Cab Booking App

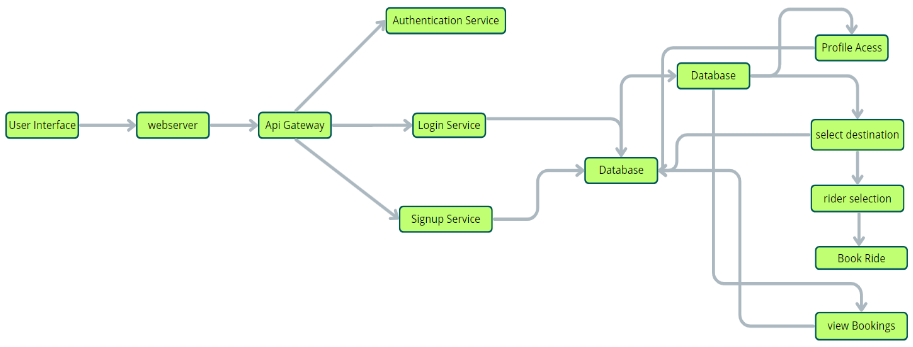
Project Description:

The Cab Booking App is a web application made using the MERN Stack (MongoDB, Express.js, React.js, Node.js).
It helps users to book a cab online by filling details like name, phone number, pickup place, drop place, and travel date.
The frontend is built with React.js, which gives a simple and user-friendly design.
The backend is built with Node.js and Express.js, which handles the booking process.
All booking details are stored safely in MongoDB database.
This app reduces manual work and makes cab booking faster and easier.
An admin can also view and manage all bookings (if required).

Scenario:

On a busy morning, a user needs to travel to another location. He opens the Cab Booking App on his phone. He enters his name, phone number, pickup, drop, and travel date. He clicks the Book Cab button, and the system saves the booking. A confirmation message appears showing the cab is booked.The admin checks the booking details and arranges a cab. The driver reaches the pickup location on time. The user travels safely and comfortably to the destination.

Technical Architecture:



In This Architecture Diagram:

1.User Input (Booking Request)

- The process starts when a passenger enters details like pickup location, drop location, and time in the mobile/web app.
- This raw request needs to be processed and matched with available cabs.

2. Backend Server (Request Handling)

- The booking request goes to the Node.js + Express backend server.This step ensures faster delivery without losing much quality.
- It verifies the request, checks authentication, and passes it for processing.

3. Database (MongoDB Storage)

- The server communicates with MongoDB where user profiles, driver details, cab availability, and booking history are stored.
- If it’s a new booking, details are saved. If it’s a past booking, records are fetched.

4. Cab Matching Engine (Ride Allocation)

- The system searches for nearby available drivers.
- Based on location, availability, and preference, the **best driver is matched** with the passenger.

5. Network Layer (Communication)

- The server sends notifications to both **driver and passenger** via the internet.
- Maps API (like Google Maps) is used to calculate routes, distance, and ETA.

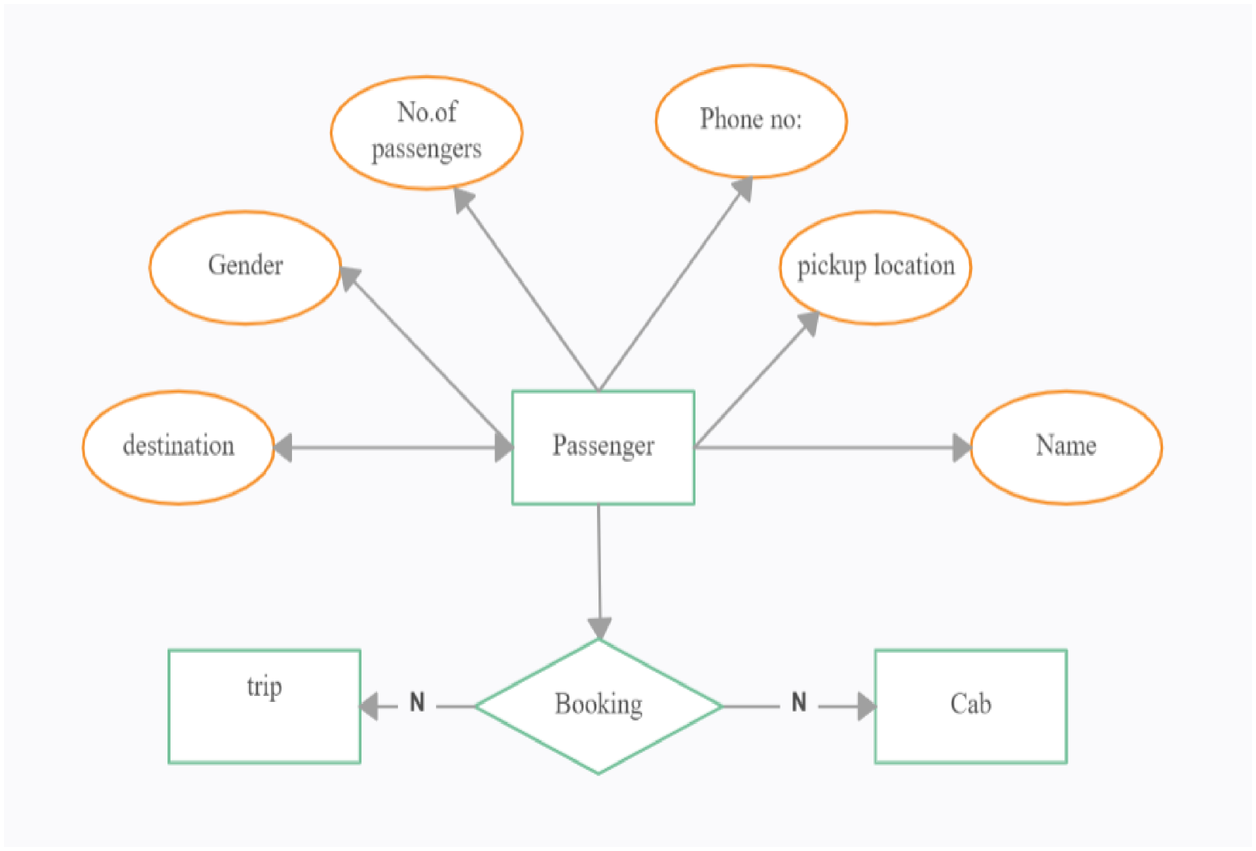
6. Relay / Notification Service (Efficiency & Scaling)

- To handle many users at once, a **relay service** (like Firebase Cloud Messaging or Socket.io) delivers real-time updates (ride status, driver location).
- This prevents the main server from being overloaded.

7.End Users (Passenger & Driver Apps)

- Finally, both passenger and driver apps get updates in real-time:
- Passenger sees driver details, cab info, and estimated arrival.
- Driver sees pickup details and navigation route.

ER Diagram:



PREREQUISITES:

1. **Node.js and npm** – Install Node.js to run JavaScript on the server side. npm comes with Node.js and helps to manage packages.
2. **MongoDB** – Install MongoDB to store user, driver, and booking details. You can use it locally or a cloud version.
3. **Express.js** – Install Express.js to create server-side APIs and handle routes.
4. **React.js** – Use React to build the frontend (user interface) for booking, login, and admin dashboard.
5. **HTML, CSS, JavaScript** – Basic knowledge to design and add interactivity in the frontend.
6. **Mongoose** – Use Mongoose to connect Node.js with MongoDB and manage data easily.
7. **Git & GitHub** – For version control and keeping track of code changes.
8. **Code Editor (VS Code)** – Install a code editor like Visual Studio Code to write and manage code.

1. **Getting Started**
   Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.
2. **Quik Start**
- npx create-react-app my-app
- cd my-app
- npm start
- If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app or yarn global remove create-react-app to ensure that npx always uses the latest version.
3. **Create a new React project:**
- Choose or create a directory where you want to set up your React project.
- Open your terminal or command prompt.
- Navigate to the selected directory using the cd command.
- Create a new React project by running the following command: npx create-react-app your-app -name.Wait for the project to be created:
- This command will generate the basic project structure and install the necessary dependencies
4. **Navigate into the project directory:**
- After the project creation is complete, navigate into the project directory by running the following command**: cd your-app-name**
5. **Start the development server:**
- To launch the development server and see your React app in the browser, run the following command: **npm start** .
- The npm start will compile your app and start the development server.
- Open your web browser and navigate to http://localhost:3000 to see your React app.
- You have successfully set up React on your machine and created a new React project. You can now start building your app by modifying the generated project files in the src directory.
- Please note that these instructions provide a basic setup for React. You can explore more ad- vanced configurations and features by referring to the official React documentation: https://react.dev/
6. **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
7. **Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.
8. **Front-end Library:** Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.
9. **Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.
   - Git: Download and installation instructions can be found at: https://git-scm.com/downloads
10. **Development Environment:** Choose a code editor or Integrated Development Environment (IDE)

that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from https://code.visualstudio.com/download
- Sublime Text: Download from https://www.sublimetext.com/download
- WebStorm: Download from https://www.jetbrains.com/webstorm/download

Choose an IDE:

● VS Code: https://code.visualstudio.com/download

● Sublime Text: https://www.sublimetext.com/download

● WebStorm: https://www.jetbrains.com/webstorm/download

Flow Chart:

This flowchart outlines the **user journey** and **system processes** for a typical cab booking application like Uber or Ola. It includes key stages such as onboarding, ride booking, driver matching, trip tracking, and payment.

