESTful APIs.
    - **User Authentication**: Manages user and admin authentication.
    - **Data Fetching**: Frontend components make API calls to retrieve and display data.
    - **Error Handling and Validation**: Managed using try-catch blocks and basic validation in application logic.

**5. Security Considerations**

* + - **Authentication**: Basic authentication for users and admins, with state management in React to secure protected routes.
    - **Data Encryption**: Future enhancements may include integrating encryption libraries for enhanced security.
* **Database:** Detail the database schema and interactions with MongoDB.



* + **Users**: \_id, name, email, password
  + **Bookings**: \_id, selectedPickupState, selectedPickupCity, selectedDropState, selectedDropCity, pickupdate, pickuptime, dropdate, droptime, drivername, fare, carname, carno, price, userID, userName, bookeddate
  + **Cabs**: \_id, drivername, carname, cartype, carno, carImage, price
  + **Admins**: \_id, AdminID, password

**Schemas:**

**4. Setup Instructions**

* + **Prerequisites:**

1.VS Code

2. Node js

3. npm

4. MongoDB

**Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

• Download: <https://nodejs.org/en/download/>

• Installation instructions:<https://nodejs.org/en/download/package-manager/>

**MongoDB:** Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

• Download:<https://www.mongodb.com/try/download/community>

• Installation instructions:<https://docs.mongodb.com/manual/installation/>

**Express.js:** Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

• Installation: Open your command prompt or terminal and run the following

   command: **npm install express**

**React js: React** is a JavaScript library for building client-side applications.

 And Creating Single Page Web-Appliaction

**Installations:**

**1. Install required tools and software:**

* Node.js.
* MongoDB.
* Create-react-app.

**2. Create project folders and files:**

* Client folders.
* Server folders.

**3. Install Packages:**

**Frontend npm Packages**

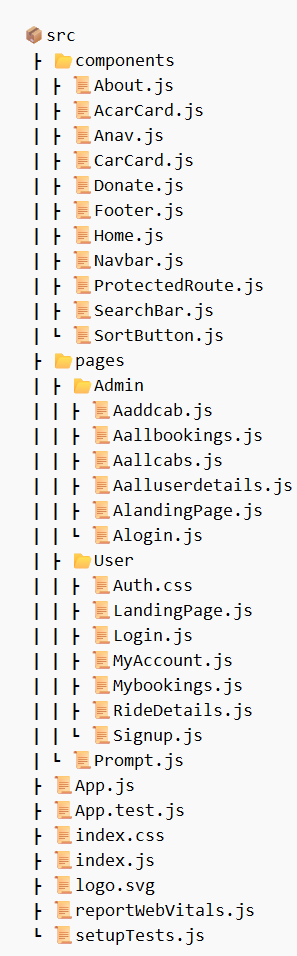
* Axios.
* React-Router –dom.

**Backend npm Packages**

* Express.
* Mongoose.
* Cors.
* dotenv

**5. Folder Structure**

* **Client:** Describe the structure of the React frontend.

****

* **Server:** Explain the organization of the Node.js backend.



**6. Running the Application**

Provide commands to start the frontend and backend servers locally.

**Quick  Start**

**Step 1 : open a terminal**

* cd frontend
* npm install 🡪 installs the node-modules that are listed in the package.json file
* npm start

