

## **Neo Cars - Rental Booking Application**

### **Objective:**

Neo Cars is an application to be built as a product that can help customers to choose a Rental car.

### **Users of the System:**

1. Super Admin
2. Admin
3. Customer

### **Functional Requirements:**

- Build an application that customers can book Cars in company.
- The application should have signup, login, profile, dashboard page for user and login,signup,profile , dashboard, add Car page for the admin.
- This application should have a provision to maintain a database for customer information,Admin information,booking information and Car information.
- Also, an integrated platform required for customers, admin and super admin.
- Administration module to include options for adding / modifying / removing the existing Car(s) and customer management.
- **There should be a maximum of 4 people per car.**

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

- ☐ Filters for products like Low to High or showcasing Companies and Cars based on the customer's price range, specific Company etc.
- ☐ Email integration for intimating new personalized offers to customers.
- ☐ Multi-factor authentication for the sign-in process
- ☐ Payment Gateway

### **Output/ Post Condition:**

- ☐ Records Persisted in Success & Failure Collections
- ☐ Standalone application / Deployed in an app Container

### **Non-Functional Requirements:**

<b>Security</b>	<ul style="list-style-type: none"><li>• App Platform –UserName/Password-Based Credentials</li><li>• Sensitive data has to be categorized and stored in a secure manner</li><li>• Secure connection for transmission of any data</li></ul>
<b>Performance</b>	<ul style="list-style-type: none"><li>• Peak Load Performance (during Festival days, National holidays etc)</li><li>• eCommerce -&lt; 3 Sec</li></ul>

	<ul style="list-style-type: none"> <li>• Admin application &lt; 2 Sec</li> <li>• Non Peak Load Performance</li> <li>• eCommerce &lt; 2 Sec</li> <li>• Admin Application &lt; 2 Sec</li> </ul>
<b>Availability</b>	<ul style="list-style-type: none"> <li>• 99.99 % Availability</li> </ul>
<b>Standard Features</b>	<ul style="list-style-type: none"> <li>• Scalability</li> <li>• Maintainability</li> <li>• Usability</li> <li>• Availability</li> <li>• Failover</li> </ul>
<b>Logging &amp; Auditing</b>	<ul style="list-style-type: none"> <li>• The system should support logging(app/web/DB) &amp; auditing at all levels</li> </ul>
<b>Monitoring</b>	<ul style="list-style-type: none"> <li>• Should be able to monitor via as-is enterprise monitoring tools</li> </ul>
<b>Cloud</b>	<ul style="list-style-type: none"> <li>• The Solution should be made Cloud-ready and should have a minimum impact when moving away to Cloud infrastructure</li> </ul>
<b>Browser Compatible</b>	<ul style="list-style-type: none"> <li>• IE 7+</li> <li>• Mozilla Firefox Latest – 15</li> <li>• Google Chrome Latest – 20</li> <li>• Mobile Ready</li> </ul>

## Technology Stack

Front End	React 16+ Google Material Design Bootstrap / Bulma
Server Side	Spring Boot Spring Web (Rest Controller) Spring Security Spring AOP Spring Hibernate
Core Platform	OpenJDK 11
Database	MySQL or H2

## Platform Prerequisites (Do's and Don'ts):

1. As soon as the project mode is opened and set up, navigate to the settings of the Visual Studio Code by clicking on **File -> preferences -> Settings** or ( **Ctrl + ,** ).

- a. Click on the settings icon located as the first icon on the right-hand side to open the **settings.json** file.

- b. paste the following code inside it.

```
{ "settings": { "files.exclude": { "**/node_modules": true } } }
```

- c. Then proceed with running the react project.

**Note: The above step has to be repeated each time, the project mode is opened up.**

2. The React app should run in port 8081. Do not run the react app in the port: 3000 or port: 4200.
3. Spring boot app should run in port 8080.

**Key points to remember:**

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. The id provided should be defined strictly with the following name **data-test-id="<your id>"**. Example : **<input data-test-id="username" />**. The naming convention has to be followed for all the id's mentioned.
- 3.
4. Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.

5. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
6. Adhere strictly to the endpoints given below.

### **Application assumptions:**

1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by using Protected Routes from the react router package. For example, if the user enters as <http://localhost:3000/signup> or <http://localhost:3000/home> the page should not navigate to the corresponding page instead it should redirect to the login page.
3. Unless logged into the system, the user cannot navigate to any other pages.
4. Logging out must again redirect to the login page.
5. To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
6. Use admin/admin as the username and password to navigate to the admin dashboard.
1. .

### **Validations:**

1. Basic email validation should be performed.
2. Basic mobile validation should be performed.

### **Project Tasks:**

### **API Endpoints:**

#### **Admin Side:**

Action	URL	Method	Response
Admin Login	/admin/login	POST-Sends email ID and password	Return True/False
Admin SignUp	/admin/signup	POST-Sends Admin Model data	Admin added
Admin Dashboard	/admin/dashboard	GET	Return All the Cars in the company

Admin Add Car	/admin/addCar	POST-Sends Car Data	Car Added
---------------	---------------	---------------------	-----------

Admin Edit Car	/admin/editCar	POST-Sends Car Data	Car Edited
Admin Delete Car	/admin/deleteCar	POST-Sends Car ID	Car Deleted
Admin Profile	/admin/profile	POST-Sends Admin ID	Return Admin Profile Details
Edit Profile	/admin/editProfile	GET-Sends Admin ID	Return Admin Profile Details
Edit Profile	/admin/editProfile	POST-Sends Admin Model data	Edits Admin Profile