Flowchart Steps (User Journey)

Here is the textual layout of the flowchart:

Backend/System Components (optional for technical flow)

These steps can be shown alongside the user flow (parallel processing):

- Authentication Service (JWT, OAuth)
- Location Services (GPS, Google Maps API)
- Driver Matching Algorithm
- Push Notification System
- Payment Gateway (Stripe, Razorpay, Paytm, etc.)
- Ride History / Analytics Module
- Admin Dashboard

[Start / App Launch]

↓

[User Login / Register]

↓

[Enter Pickup & Drop Location]

↓

[Search for Nearby Cabs]

↓

[Show Available Ride Options]

↓

[User Selects Cab Type (Mini, Sedan, SUV)]

↓

[Confirm Booking]

↓

[Match with Driver]

↓

[Driver Accepts Ride]

↓

[Live Ride Tracking Begins]

↓

[Pickup Passenger]

↓

[Trip In Progress]

↓

[Drop-off at Destination]

↓

[Calculate Fare]

↓

[Payment (Cash / Card / Wallet)]

↓

[User Rating & Feedback]

↓

[End]

Project Setup and Configuration Documentation

The project setup involves installing Node.js and Git, creating a structured folder system for the frontend and backend, initializing the backend with Express, setting up MongoDB for data storage, and implementing version control using Git.

■Install Node.js, MongoDB, and Git

Run these commands one by one in your terminal:

bash

# Verify installations

node --version

npm --version

git --version

mongod –version

If any command fails, download:

- Node.js from nodejs.org
- Git from git-scm.com
- MongoDB from mongodb.com

■Create Project Structure

Set up folders for **frontend** and **backend**:

- mkdir cab-booking-app
- cd cab-booking-app
- mkdir backend   # Create backend folder
- npx create-react-app frontend # Create React frontend

■Set Up React Frontend

Go to frontend/:

- cd ../frontend
- npm start

npx create-react-app frontend

This will generate:

- frontend/src/ for React components
- frontend/public/ for static assets
- All necessary configuration files

■Initialize Backend
Run these commands to set up your Node.js backend:

- mkdir backend
- cd backend
- npm init -y

■Install Backend Dependencies
Install the required libraries:

Bash

```
npm install express mongoose cors dotenv bcrypt jsonwebtoken

npm install --save-dev nodemon
```

■Set Up Git Version Control
Initialize Git and create your first commit:

Bash

```
git init

git add .

git commit -m "Initial OTT Platform setup"
```

With this setup, we will have:

- React frontend ready
- Express backend initialized
- MongoDB integration support
- Git for version control

■ Start Development Servers Open two terminal windows and run:

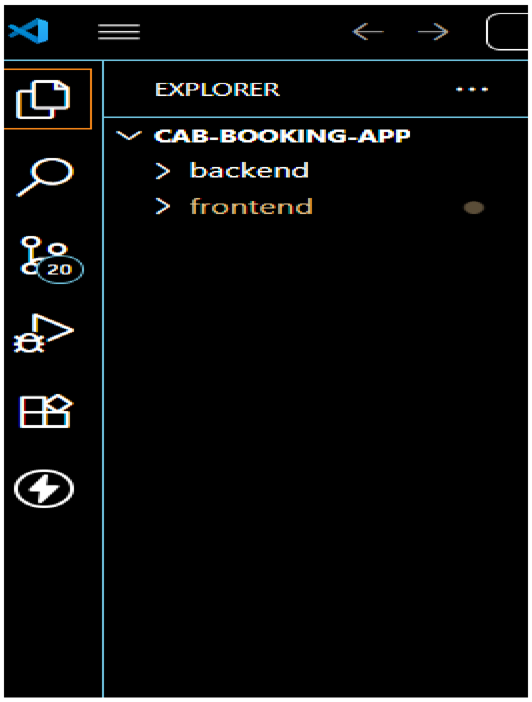
bash

# Terminal 1 (frontend)

cd frontend npm start

# Terminal 2 (backend)

cd Backend node server.js



Backend Development:

Initialize Express Server

- Navigate to your Backend folder and install required dependencies:

cd backend

npm install express cors body-parser mongoose jsonwebtoken dotenv bcryptjs

● Create a new index.js file as your main server file:

bash

```
touch index.js
```

- Configure Environment Variables
- Create a .env file in your Backend folder:

bash

```
touch .env
```

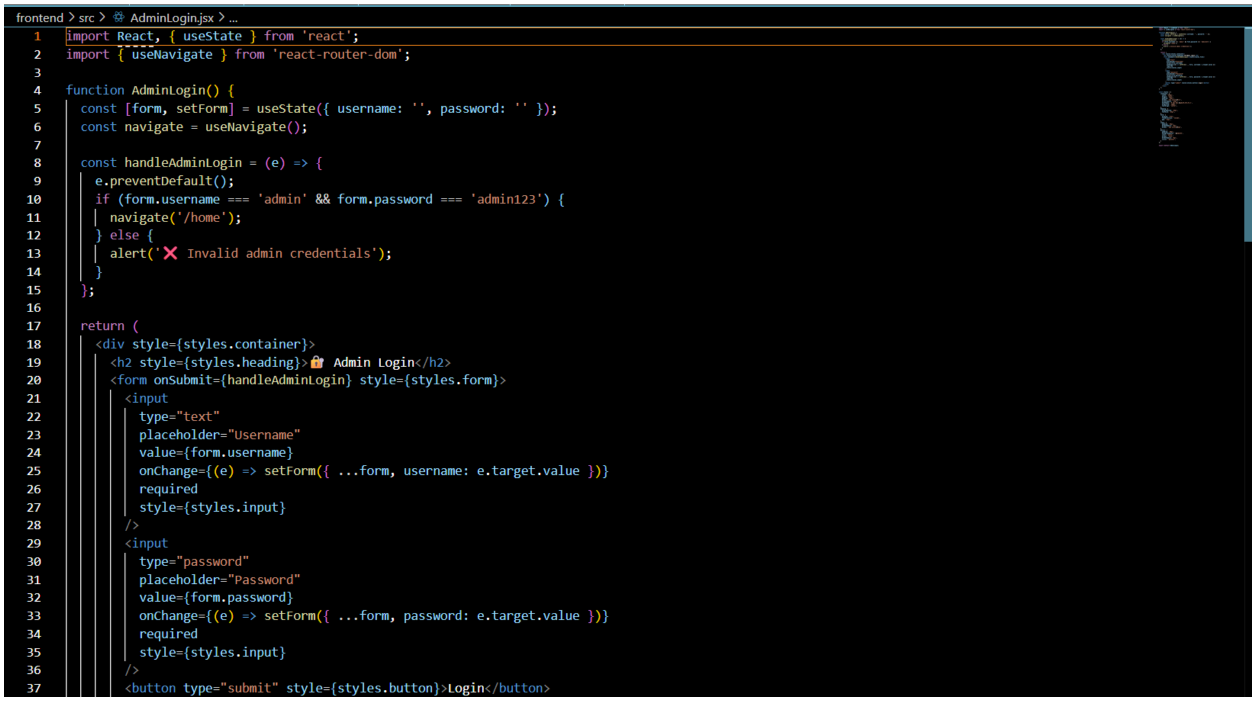
- Add these configurations to your .env file:

Env

```
PORT=3000

MONGO_URI=mongodb+srv://23a55a0517:Joshna1234@myatlasclusteredu.tvnin72.mongodb.net/?retryWrites=true&w=majority&appName=myAtlasClusterEDU

JWT_SECRET=612e0ab063d27c2b0a8c8698123708e6

NODE_ENV=development
```

- ## Basic Server Setup

  - Add this code to your indexserver.js file:

