

Exotic Drive Dealership

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces Spring 2025

Team Members:

Casper Run - casper@iastate.edu

Ayan Asim - ayan7180@iastate.edu

May 11, 2025

1. Introduction

Overview of the problem, motivation, users, and goals. State whether the project is original or inspired.

Ayan and Casper collaborated to create a dealership website for luxury/exotic vehicles. We both like luxury performance vehicles, and we decided that this would be a good project that we would both enjoy while also learning the ins and outs of developing a website. The main goal for this website is to mimic an actual luxury car dealership website where the user can have an account, make announcements, and compare the prices and specs of vehicles with each other.

2. Project Description

Explain major features, user flow, CRUD operations and entities affected.

- **Login/Sign up feature**
 - The user will click on the signup tab on the taskbar and will be led to make an account. If the user already has an account, the user can skip to login from a button.
 - Once on the login page, the user has the option to log in or delete their account.
 - Deleting an account will ask the user to enter a name and password to delete an account. If the user entered the wrong password and name, it will not delete the account. It is the same thing for Login, except that it will allow the user to access the website if logged in correctly.
- **Appointments feature**
 - The user will click on the service page to schedule a servicing appointment. The user will enter their info, such as a phone number, email, name, time, date, etc.
 - The user will have the option to add an appointment and view all appointments in the same card.
 - The add appointment button adds an appointment to the database
 - The View appointment button takes you to another page where all the appointments in the database are visible in separate cards.
 - In each card, there is an option to update the appointment and delete an appointment.
 - Update appointment lets you update any specifications that you listed when you added an appointment.
 - Delete the appointment that was clicked on from the database.
- **Catalog feature**
 - The user is able to click on the catalog page and make a GET request to the database, pulling the car catalog object.
 - The catalog displays all cars in a card object, showing basic vehicle information and an image of the vehicle. Each card also has a view car button.
 - The catalog page also features a search bar that is able to display specific cars using keywords.
 - The view car button takes the user to a separate page where another GET request is made to pull and display extra information about the vehicle and several images.
 - Next to the search bar there are two buttons one that allows the user to add car and the other to delete car
 - The add car button takes the user to a separate page to input vehicle information, which is then converted to JSON and sent to the backend to store using a POST

- request. After the operation is complete, a status alert is displayed for the user.
- The delete car button takes the user to a separate page that displays the car catalog in a similar manner to the main catalog page, but each car card has a delete button. When a user clicks the delete button, a DELETE request is sent to the database, which then subsequently deletes the vehicle and displays a status message.

3. File and Folder Architecture

Describe structure of frontend/, backend/, Documents/. Include a folder tree diagram.

```
├── .git
├── .gitignore
├── Backend
│   ├── node_modules
│   ├── package-lock.json
│   ├── package.json
│   └── server.js
├── Documents
├── Frontend
│   ├── initial_app
│   │   ├── .gitignore
│   │   ├── README.md
│   │   ├── eslint.config.js
│   │   ├── index.html
│   │   ├── node_modules
│   │   ├── package-lock.json
│   │   ├── package.json
│   │   ├── public
│   │   │   └── vehicleImages
│   │   └── src
│   │       └── App.css
```

```
|      | | — App.jsx
|      | | — Appointments
|      | | — Catalog
|      | | — Login/Sign-up
|      | | — aboutPage.jsx
|      | | — assets
|      | | — brandsPage.jsx
|      | | — carContext.jsx
|      | | — data
|      | | — footer.jsx
|      | | — homePage.jsx
|      | | — index.css
|      | | — main.jsx
|      | | — navBar.jsx
|      | | — tailwind.config.js
|      | — vite.config.js
| — JSONs
```

There are four main folders in our file structure are Documents, Backend, Frontend, and JSONs. Document folder stores our proposal, mini-assignment documentation, and final report. The Backend stores the necessary node packages for the database and API as well as the [server.js](#) file. Frontend has all the image assets, component jsx files, and all the required configuration files. The Frontend components are put into folders based upon features and all single page features are left in the src folder. The JSONs folder stores all the JSON files that are imported into MongoDB for our website to interact with using our backend API.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