### User Side:

Action	URL	Method	Response
User Login	/user/login	POST-Sends email ID and password	Return True/False
Admin SignUp	/user/signup	POST-Sends User Model data	User added
User Dashboard	/user/dashboard	GET	Return all the Companies available
Displaying Hotel Cars	/user/Cars	POST - Sends Company name and admin ID of that company	Return all the Cars of the company
Car Details	/user/CarDetails	POST-Sends Car ID	Return Car details
Booked Cars	/user/bookings	POST-Sends User ID	Return user bookings

### Super Admin:

Action	URL	Method	Response
Super Admin Login	/super/login	POST-Sends email ID and password	Return True/False
Delete an admin	/super/deleteAdmin	POST-Sends admin email ID	Delete an admin
Delete an User	/super/deleteUser	POST-Sends user email ID	Delete a user

## **Frontend:**

### **Customer:**

1. Signup: Design a signup page component where the new customer has options to sign up by providing their basic details.
  - a. Ids:
    - i. signupBox
    - ii. email
    - iii. password
    - iv. userrole
    - v. username
    - vi. age
    - vii. submitButton
    - viii. loginLink
  - b. API endpoint Url: <http://localhost:3000/signup>
  - c. Output screenshot:

Diagram illustrating the structure of a SIGN UP form component with associated IDs:

- id = "signupBox" (points to the entire form container)
- id = "email" (points to the "Enter Email" input field)
- id = "password" (points to the "Enter Password" input field)
- id = "mobilenumber" (points to the "Enter Mobile Number" input field)
- id = "userrole" (points to the "User" dropdown menu)
- id = "username" (points to the "Enter Username" input field)
- id = "age" (points to the "Enter Age" input field)
- id = "submitButton" (points to the blue "Submit" button)
- id = "loginLink" (points to the "Click Here" link text)

2. Login: Design a login page component where the existing customer can log in using the registered email id and password.

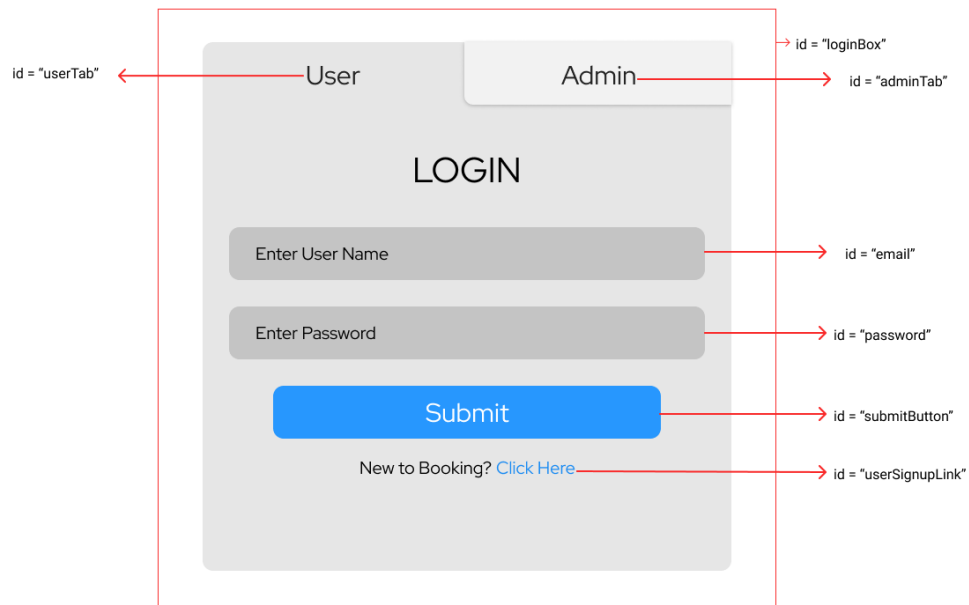
a. Ids:

- i. loginBox
- ii. userTab
- iii. email
- iv. password
- v. submitButton
- vi. userSignupLink

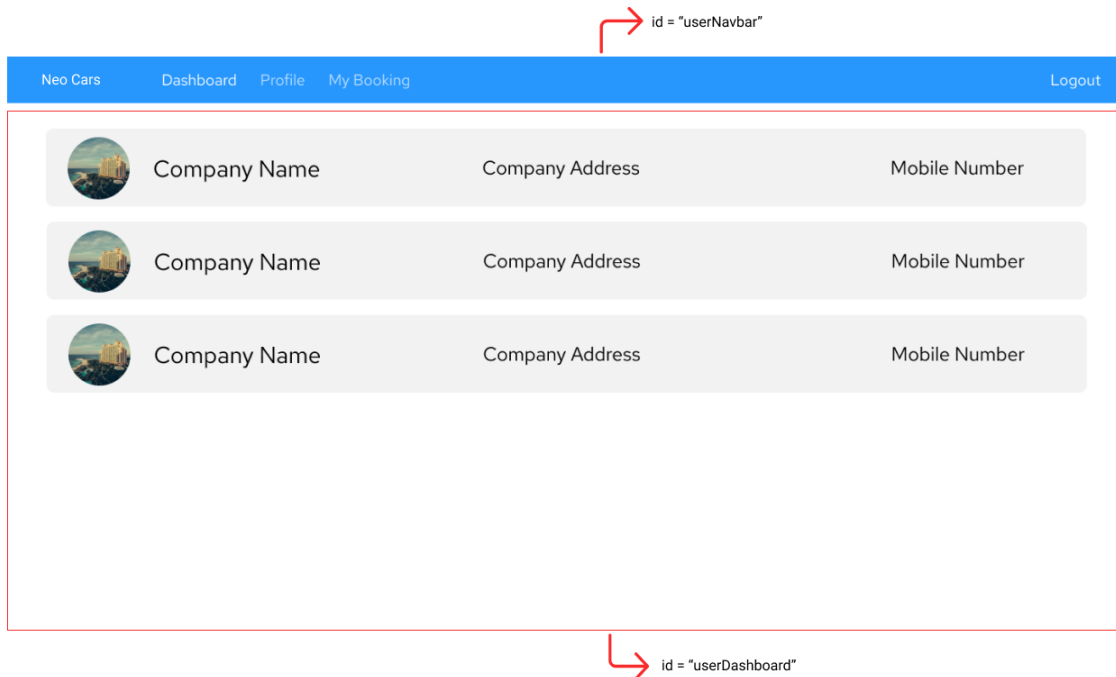
b. API endpoint Url: <http://localhost:3000/login>