**Step 2 : open another terminal**

* cd backend
* npm install
* node app.js

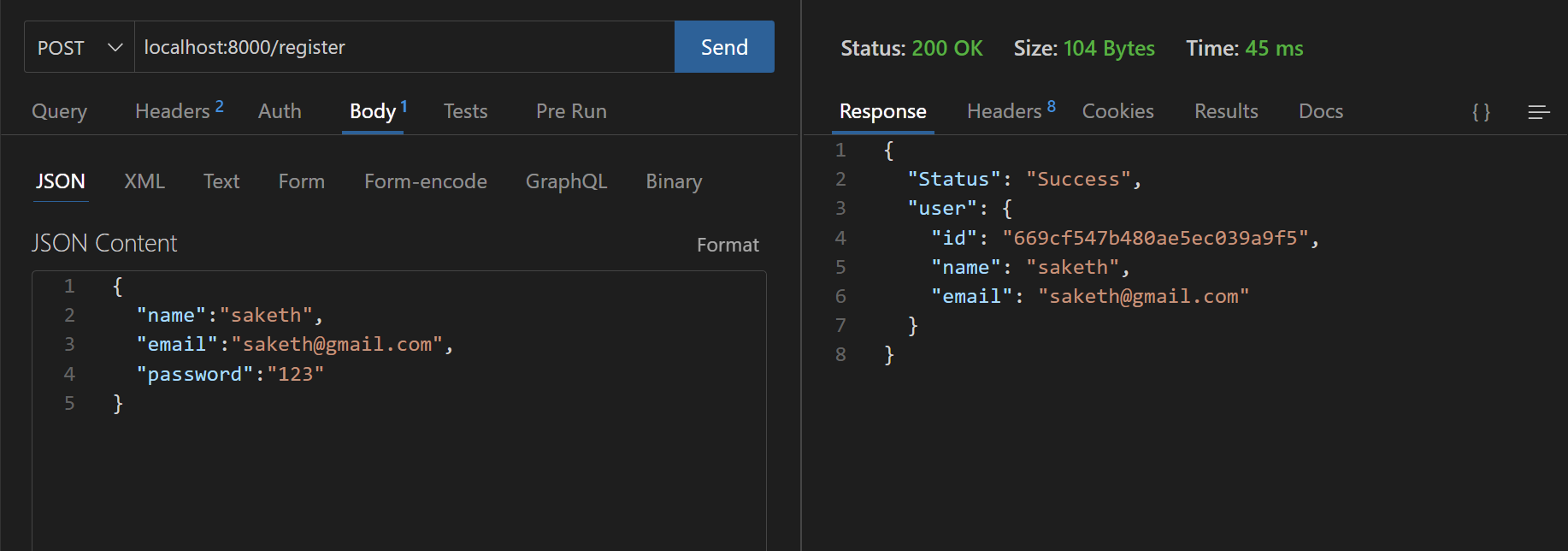
**7. API Documentation**

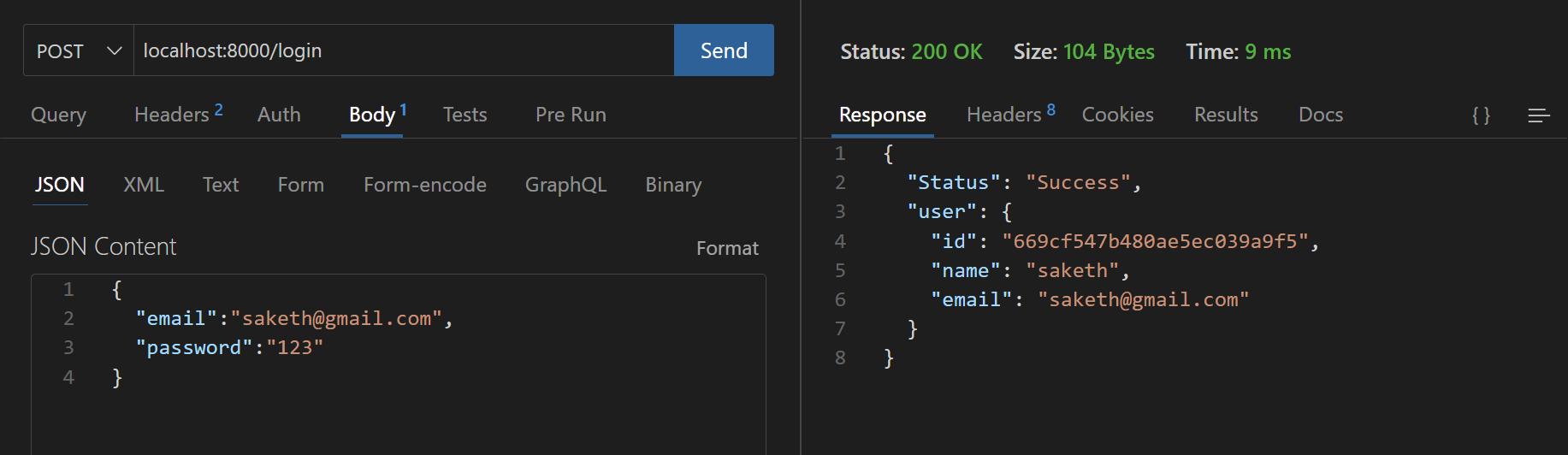
* Document all endpoints exposed by the backend.

The frontend communicates with backend APIs hosted on localhost:8000.

**Key Endpoints**:

* + **GET**: /api/data, /getrides, /cabs, /getAllUsers, /getUser/:id, /getrides/:id, /cab/:id
  + **POST**: /cabs, /adminlogin, /login, /rides, /register
  + **PUT**: /rides/:id, /userEdit/:id
  + **DELETE**: /deleteride/:id, /deleteCar/:id, /deleteUser/:id
* Include request methods, parameters, and example responses.