All API requests for the backend are declared in the [server.js](#) file. In the [server.js](#) file all CRUD operations are specified with endpoint links like add, delete, or update for POST, DELETE, and PUT respectively. For example, a car catalog featuring a CRUD operation would look like this “/addCar”. Most of these backend requests have success and fail messages that are displayed to the user whenever the request reaches the backend. Our login system implements cookies to store user

information and is connected directly to the appointments feature. Whenever a user logs into the system only they are able to see their appointments.

Each frontend feature and component has a function that is able to call these API requests on the backend to perform CRUD operations on our Mongo database.

4.2. React Component Structure

For illustrating our component hierarchy we provided an image of our App.jsx file. Our project has many props and to keep our overview concise we will highlight only the important ones.

```
function App() {  
  const [user, setUser] = useState(null);  
  
  return (  
    <>  
      <Navbar />  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/brands" element={<Brands />} />  
        <Route path="/catalog" element={<Catalog />} />  
        <Route path="/addCar" element={<AddCar />} />  
        <Route path="/service" element={<AppointmentsData />} />  
        <Route path="/about" element={<About />} />  
        <Route path="/viewCar" element={<ViewCar />} />  
        <Route path="/deleteCar" element={<DeleteCar />} />  
        <Route path="/signup" element={<SignUpPage setUser={setUser} />} />  
        <Route path="/login" element={<LoginPage setUser={setUser} />} />  
        <Route path="/viewAppointments" element={<ViewAppointments />} />  
        <Route path="/updateAppointment" element={<UpdateAppointment />} />  
        <Route path="/deleteAppointment" element={<DeleteAppointment />} />  
        <Route path="/deleteAccount" element={<DeleteAccount />} />  
      </Routes>  
      <Footer />  
    </>  
  );  
}
```

Some components that are apart of every page are the Navbar and the Footer. The main components are under the routes flag and each component has a route path associated with it. "/" is considered the home page and other standalone pages like brands.

The catalog has several components associated with it that correspond to the CRUD operations for the feature. The main state that is used in the catalog components is [catalog, setCatalog], which handles the car catalog listings and allows for them to be dynamically added and deleted.

The Login/Sign-up have their respective components in the route paths. An important state in this feature is the [user, setUser], which manages that user that is currently logged in and attaches to the appointments feature as well as the catalog features.

The appointments feature similar to the catalog feature has several components that correspond to its CRUD operations. Some important states in this feature are [appointmentId, setAppointmentId] and [form, setForm]. Appointment ID assists with the deletion feature by uniquely identifying appointment listings in the database. The form state is how the user is able to input appointment information, send it to the database, and edit each individual appointment state.

4.3. Database Interaction

The database is used for storing information such as appointments, account information, and the vehicle catalog. For account information and appointments specifically, all the information is combined and stored in one JSON. This is because logging in and making appointments are dependent on each other. If you are not signed in, you cannot make an appointment.

4.4. Code Snippets

Code origin: appointments.jsx: If the user tries to access appointments without being logged in, the user will be prompted to log in and will be forwarded to the login page.

```
const handleViewAll = async () => {
  try {
    const res = await fetch("http://localhost:8080/app", {
      credentials: "include",
    });
    if (res.status === 401) {
      alert("Please log in to view appointments.");
      return navigate("/login");
    }
    navigate("/viewAppointments");
  } catch (err) {
    console.error(err);
    alert("Unable to load appointments.");
  }
};
```