c. Output screenshot:

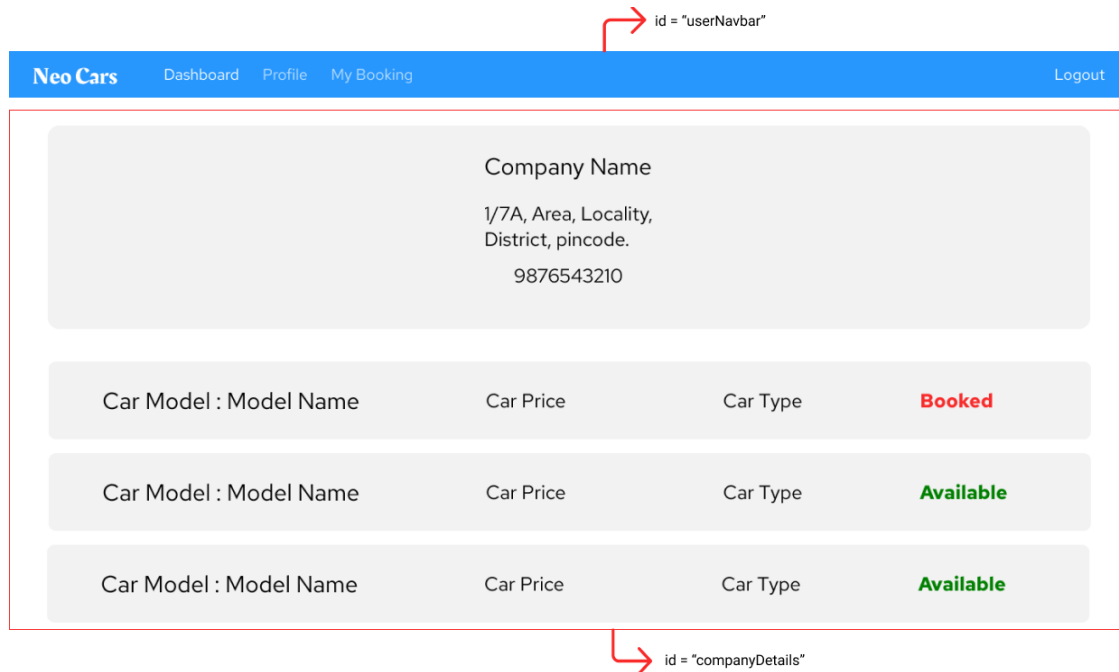




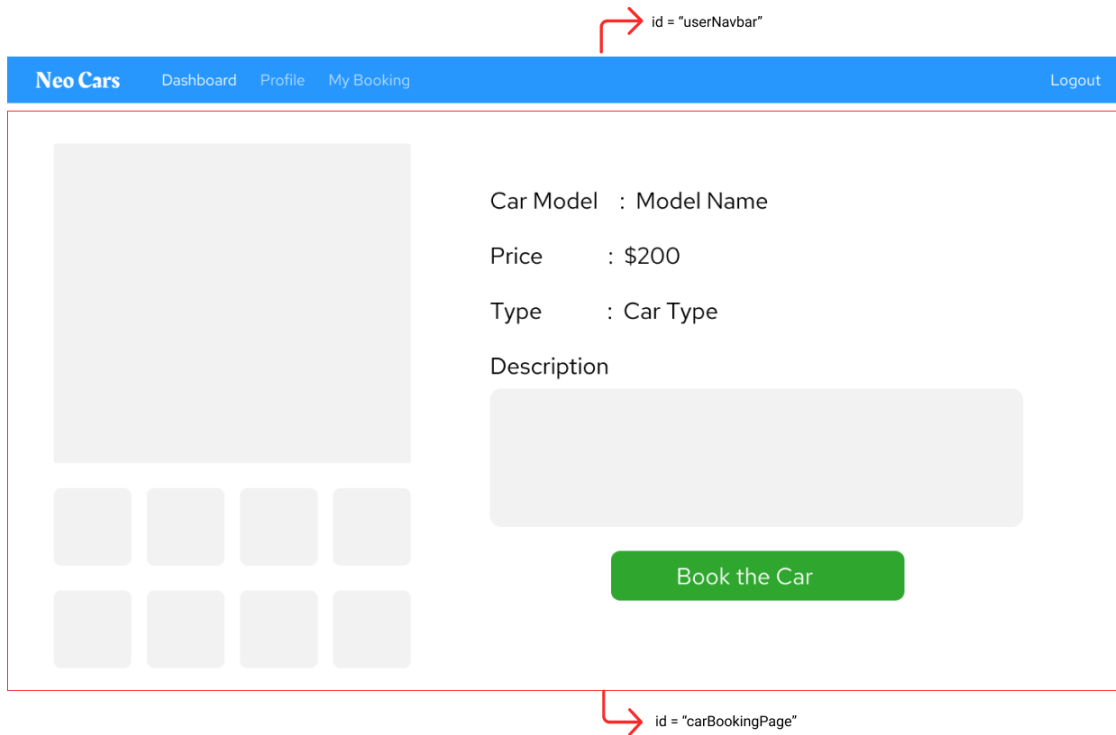
3. Dashboard / Home: Design a dashboard page component which provides a list of hotels available where the user can book a particular Car..
  - a. Ids:
    - i. `userNavbar`
    - ii. `userDashboard`
  - b. API endpoint Url: <http://localhost:3000/home>
  - c. Screenshot