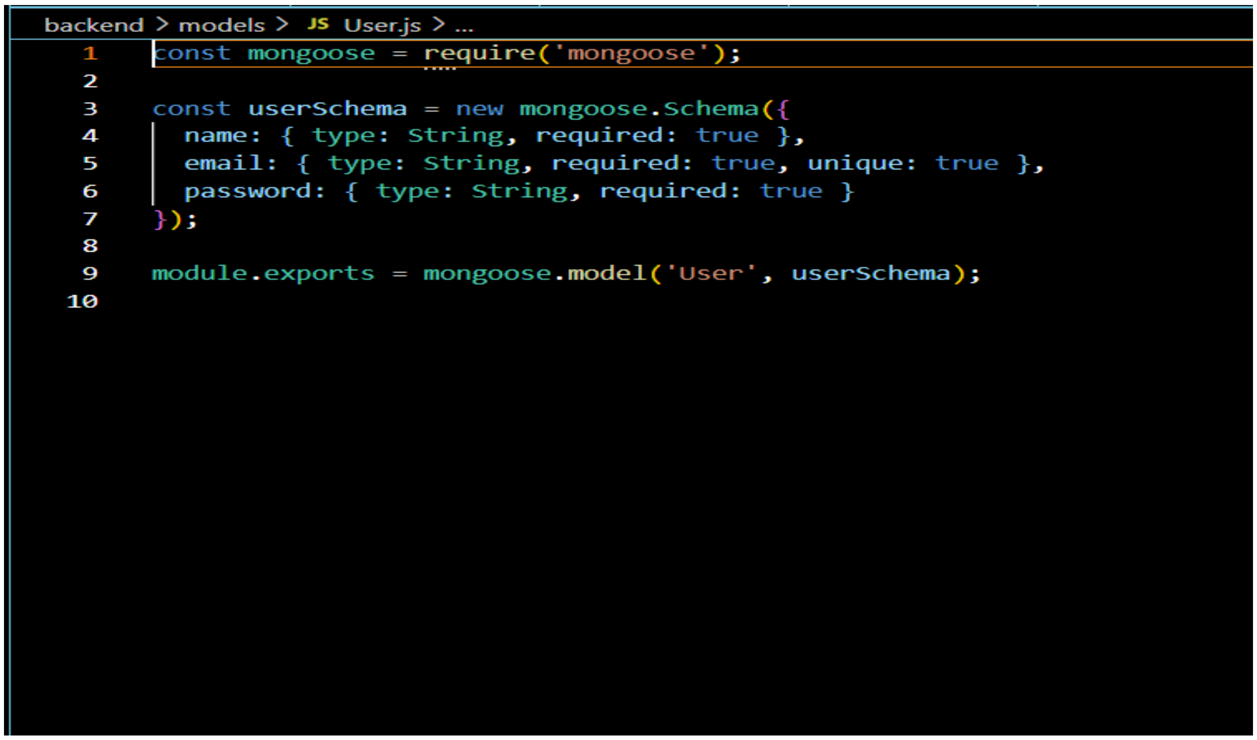


frontend > src > AdminLogin.jsx > ...

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

function AdminLogin() {
  const [form, setForm] = useState({ username: '', password: '' });
  const navigate = useNavigate();

  const handleAdminLogin = (e) => {
    e.preventDefault();
    if (form.username === 'admin' && form.password === 'admin123') {
      navigate('/home');
    } else {
      alert('❌ Invalid admin credentials');
    }
  };

  return (
    <div style={styles.container}>
      <h2 style={styles.heading}>🔐 Admin Login</h2>
      <form onSubmit={handleAdminLogin} style={styles.form}>
        <input
          type="text"
          placeholder="Username"
          value={form.username}
          onChange={(e) => setForm({ ...form, username: e.target.value })}
          required
          style={styles.input}
        />
        <input
          type="password"
          placeholder="Password"
          value={form.password}
          onChange={(e) => setForm({ ...form, password: e.target.value })}
          required
          style={styles.input}
        />
        <button type="submit" style={styles.button}>Login</button>
```

Authentication Setup

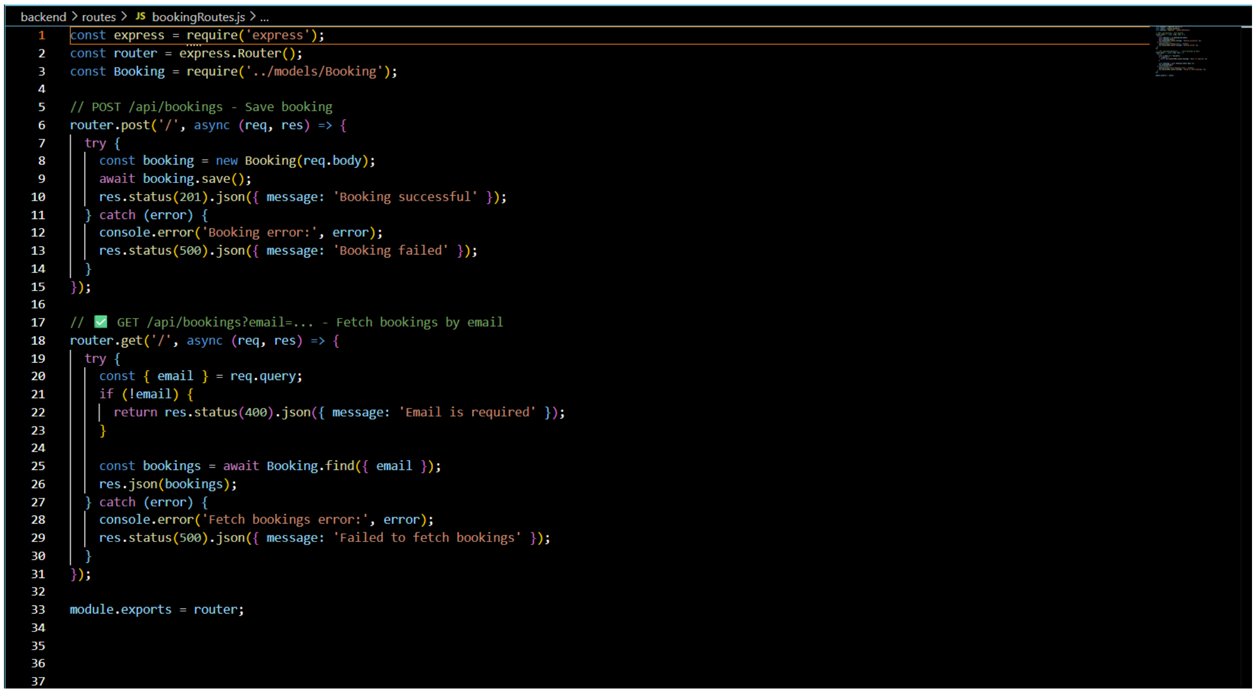
- Create a models/User.js file for user schema:



backend > models > JS User.js > ...

```javascript
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true }
});

module.exports = mongoose.model('User', userSchema);
```

●Create a routes/auth.js file for authentication routes:



backend > routes > JS bookingRoutes.js > ...

```javascript
const express = require('express');
const router = express.Router();
const Booking = require('../models/Booking');

// POST /api/bookings - Save booking
router.post('/', async (req, res) => {
  try {
    const booking = new Booking(req.body);
    await booking.save();
    res.status(201).json({ message: 'Booking successful' });
  } catch (error) {
    console.error('Booking error:', error);
    res.status(500).json({ message: 'Booking failed' });
  }
});

// ✅ GET /api/bookings?email=... - Fetch bookings by email
router.get('/', async (req, res) => {
  try {
    const { email } = req.query;
    if (!email) {
      return res.status(400).json({ message: 'Email is required' });
    }

    const bookings = await Booking.find({ email });
    res.json(bookings);
  } catch (error) {
    console.error('Fetch bookings error:', error);
    res.status(500).json({ message: 'Failed to fetch bookings' });
  }
});