Code origin: carCatalog.jsx & carContext.jsx: In carCatalog.jsx the view and catalog is set using props. The admin flag is also checked to allow for the admin buttons to pop up. For enabling the view car feature, I used a context jsx file to store the state and it allows for the state and props to be used in several jsx files rather than its host file.

```
const Catalog = ({ user }) => {
  const [catalog, setCatalog] = useState([]);
  const [query, setQuery] = useState("");
  const [admin, setAdmin] = useState(false);
  const { setSelectedCar } = useCar();
  const navigate = useNavigate();

  useEffect(() => {
    if (user?.admin === "1") {
      setAdmin(true);
    } else {
      setAdmin(false);
    }
    fetchCatalog()
  }, [user]);

  const handleView = (car) => {
    setSelectedCar(car)
    navigate('/viewCar', {
      state: {
        model: car.model, driveTrain: car.driveTrain,
        mileage: car.mileage, price: car.price, type: car.type, imageURL: car.imageURL
      }
    })
  }
}
```

```
import React, { createContext, useContext, useState } from 'react';
💡
// Create context
const CarContext = createContext();

// Export the provider
export const CarProvider = ({ children }) => {
  const [selectedCar, setSelectedCar] = useState(null);

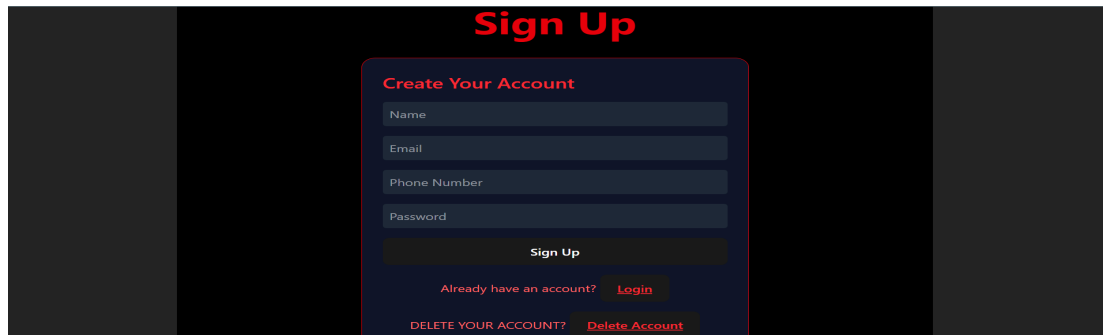
  return (
    <CarContext.Provider value={{ selectedCar, setSelectedCar }}>
      {children}
    </CarContext.Provider>
  );
};

// Custom hook for easy access
export const useCar = () => useContext(CarContext);
```

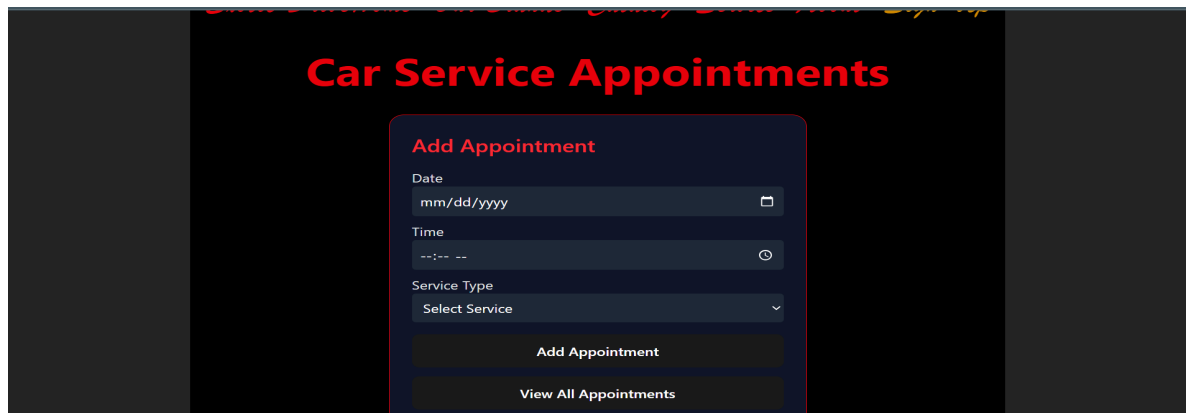
5. Web View Screenshots and Annotations

Signup/login feature (Ayan) - This feature allows the user to sign up and make an account. If the user already has an account, the user can click the login button and get to the login page where they