4. Hotel Details : Design a Hotel Details component which lists the total number of Cars available along with the individual status of each Car ( available / Booked ).
  - a. Ids
    - i. userNavbar
    - ii. companyDetails
  - b. API endpoint URL : <http://localhost:3000/hotelDetail/{id}>
  - c. Screenshot



5. Bookings : Design a Booking Page that displays the details of the Car that the user is currently booking.
  - a. Ids
    - i. userNavbar
    - ii. CarBookingPage
  - b. API endpoint URL : <http://localhost:3000/booking/{id}>
  - c. Screenshot

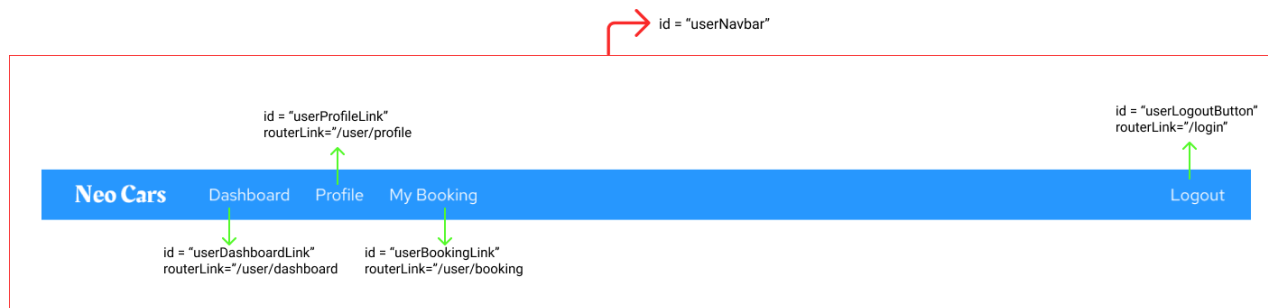


6. User Navigation : Design a Booking page component where the user can navigate to their profile and bookings page.

a. Ids

- i. userNavbar
- ii. userProfileLink
- iii. userDashboardLink
- iv. userBookingLink
- v. userLogoutButton

b. Screenshot



7. User Profile Edit : Design an Profile Edit Page where the user can edit the details of him/herself.

a. Ids

- i. userNavbar
- ii. editProfileBox
- iii. username
- iv. email
- v. password
- vi. userAge
- vii. mobilenummer
- viii. editProfileButton

b. API endpoint URL : <http://localhost:3000/editProfile/{id}>

c. Screenshot:

Neo Cars

DashboardProfileMy Booking

Logout

< back

Name

Jennifer Aniston

Email

rachelgreen@iamneo.ai

Password

JoeyTribbiani

Age

27

Mobile Number

1234567890

Save Changes

id = "editProfileBox"

id = "username"

id = "email"

id = "password"

id = "userAge"

id = "mobilenumber"

id = "editProfileButton"

8. User Bookings : Design a User Bookings Page that allows the user to view the bookings that the user has made till date.

a. Ids:

- i. userNavbar
- ii. userBookingBody

b. API endpoint URL : <http://localhost:3000/bookings/{id}>

c. Screenshot:

Neo Cars	Dashboard	Profile	My Booking	Logout
----------	-----------	---------	------------	--------

Company Name	Car Model	Rent	Days	Total Price
Company Name	Model Name	Rent	Days	Total Price
Company Name	Model Name	Rent	Days	Total Price
Company Name	Model Name	Rent	Days	Total Price
Company Name	Model Name	Rent	Days	Total Price
Company Name	Model Name	Rent	Days	Total Price

### **Admin:**

9. Admin Signup : Design a Signup page that registers a particular user with the role of Admin.

a. Ids

- i. signupbox
- ii. email
- iii. password
- iv. mobilenumber
- v. userrole
- vi. adminname
- vii. companyname
- viii. companyimageURL
- ix. companyAddress
- x. submitButton
- xi. adminLoginLink

b. API endpoint Url : <http://localhost:3000/admin/signup>

c. Screenshot

The image shows a 'SIGN UP' form with the following elements and their corresponding IDs:

- id = "signupBox"**: Points to the entire form container.
- id = "email"**: Points to the 'Enter Email' input field.
- id = "password"**: Points to the 'Enter Password' input field.
- id = "mobilenumber"**: Points to the 'Enter Mobile Number' input field.
- id = "userrole"**: Points to the 'Admin' dropdown menu.
- id = "adminname"**: Points to the 'Enter Seller Name' input field.
- id = "companyname"**: Points to the 'Enter Company Name' input field.
- id = "companyimageURL"**: Points to the 'Enter Company Image Url' input field.
- id = "companyAddress"**: Points to the 'Enter Company Address' input field.
- id = "submitButton"**: Points to the blue 'Submit' button.
- id = "adminLoginLink"**: Points to the 'Click Here' link in the 'Go to Login Click Here' text.