module.exports = router;
```

Backend Development Steps:

1. Initialize Project Structure:

- Create a backend directory/folder for the server-side code.

- Set up a new Node.js project with `npm init` to create a `package.json` file.

2. Install Required Dependencies:

- Install Express.js: `npm install express`

- Install MongoDB driver: `npm install mongoose`

- Install additional packages as needed (e.g., cors, bcryptjs, dotenv, etc.).

3. Set Up Express Server:

- Create an `index.js` file to set up the Express server.

- Import Express and initialize the server.

- Define middleware functions like `cors`, `body-parser`, and error handling.

4. Connect to MongoDB:

- Configure MongoDB connection using Mongoose in a `config.js` file. - Create models for User, Movie, and Review using Mongoose Schemas.

5. Implement User Authentication:

- Implement user registration and login routes with hashing passwords using bcryptjs.

- Generate JWT (JSON Web Tokens) for user authentication and authorization.

6. Develop Movie and Review Routes:

- Create CRUD (Create, Read, Update, Delete) routes for movies and reviews.

- Implement functionalities like adding movies to favourites, writing reviews, etc.

7. Implement Middleware for Authentication:

- Create a middleware function (`authMiddleware.js`) for verifying JWT tokens. - Use the middleware to protect routes that require authentication.

8. Testing:

- Write unit tests and integration tests using testing frameworks like Jest or Mocha. - Test each route, middleware, and database interactions to ensure they work as expected.

9. Environment Configuration:

- Use `dotenv` to manage environment variables for database connection, JWT secret, etc.

- Create a `.env` file to store sensitive information.

10. Error Handling and Logging:

- Implement error handling middleware to catch and handle errors gracefully. - Set up logging to track server activities and errors for debugging.

11. Deployment:

- Choose a cloud provider (e.g., AWS, Heroku, Digital Ocean) to deploy the backend.

- Configure deployment settings, environment variables, and deploy the backend application.

12. Documentation:

- Document the API endpoints, request/response formats, and usage in a README.md file.

- Provide instructions on setting up, running, and testing the backend application.

These steps provide a structured approach to developing the backend of the Streamify OTT platform. Each step focuses on building specific functionalities, implementing security measures, testing, and preparing for deployment

Database Development:

Create database in cloud video link:-

https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/vi • Install Mongoose.

• Create database connection.

Reference Video of connect node with mongoDB database:

https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?ussharing

Reference Article:

https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/

Schema use-case:

Review Schema:

- Schema: reviewModel.js

- Model: 'Review'

- The Review schema captures user reviews with details like reviewId, username, userId, datetime, movieId, and review text for movies.

- It is used to store user-generated reviews associated with specific movies f other users to read and reference.

Movie Schema:

- Schema: movieModel.js

- Model: 'Movie'

- The Movie schema stores movie details including title, id, genre_ids, poster_path, release_date, vote_average, and mediaType fetched from the API.

- It is used to manage and display movie information to users, enabling them to browse, select, and watch movies on the Streamify platform.

User Schema:

- Schema: userModel.js

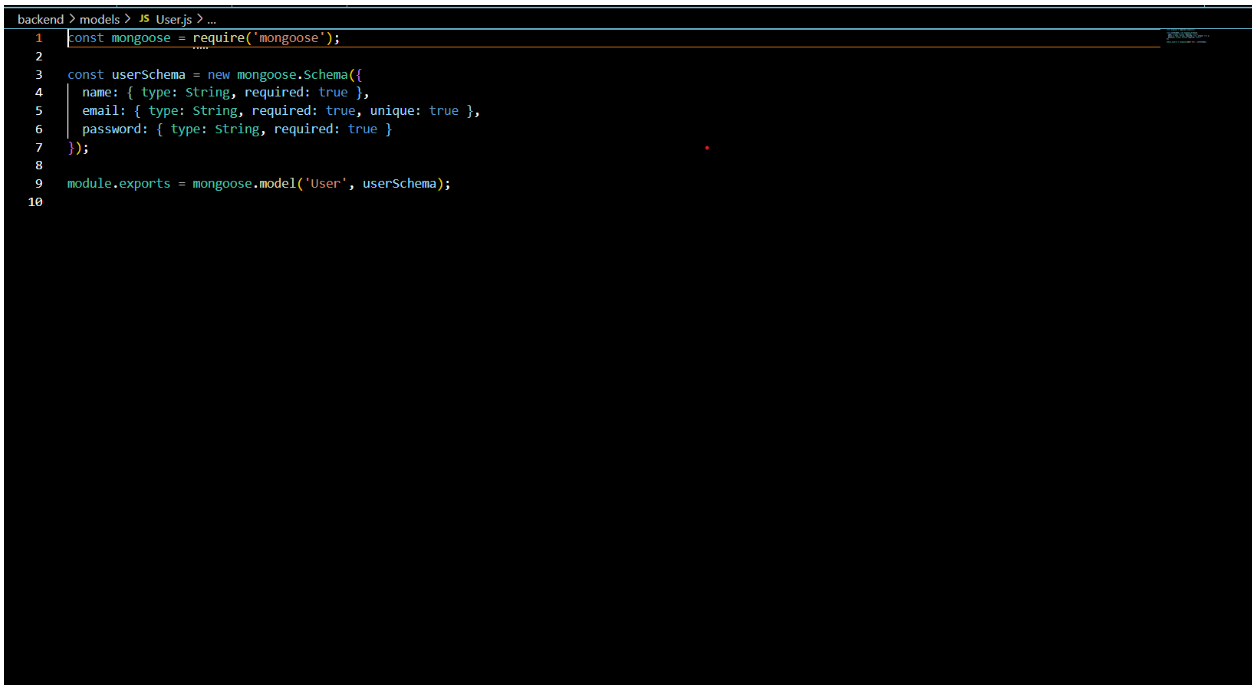
- Model: 'User'

- The User schema manages user registration and authentication with fields li username, email, and password.

- It is used to store and validate user information, ensuring secure access and personalized experiences on the Streamify platform.

- The email field is marked as unique to ensure that each user has a unique ema address.

Reference Images:



backend > models > User.js > ...

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

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true }
});

module.exports = mongoose.model('User', userSchema);
```

Frontend Development:

- **1. Setup React Application:**

• Create a React app in the client folder.

• Install required libraries

• Create required pages and components and add routes.

2.Design UI components:

• Create Components.

• Implement layout and styling.

• Add navigation.

3.Implement frontend logic:

• Integration with API endpoints.

• Implement data binding.

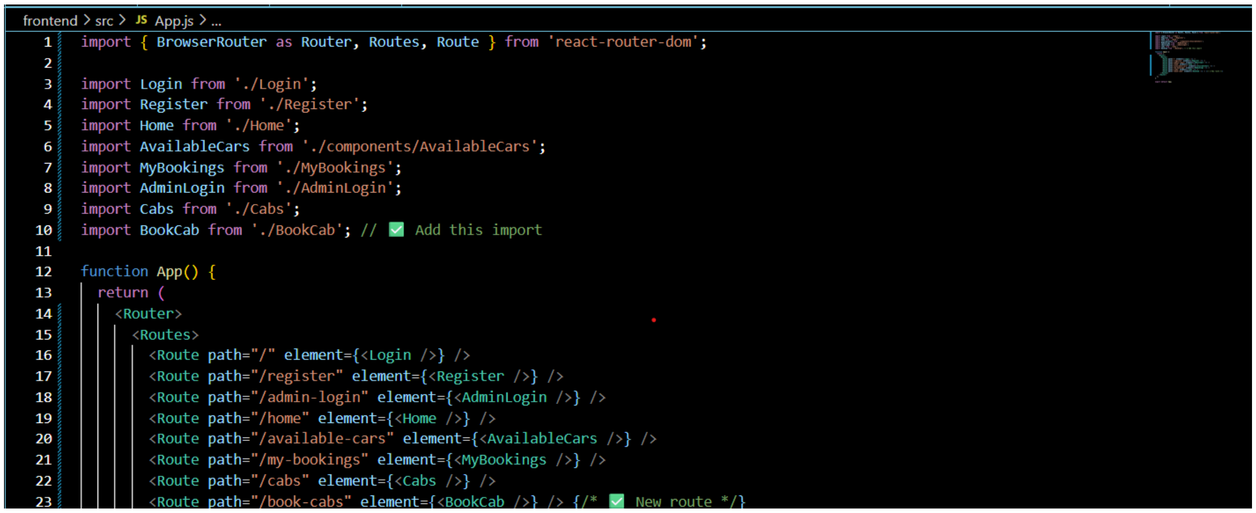
Reference Video Link:

https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?u=sharing

Reference Article Link:

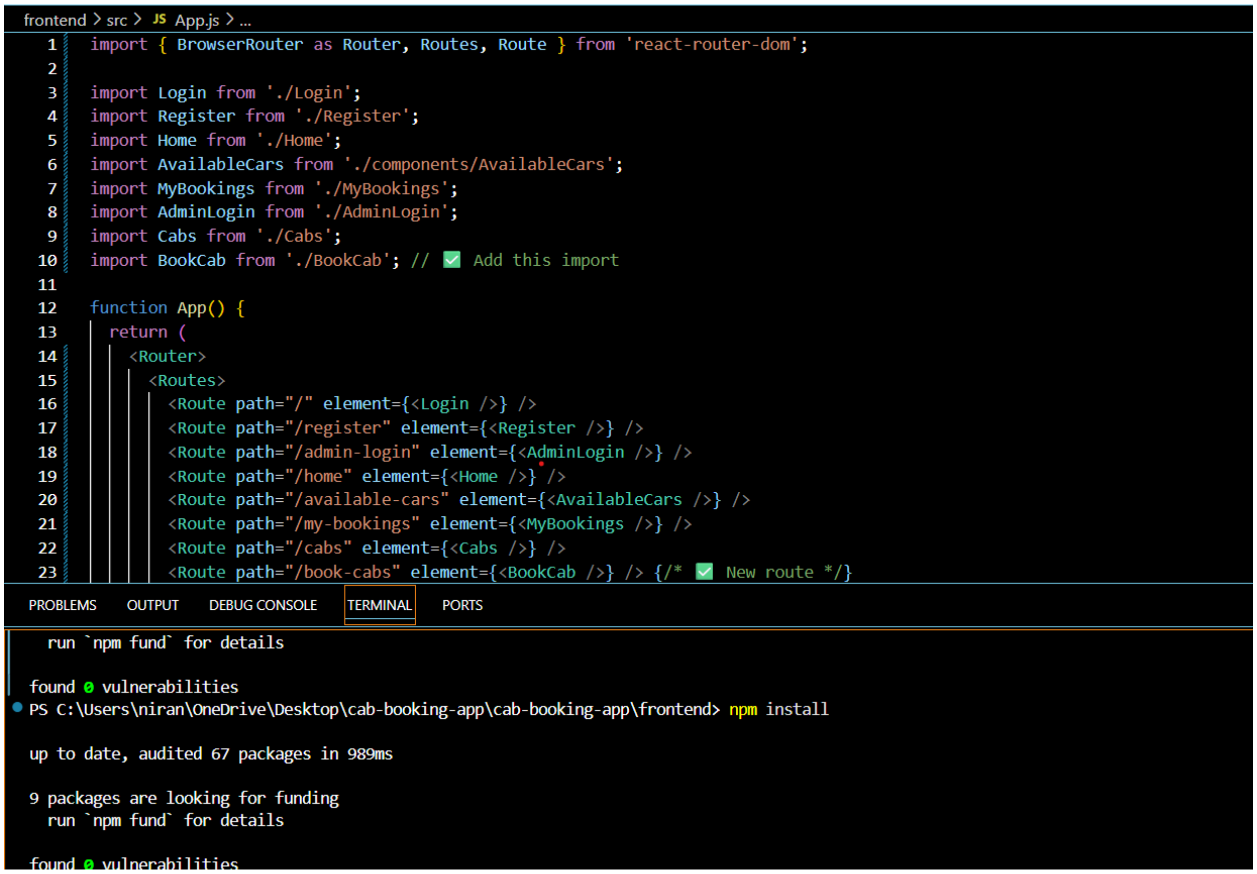
https://www.w3schools.com/react/react_getstarted.asp

Reference Image:



frontend > src > JS App.js > ...