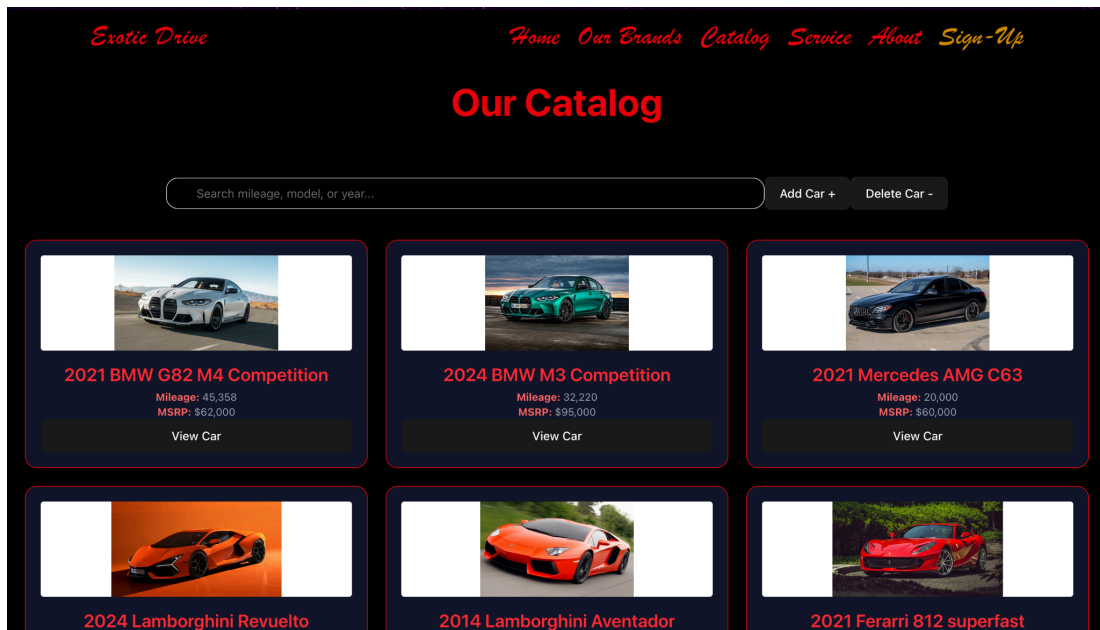
can enter their email and password. The user will then be able to gain more access to the website as they can now make an appointment.

A screenshot of a 'Sign Up' form. The title 'Sign Up' is in large red font at the top. Below it, 'Create Your Account' is in smaller red font. The form contains four input fields: 'Name', 'Email', 'Phone Number', and 'Password'. Below these fields is a 'Sign Up' button. At the bottom, there is a link 'Already have an account? Login' and a link 'DELETE YOUR ACCOUNT? Delete Account'.