10. Admin Profile: Design a Admin Profile page that displays the details of the admin currently logged in.

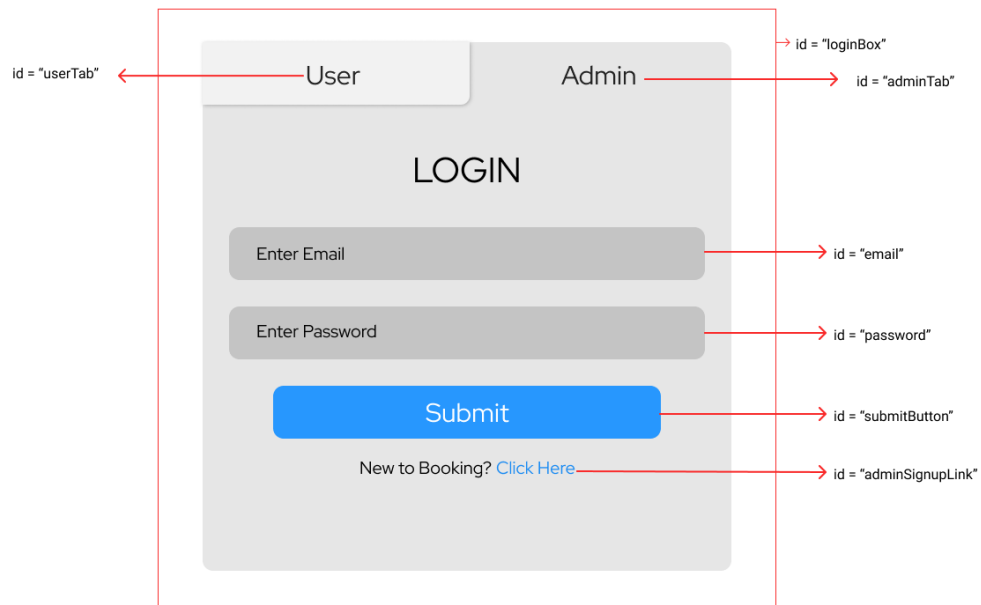
a. Ids

- i. adminNavbar
- ii. adminProfile
- iii. editAdminProfile

b. API endpoint URL: <http://localhost:3000/admin/profile>

c. Screenshots





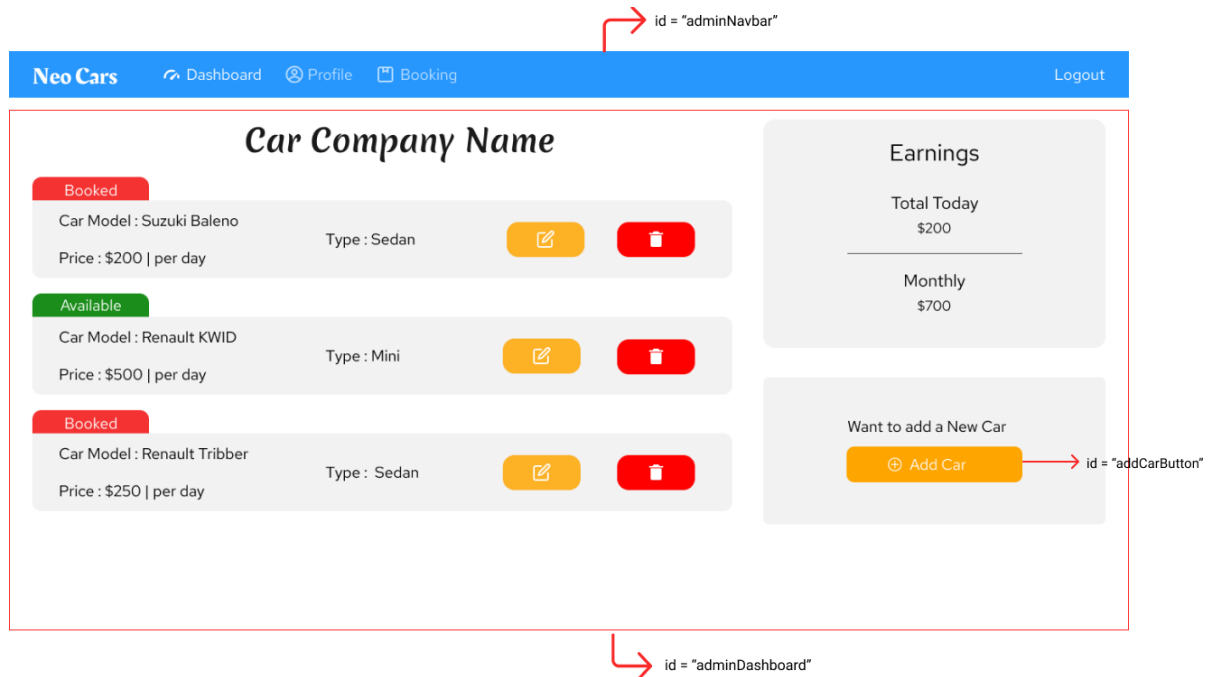
11. Admin Dashboard: Design a dashboard page where the list of products is displayed on the admin side.

a. Ids

- i. adminNavbar
- ii. addCarButton
- iii. adminDashboard

b. API endpoint Url: <http://localhost:3000/admin/dashboard>

c. Screenshot



12. Admin Navigation: Design an Admin navigation component that can navigate to the Dashboard, profile and bookings done by the users

a. Ids:

- i. adminNavbar
- ii. adminProfileLink
- iii. adminDashboardLink
- iv. adminBookingLink
- v. adminLogoutButton

b. Screenshot:



13. Admin Add Car: Design an Add Car component in which the admin can add new Cars to the inventory.

a. Ids:

- i. adminNavbar
- ii. addCarBox
- iii. CarNo
- iv. CarPrice
- v. CarType
- vi. addCarButton

b. API endpoint Url: <http://localhost:3000/admin/addCar>

c. Screenshot

< back

Car Model

Car Rent Price

Car Type

Add Car

id = "adminNavbar"

id = "addCarBox"

id = "carModel"

id = "carPrice"

id = "carType"

id = "addCarButton"

14. Admin Edit Car: Design an Edit Car component in which the admin can edit details of an already existing Car in the inventory.

a. Ids:

- i. adminNavbar
- ii. editCarBox
- iii. CarNo
- iv. CarPrice
- v. CarType
- vi. editCarButton

b. API endpoint Url: <http://localhost:3000/admin/editCar>

c. Screenshot

The screenshot shows a mobile application interface for editing car details. It features a blue header bar with the 'Neo Cars' logo and navigation links. Below the header is a light gray card containing a 'back' button, three text input fields for 'Car Model', 'Car Rent Price', and 'Car Type', and a blue 'Save Change' button. Red arrows point from text labels to specific UI elements: 'id = "adminNavbar"' points to the header bar, 'id = "editCarBox"' points to the gray card, 'id = "carModel"' points to the 'Car Model' input field, 'id = "carPrice"' points to the 'Car Rent Price' input field, 'id = "carType"' points to the 'Car Type' input field, and 'id = "editCarButton"' points to the 'Save Change' button.

15. Admin Bookings : Design a Admin Bookings component where the admin can see the booking details that all the users have made till present date.

a. Ids

- i. adminNavbar
- ii. adminBookings
- iii. bookingList

b. API endpoint URL : <http://localhost:3000/admin/bookings>

c. Screenshot:

id = "adminNavbar"

Booking ID	User ID	Car Model	Rent	Days	Total Price
Booking ID	User ID	Car Model	Rent	Days	Total Price
Booking ID	User ID	Car Model	Rent	Days	Total Price
Booking ID	User ID	Car Model	Rent	Days	Total Price
Booking ID	User ID	Car Model	Rent	Days	Total Price
Booking ID	User ID	Car Model	Rent	Days	Total Price

id = "bookingList"

id = "adminBookings"

16. Admin Profile Edit : Design an Admin Profile Edit page where the admin can edit the details of himself.

a. Ids

- i. adminEditBox
- ii. adminName
- iii. adminEmail
- iv. adminMobileNumber
- v. adminPassword
- vi. hotelName
- vii. hotelAddress
- viii. profileEditButton

b. API endpoint URL : <http://localhost:3000/admin/editProfile>

c. Screenshot:

The screenshot shows a mobile application interface for editing a profile. It features a 'back' button at the top left. Below it are input fields for 'Name' (containing 'Seller 1'), 'Email' (containing 'seller01@iamneo.ai'), 'Mobile Number' (containing '9876543210'), 'Password' (containing 'iamneo'), 'Company Name' (containing 'Examly'), and 'Company Address' (containing '1/7A, Area, Locality, District, pincode'). A blue 'Save Changes' button is at the bottom. Red arrows point from each input field and the button to specific IDs listed on the right: 'adminEditBox' for the back button, 'adminName' for the Name field, 'adminEmail' for the Email field, 'adminMobilenumber' for the Mobile Number field, 'adminPassword' for the Password field, 'companyName' for the Company Name field, 'companyAddress' for the Company Address field, and 'profileEditButton' for the Save Changes button.

Field	ID
back	id = "adminEditBox"
Name	id = "adminName"
Email	id = "adminEmail"
Mobile Number	id = "adminMobilenumber"
Password	id = "adminPassword"
Company Name	id = "companyName"
Company Address	id = "companyAddress"
Save Changes	id = "profileEditButton"

### **Super Admin:**

17. Super Admin Login Page: Design a Login page through which the super user can log in.

a. Ids:

- i. superAdminLoginBox
- ii. email
- iii. password
- iv. submitButton

b. API endpoint URL : <http://localhost:3000/superadmin/login>

c. Screenshot:

id = "superAdminLoginBox"

Super Admin

Enter Super Admin Email

id = "email"

Enter Password

id = "password"

Submit

id = "submitButton"

18. Super Admin Users page : Design a Super-Admin users page where the super admin has authority to view and delete any user currently registered into the system.

a. Ids:

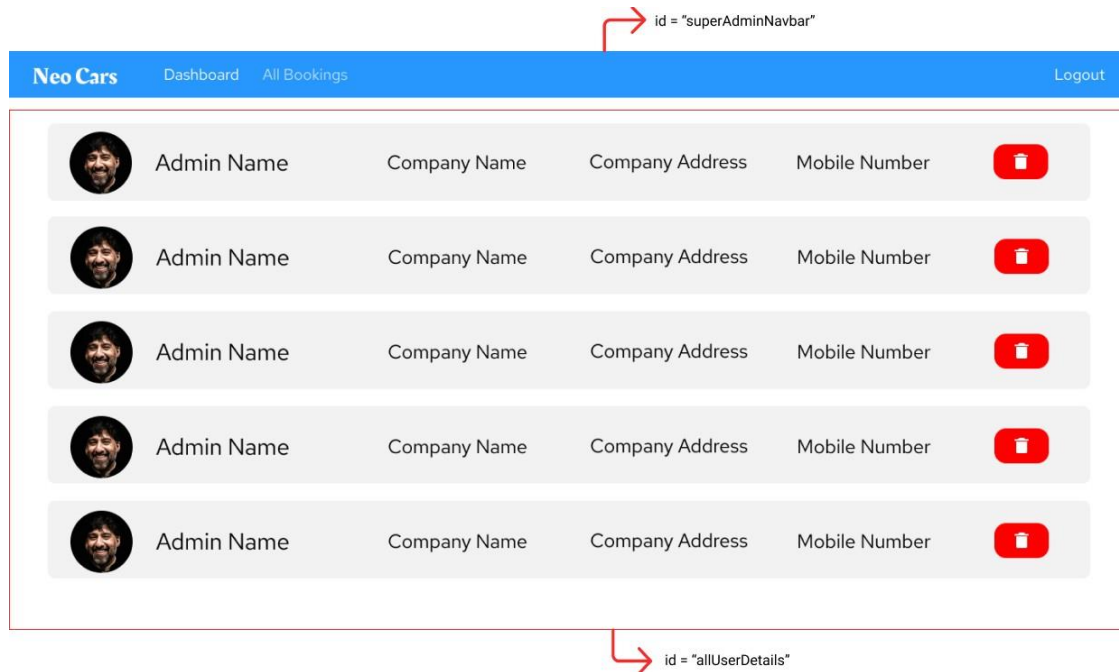
i. superAdminNavbar

ii. allUserDetails

b. API endpoint URL : <http://localhost:3000/superadmin/users>

c. Screenshot





19. Super Admin Booking page : Design a Super-Admin Admin component where the super admin has authority to view and delete any user currently registered into the system.

a. Ids:

i. superAdminNavbar

ii. allAdminDetails

b. API endpoint URL : <http://localhost:3000/superadmin/admin>

c. Screenshot

id = "superAdminNavbar"

Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price
Admin ID	Company Name	Car Model	Rent	Days	Total Price

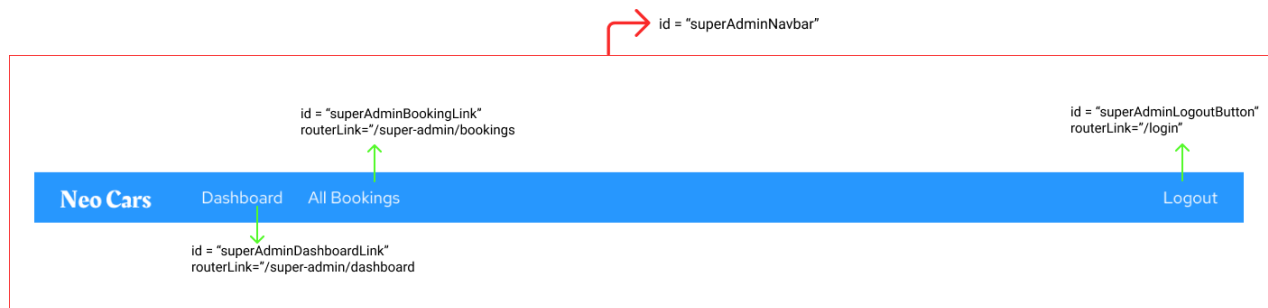
id = "allAdminDetails"

20. Super Admin Navigation : Design a Navigation page for the super user where he can navigate to the dashboard, all the bookings and logout.

a. Ids :

- i. superAdminNavbar
- ii. superAdminBookingLink
- iii. superAdminLogoutButton
- iv. superAdminDashboardLink

b. Screenshot:



## **Backend:**

### **Class and Method description:**

#### **Model Layer:**

1. UserModel: This class stores the user type (admin or the customer) and all user information.
  - a. Attributes:
    - i. email: String
    - ii. password: String
    - iii. username: String
    - iv. mobileNumber: String
    - v. age: int
    - vi. userRole: String
2. LoginModel: This class contains the email and password of the user.
  - a. Attributes:
    - i. email: String
    - ii. password: String

3. AdminModel: This class stores the details of the product.

a. Attributes:

- i. email:String
- ii. password:String
- iii. mobileNumber:String
- iv. sellerName:String
- v. userRole:String
- vi. hotelName:String
- vii. hotelImageURL:String
- viii. hotelAddress: String
- ix. earnings:int

4. CarModel: This class stores the cart items.

a. Attributes:

- i. CarID:String
- ii. CarNo:String
- iii. adminID:String
- iv. status:String
- v. price:String
- vi. type:String

### **Controller Layer:**

5. AuthController: This class control the user /admin signup and signin

a. Methods:

- i. isUserPresent(LoginModel data): This method helps to check whether the user present or not and check the email and password are correct and return the boolean value.
- ii. isAdminPresent(LoginModel data): This method helps to check whether the admin present or not and check the email and password are correct and return the boolean value.
- iii. saveUser(UserModel user): This method helps to save the user data in the database.
- iv. saveAdmin(UserModel user): This method helps to save the admin data in the database.

6. CarController: This class controls the save/edit/delete/view Cars.

a. Methods:

- i. saveCar(CarModel data): This method helps the admin to save new Cars in the database.
- ii. editCar(CarModel data): This method helps the admin to edit the details of the Cars and save it again in the database.
- iii. deleteCar(String Car ID): This method helps the admin to delete a Car from the hotel and as well as in the database.
- iv. getCars(String Admin\_Email\_ID): This method helps the admin to get all the Cars in the hotel.
- v. bookCar(String Car\_ID): This method helps the user to book the Car by changing the Car status and add the record in the bookings table.

7. AdminController: This class controls the edit/view admin details.

a. Methods:

- i. editAdmin(AdminModel data): This method helps the admin to edit the profile details and save it in the database.
- ii. getProfile(String Admin\_Email\_ID): This method helps admin to get the profile details.