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

import Login from './Login';
import Register from './Register';
import Home from './Home';
import AvailableCars from './components/AvailableCars';
import MyBookings from './MyBookings';
import AdminLogin from './AdminLogin';
import Cabs from './Cabs';
import BookCab from './BookCab'; // ✅ Add this import

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/admin-login" element={<AdminLogin />} />
        <Route path="/home" element={<Home />} />
        <Route path="/available-cars" element={<AvailableCars />} />
        <Route path="/my-bookings" element={<MyBookings />} />
        <Route path="/cabs" element={<Cabs />} />
        <Route path="/book-cabs" element={<BookCab />} /> {/* ✅ New route */}
```



frontend > src > JS App.js > ...

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

import Login from './Login';
import Register from './Register';
import Home from './Home';
import AvailableCars from './components/AvailableCars';
import MyBookings from './MyBookings';
import AdminLogin from './AdminLogin';
import Cabs from './Cabs';
import BookCab from './BookCab'; // ✅ Add this import

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/admin-login" element={<AdminLogin />} />
        <Route path="/home" element={<Home />} />
        <Route path="/available-cars" element={<AvailableCars />} />
        <Route path="/my-bookings" element={<MyBookings />} />
        <Route path="/cabs" element={<Cabs />} />
        <Route path="/book-cabs" element={<BookCab />} /> {/* ✅ New route */}
```

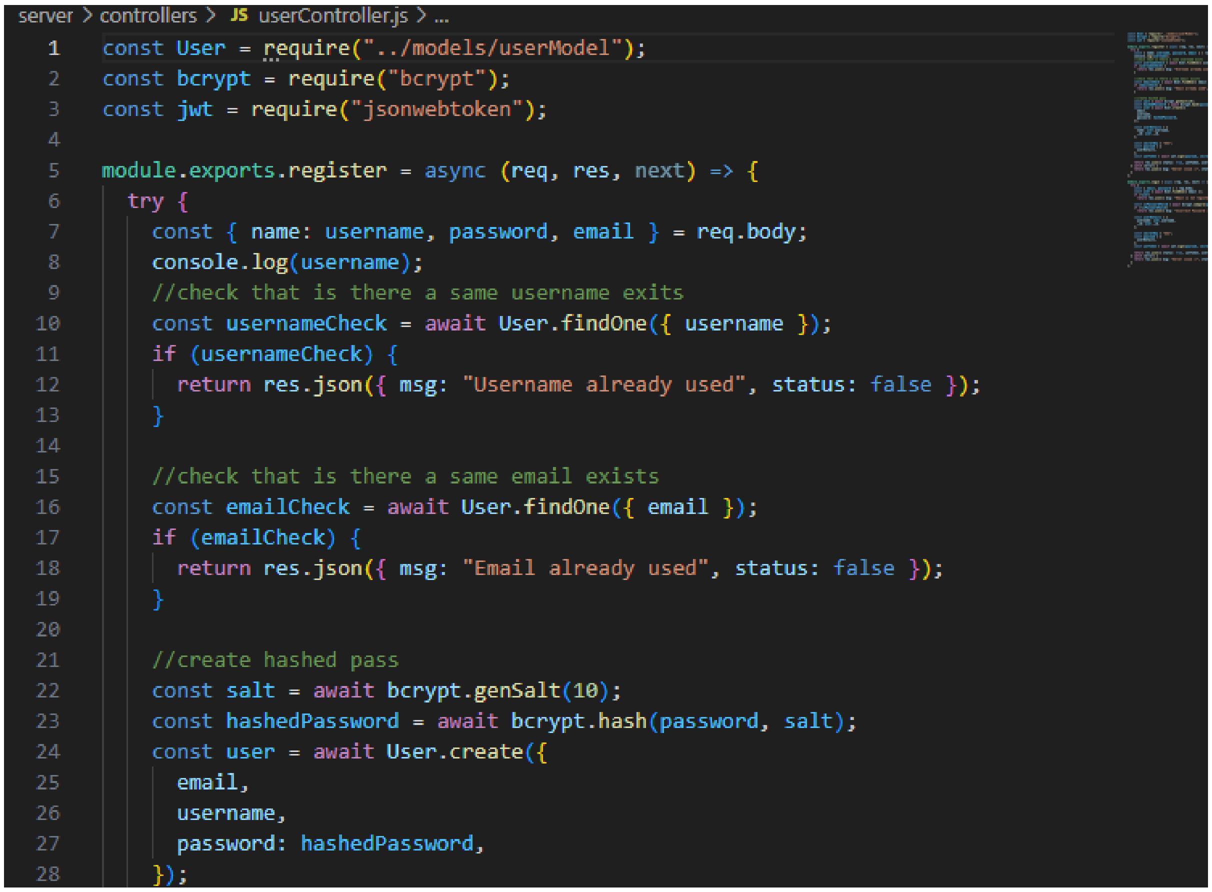
```
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\niran\OneDrive\Desktop\cab-booking-app\cab-booking-app\frontend> npm install

up to date, audited 67 packages in 989ms

9 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Project Implementation & Execution:

User Authentication:

Login - Backend

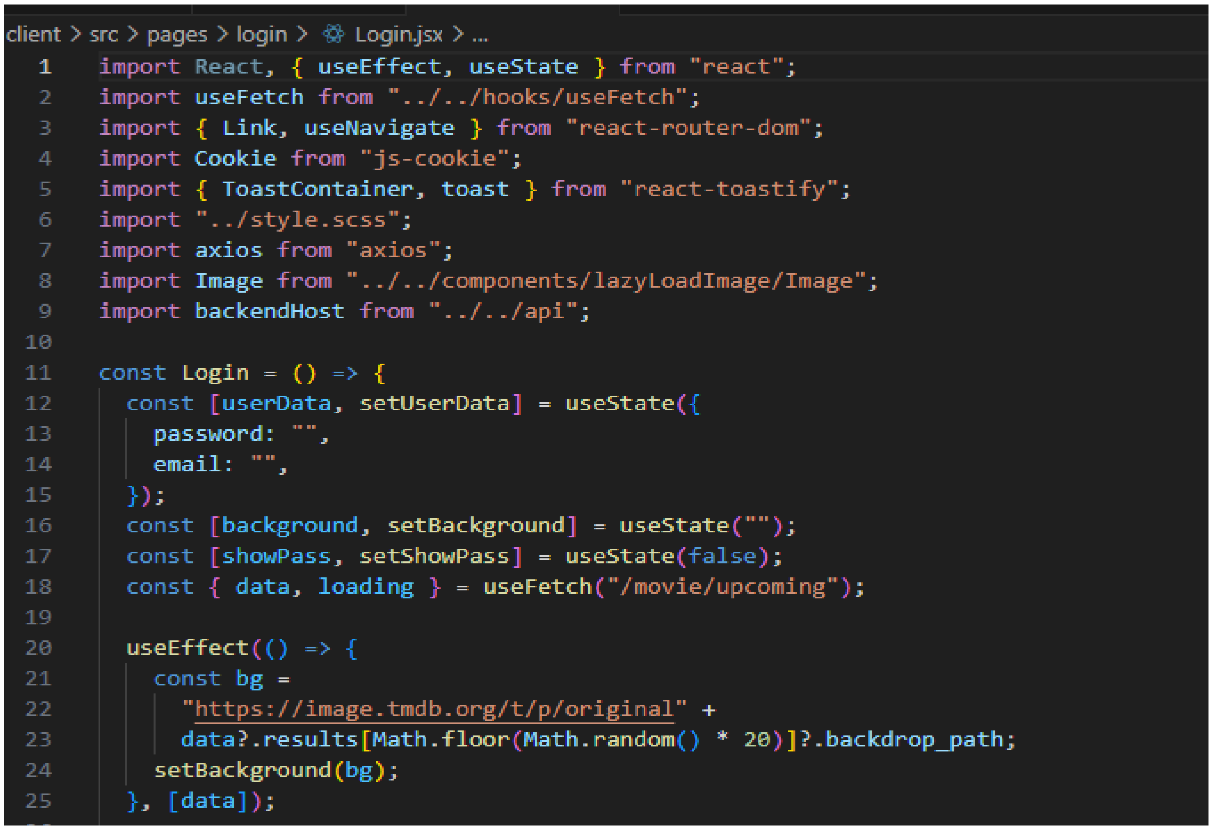
Now, here we define the functions to handle http requests from the client f authentication.



server > controllers > JS userController.js > ...