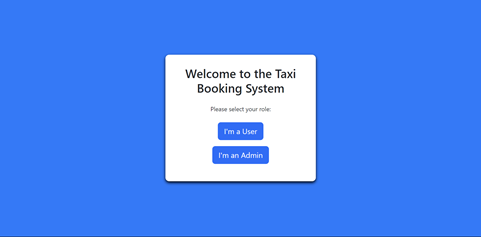
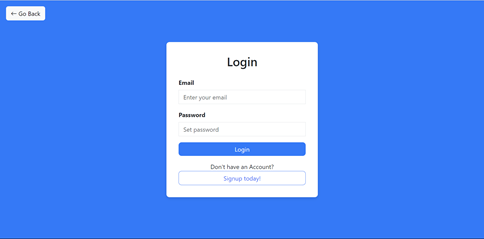
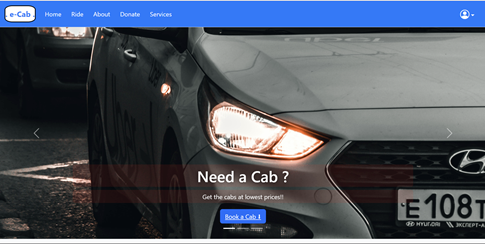
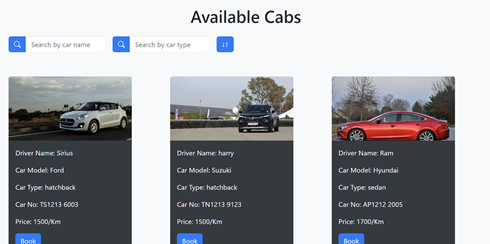
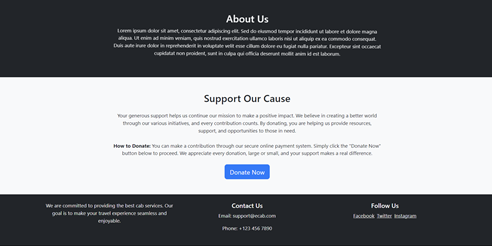
**8. Authentication**

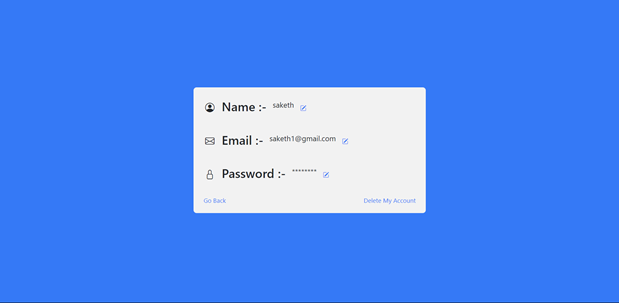
* **User Authentication**: Implemented basic user and admin authentication within the MERN stack. Upon successful login, the frontend stores the user state and authentication status to manage session persistence and secure access to protected routes.
* **Data Fetching**: Direct data fetching is employed where frontend components make API calls to retrieve and display necessary data. This includes fetching cab details, user bookings, and administrative data directly from the backend endpoints, ensuring seamless interaction and real-time data updates.

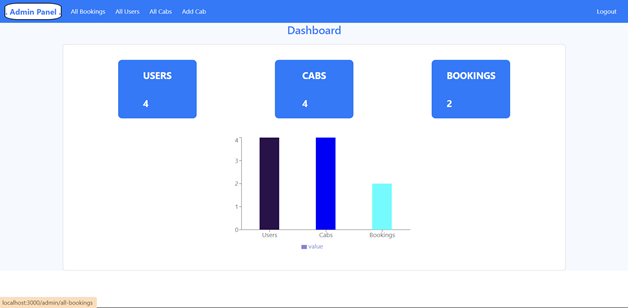
**9. User Interface**

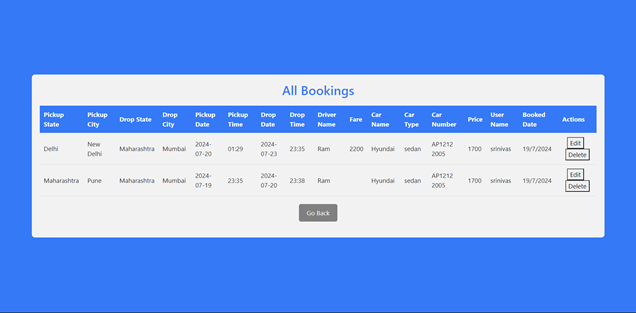
* Provide screenshots or GIFs showcasing different UI features.

Navigation Elements:

* 
* Forms :
* 
* Navbar + Carousel display :
* 
* Car cards + Search Bars:
* 
* 







**10. Testing**

* Describe the testing strategy and tools used.
* Manual Testing is performed . In Manual testing , Initially, test objectives are defined, and a test plan is created outlining the scope, approach, resources, and schedule. Test cases and scenarios are then developed to cover various functionalities and real-world use cases, with appropriate test data prepared. During test execution, testers manually perform the tests, document any defects encountered, and verify fixes. The process concludes with evaluating results, preparing reports, and conducting a retrospective to identify improvements.

**11. Screenshots or Demo**

* Provide screenshots or a link to a demo (if available) to showcase the application.
* <https://www.youtube.com/watch?v=4ltNgzm1Bws>

**12. Known Issues**

* Past Date Pickup: The system allows booking rides with pickup dates in the past.
* Fare Calculation: Rides can be booked even if the fare is not calculated.
* Unauthenticated Access: Users can access cabs without logging in.
* Admin Registration: There's no functionality for registering new admins.

**13. Future Enhancements**

* **Google Maps API Integration: Plan to integrate with Google Maps for improved functionality.**
* **Checkout Page: Adding a checkout page for a more complete real-world app experience.**
* **Password Security: Use bcrypt for securely hashing passwords in the real-world app.**