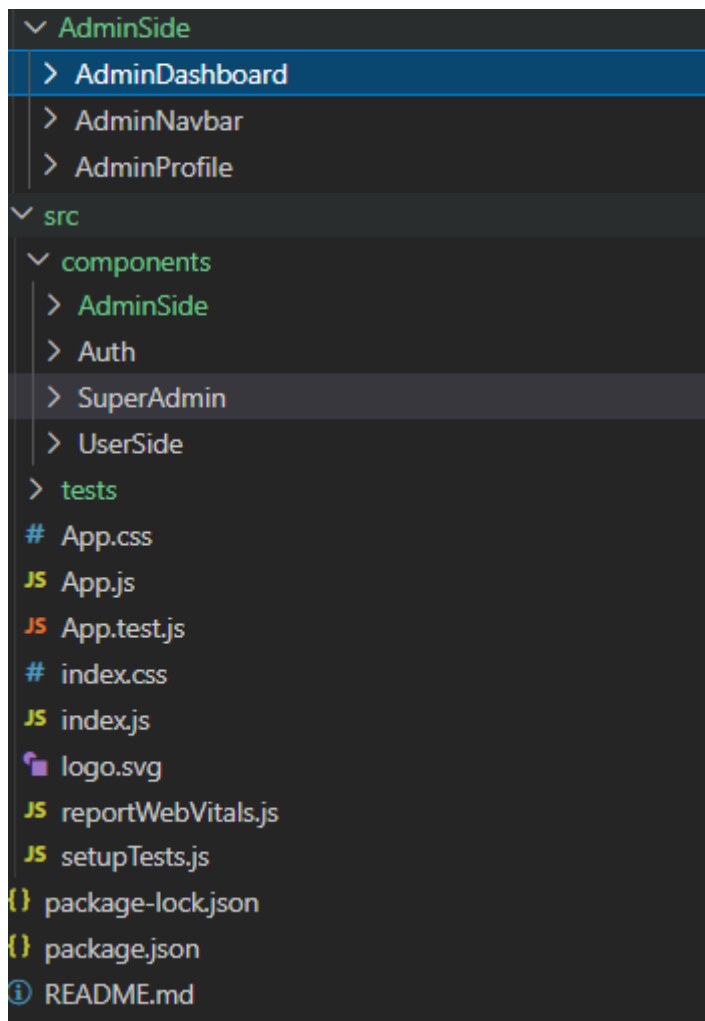
8. UserController: This class helps to get the Cars.

a. Methods:

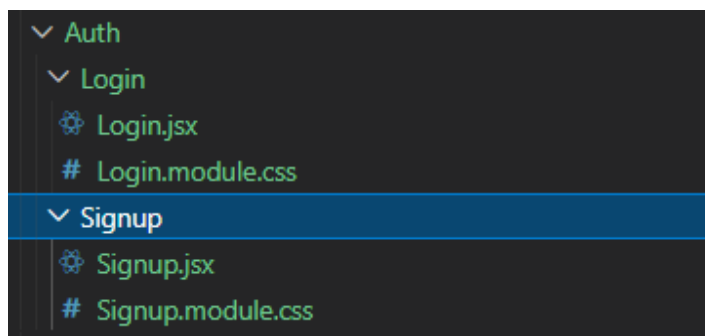
- i. userBookings(String User\_ID): This method helps users to get all the booking details done by them.

## **React Folder Structure:**

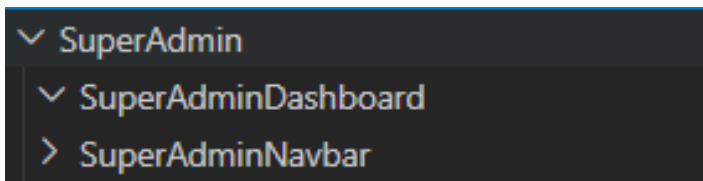
## **Admin Side:**



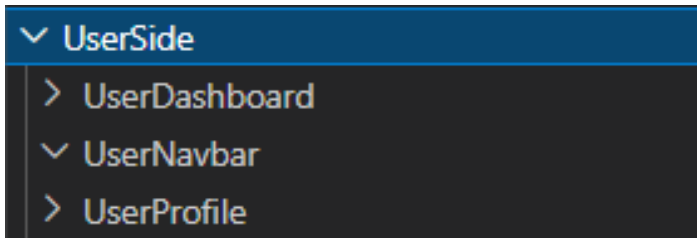
### Auth Side:



### Super Admin Side:



### User Side:



### React component Structure:

- **src ( folder )**
  - **components ( folder )**
    - **Navbar ( folder )**
      - **Navbar.jsx ( jsx file )**
      - **Navbar.module.css ( css file )**

### **NOTE:**

You should create the above folder structure mandatorily to pass the test cases and you can also create extra components if you need.

## **Workflow Prototypes:**

### **Admin**

<https://www.figma.com/proto/65gAU93Fw4R351v2NnzHIC/Neo-Car-Rental-Admin-Flow?node-id=1%3A482&viewport=587%2C493%2C0.1219395324587822&scaling=scale-down>

### **User**

<https://www.figma.com/proto/U5BYMfRAOfRxZkhOgO04ie/Neo-Car-Rental-User-Flow?node-id=1%3A33&viewport=632%2C540%2C0.07334188371896744&scaling=scale-down>

### **Super Admin**

<https://www.figma.com/proto/lxLNSqPprebmhIC3pdaxBy/Neo-Car-Rental-Super-Admin-Flow?node-id=1%3A4&viewport=369%2C302%2C0.04095541685819626&scaling=scale-down>