```javascript
const User = require("../models/userModel");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");

module.exports.register = async (req, res, next) => {
  try {
    const { name: username, password, email } = req.body;
    console.log(username);
    //check that is there a same username exits
    const usernameCheck = await User.findOne({ username });
    if (usernameCheck) {
      return res.json({ msg: "Username already used", status: false });
    }

    //check that is there a same email exists
    const emailCheck = await User.findOne({ email });
    if (emailCheck) {
      return res.json({ msg: "Email already used", status: false });
    }

    //create hashed pass
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    const user = await User.create({
      email,
      username,
      password: hashedPassword,
    });
```

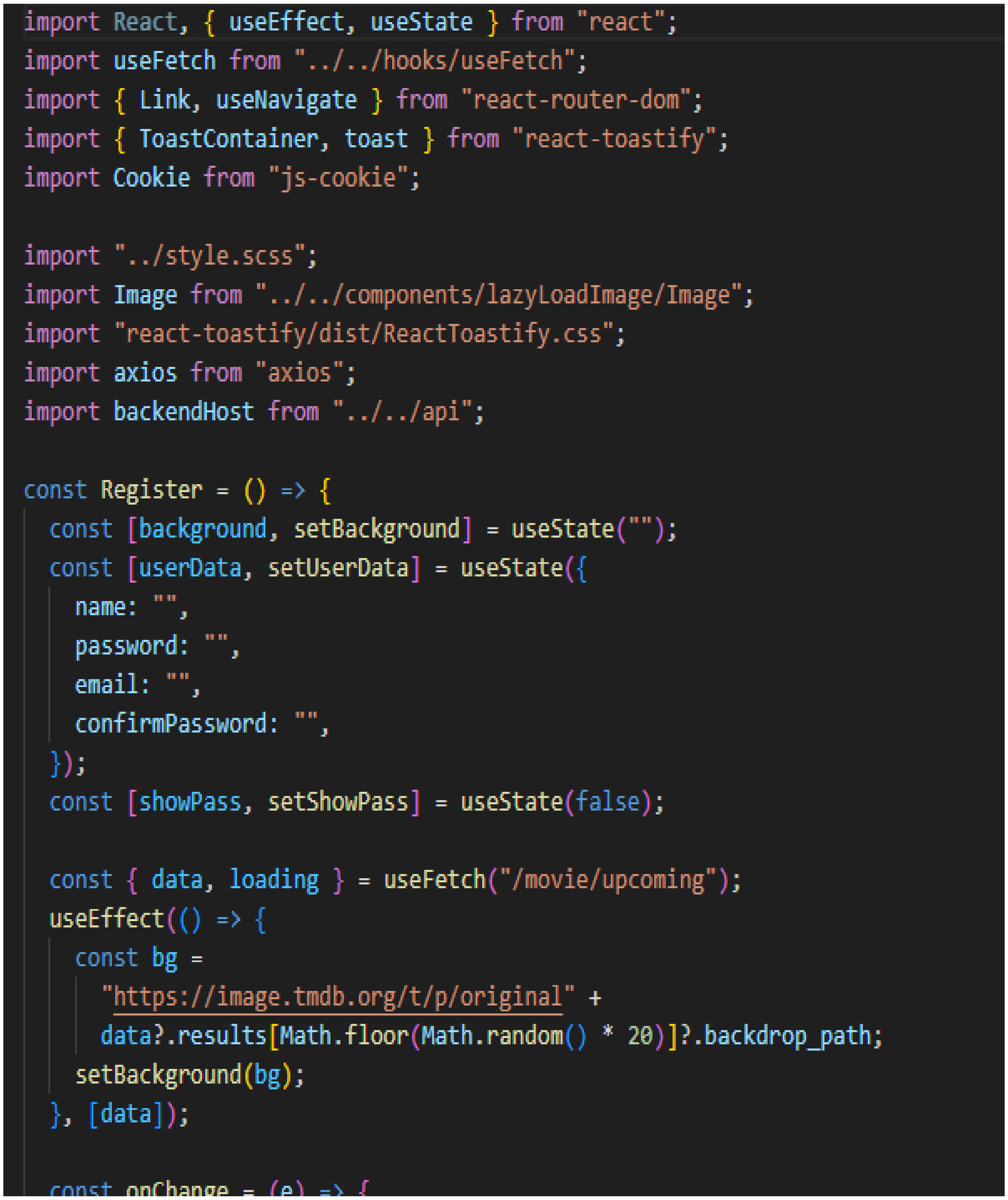
Login – Frontend:



client > src > pages > login > Login.jsx > ...

```jsx
import React, { useEffect, useState } from "react";
import useFetch from "../../hooks/useFetch";
import { Link, useNavigate } from "react-router-dom";
import Cookie from "js-cookie";
import { ToastContainer, toast } from "react-toastify";
import "../style.scss";
import axios from "axios";
import Image from "../../components/lazyLoadImage/Image";
import backendHost from "../../api";

const Login = () => {
  const [userData, setUserData] = useState({
    password: "",
    email: "",
  });
  const [background, setBackground] = useState("");
  const [showPass, setShowPass] = useState(false);
  const { data, loading } = useFetch("/movie/upcoming");

  useEffect(() => {
    const bg =
      "https://image.tmdb.org/t/p/original" +
      data?.results[Math.floor(Math.random() * 20)]?.backdrop_path;
    setBackground(bg);
  }, [data]);
```

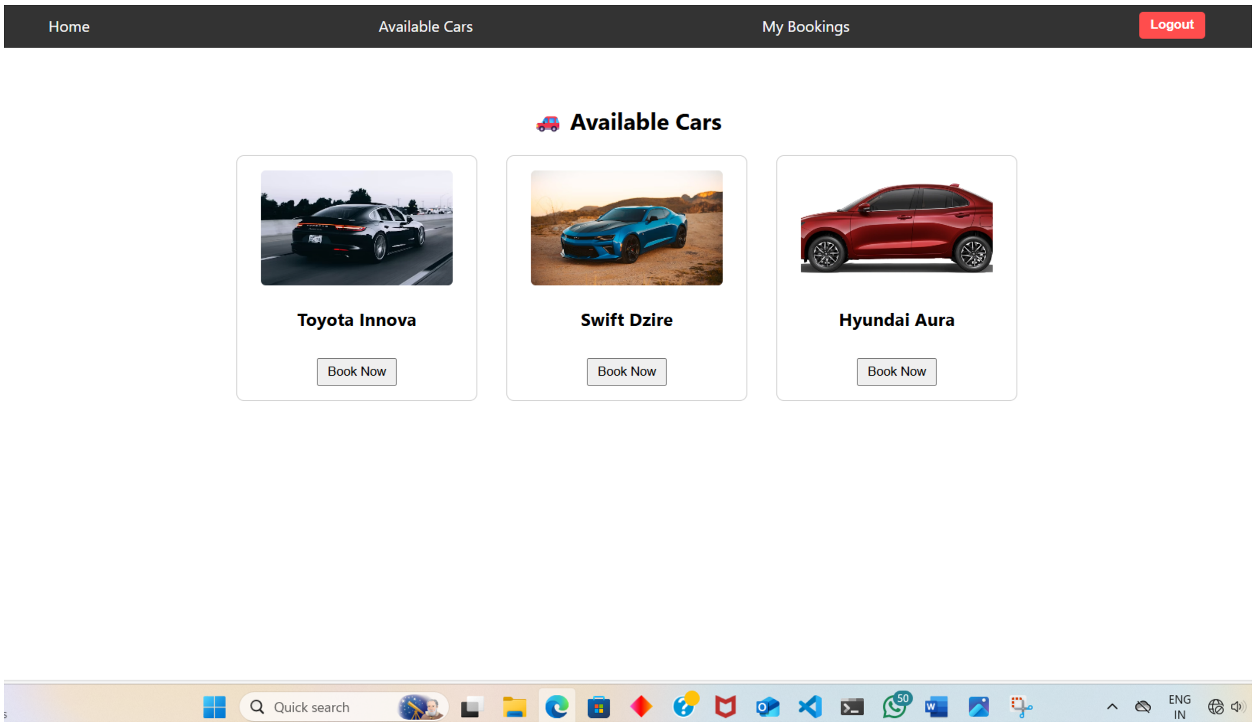
Register – Frontend:



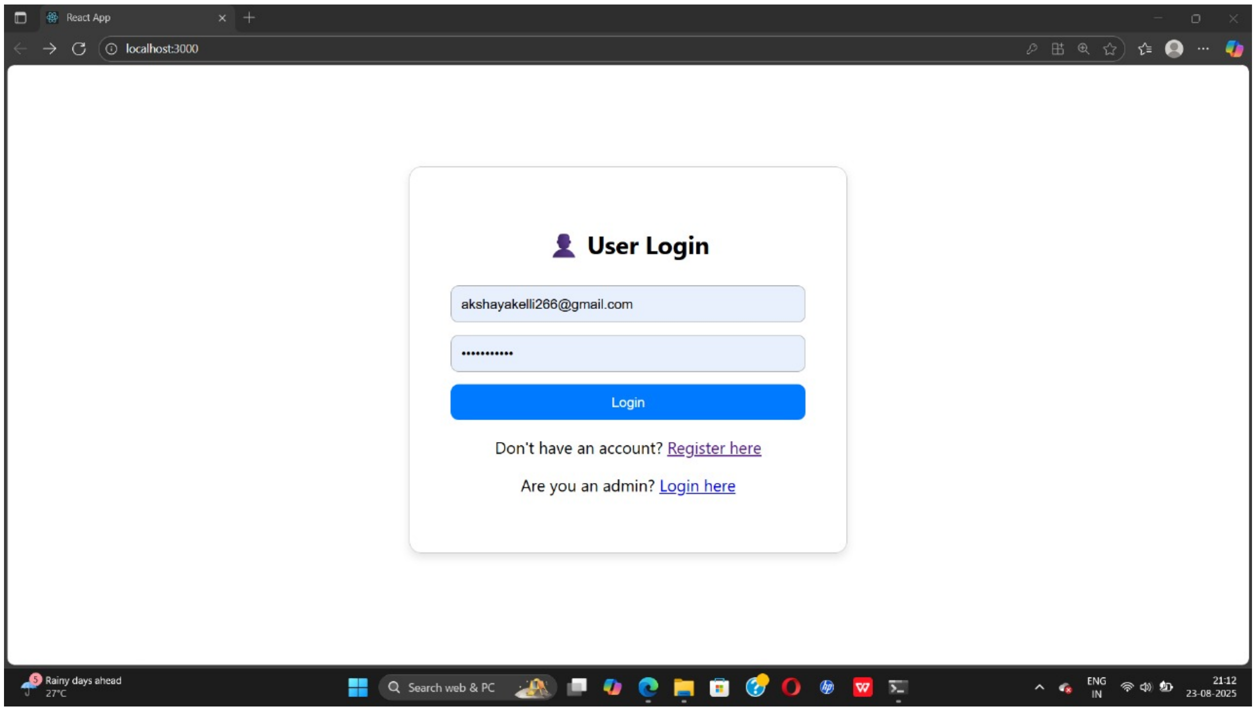
```
import React, { useEffect, useState } from "react";
import useFetch from "../../hooks/useFetch";
import { Link, useNavigate } from "react-router-dom";
import { ToastContainer, toast } from "react-toastify";
import Cookie from "js-cookie";

import "../style.scss";
import Image from "../../components/lazyLoadImage/Image";
import "react-toastify/dist/ReactToastify.css";
import axios from "axios";
import backendHost from "../../api";

const Register = () => {
  const [background, setBackground] = useState("");
  const [userData, setUserData] = useState({
    name: "",
    password: "",
    email: "",
    confirmPassword: "",
  });
  const [showPass, setShowPass] = useState(false);

  const { data, loading } = useFetch("/movie/upcoming");
  useEffect(() => {
    const bg =
      "https://image.tmdb.org/t/p/original" +
      data?.results[Math.floor(Math.random() * 20)]?.backdrop_path;
    setBackground(bg);
  }, [data]);

  const onChange = (e) => {
```

Output Screenshots

Home page: Contains several carousels of top rated, upcoming, popular movies.



Login Page:

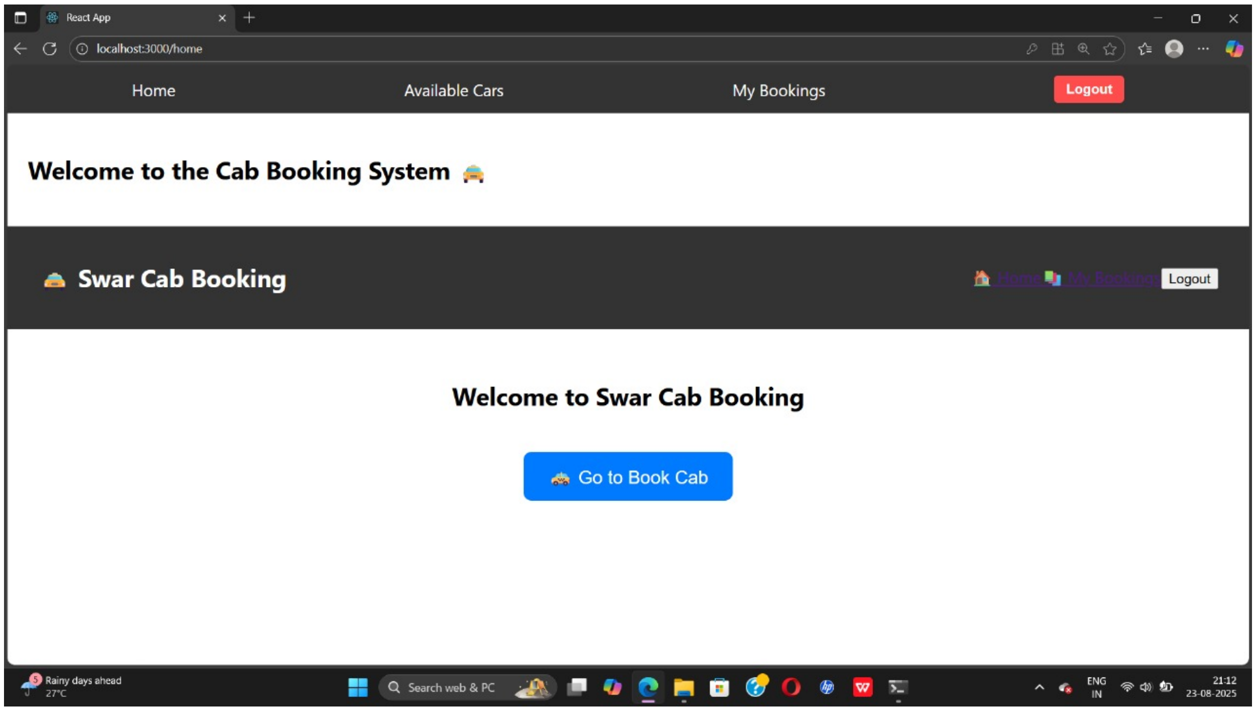

User Login

akshayakelli266@gmail.com

Don't have an account? Register here

Are you an admin? Login here

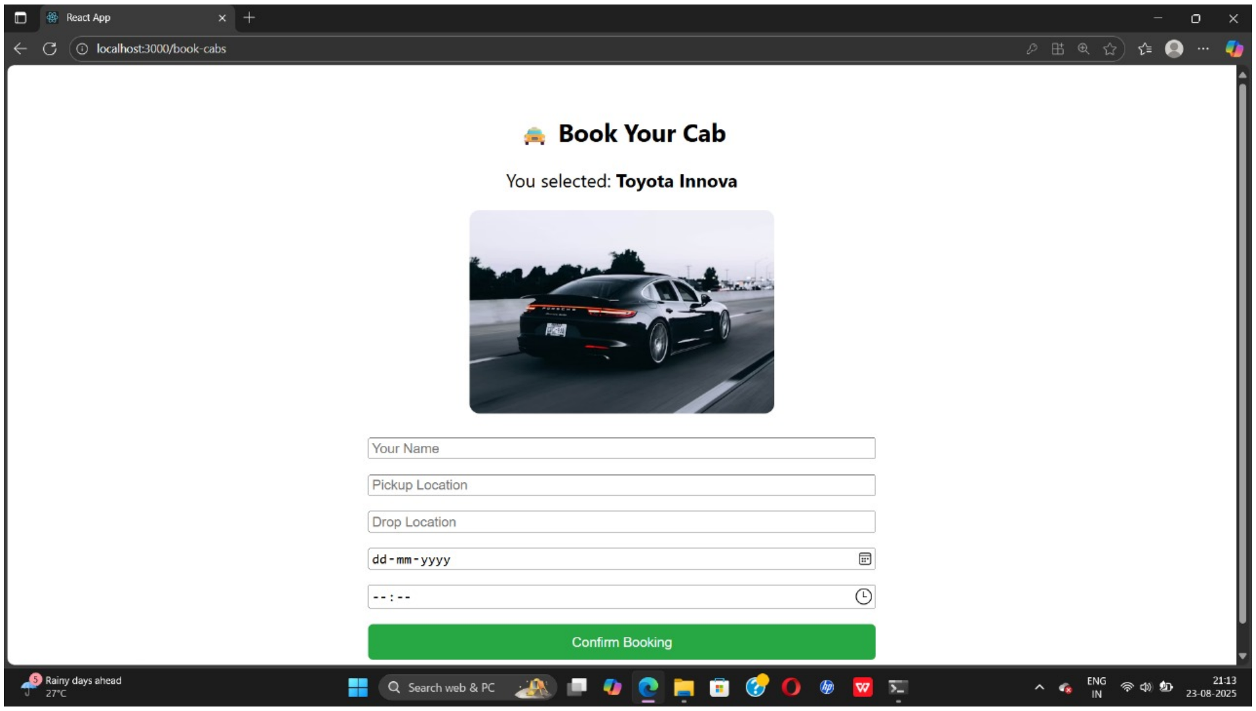
Booking Conformation Page:

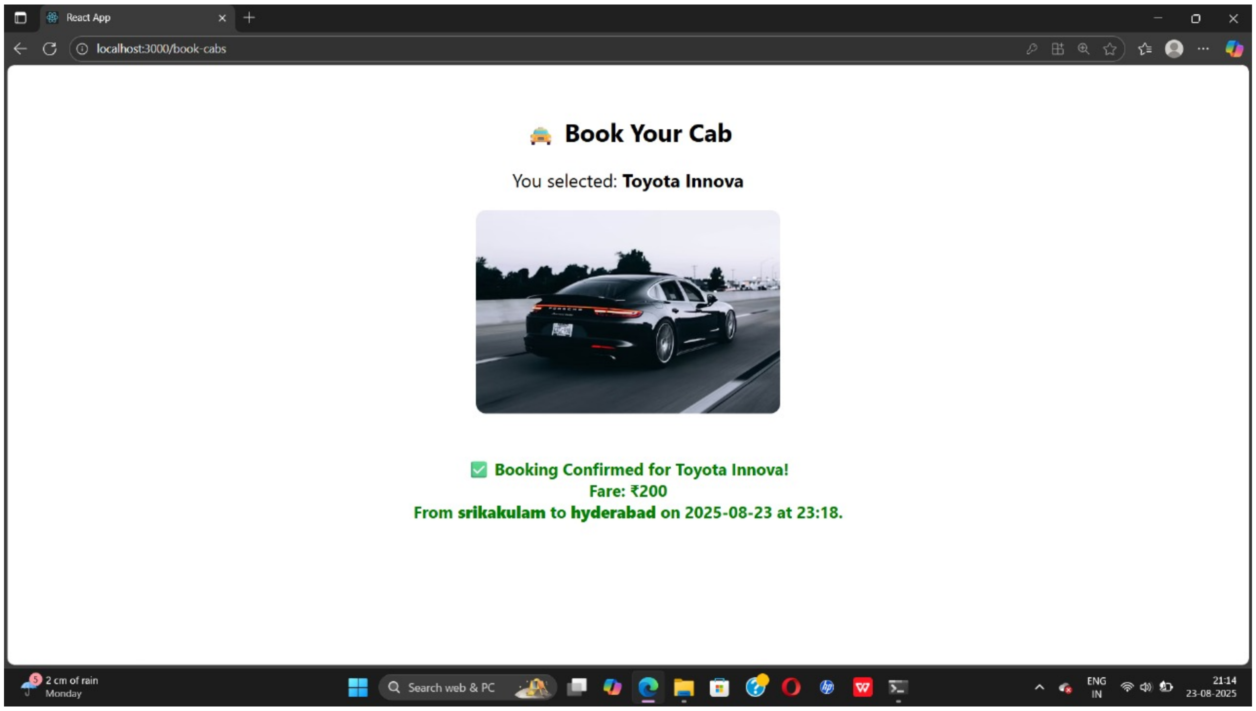

Welcome to the Cab Booking System 🚕

Swar Cab Booking

Welcome to Swar Cab Booking

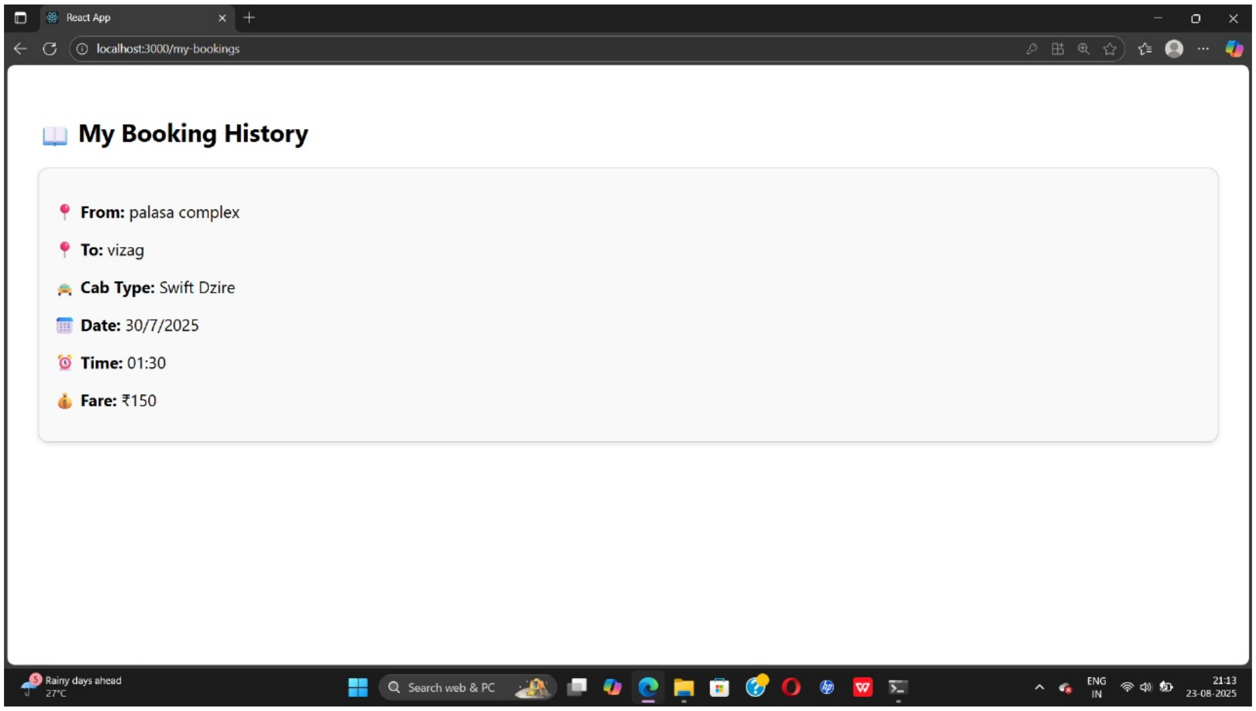
Cab Details Page:





✅ Booking Confirmed for Toyota Innova!
Fare: ₹200
From **srikakulam** to **hyderabad** on 2025-08-23 at 23:18.

My Booking History Page:



React App

localhost:3000/my-bookings

📖 **My Booking History**

📍 **From:** palasa complex

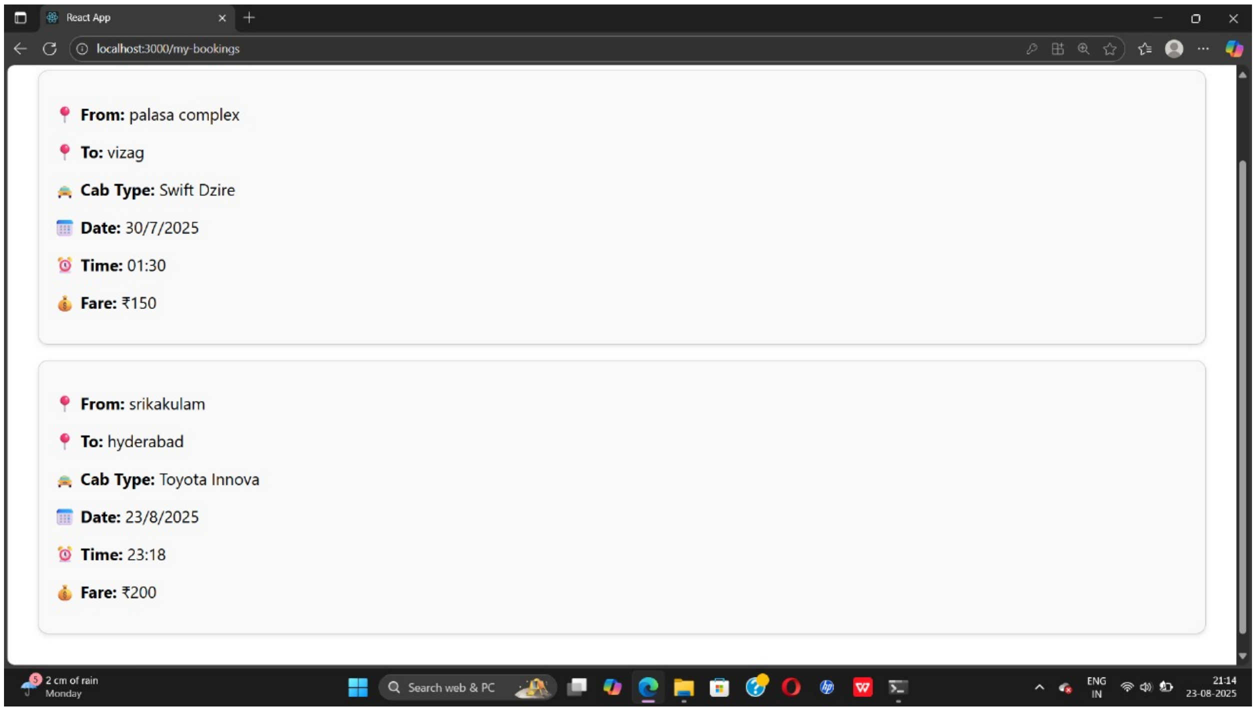
📍 **To:** vizag

🚕 **Cab Type:** Swift Dzire

📅 **Date:** 30/7/2025

⏰ **Time:** 01:30

💰 **Fare:** ₹150



React App

localhost:3000/my-bookings

📍 **From:** palasa complex

📍 **To:** vizag

🚕 **Cab Type:** Swift Dzire

📅 **Date:** 30/7/2025

⏰ **Time:** 01:30

💰 **Fare:** ₹150

📍 **From:** srikakulam

📍 **To:** hyderabad

🚕 **Cab Type:** Toyota Innova

📅 **Date:** 23/8/2025

⏰ **Time:** 23:18

💰 **Fare:** ₹200

Conclusion:

The cab booking app provides a seamless and efficient solution for booking rides anytime, anywhere.
With features like real-time tracking, fare estimation, secure payments, and driver ratings, it enhances both user and driver experiences.

The app reduces waiting time and improves ride management, offering convenience and reliability.
It promotes safer travel through verified driver profiles and GPS tracking.
Designed with scalability in mind, it can be easily adapted to growing user demands and new technologies.
The app contributes to modernizing urban transport and supports a more connected and mobile lifestyle.