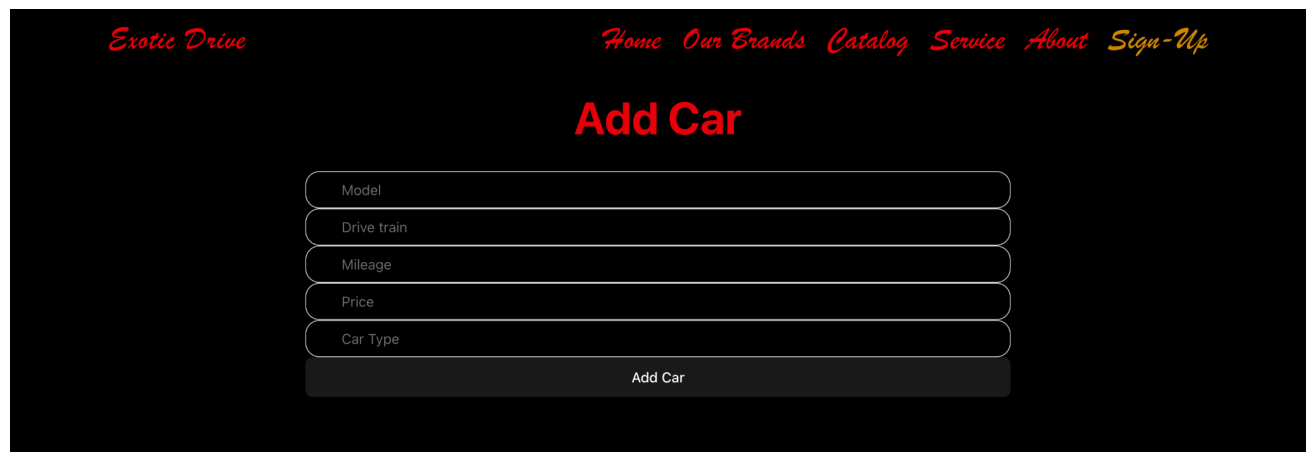
Appointments Feature (Ayan) - This feature allows the user to make an appointment as long as they are signed into their account. The user will enter the info on the card, and it will be saved in the database. Once you click “View All Appointments”, the user will be led to a page where all their appointments will be shown. The user can choose to delete those appointments or update them.

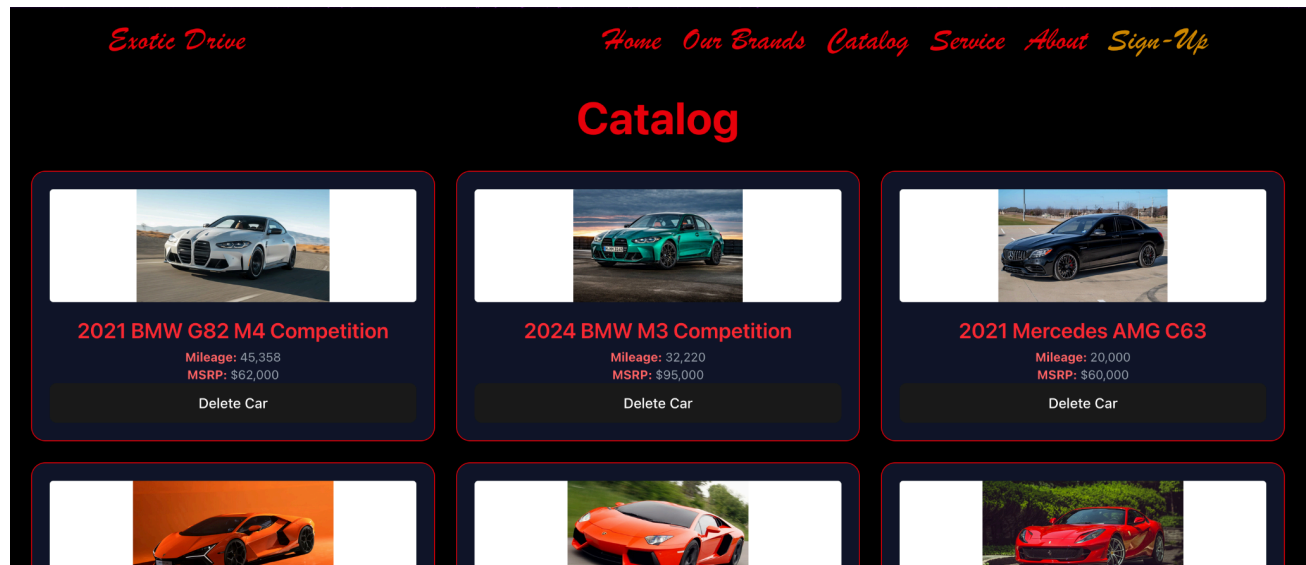
A screenshot of a 'Car Service Appointments' form. The title 'Car Service Appointments' is in large red font at the top. Below it, 'Add Appointment' is in smaller red font. The form contains four input fields: 'Date' (with a date picker icon), 'Time' (with a time picker icon), 'Service Type' (with a dropdown menu), and 'Add Appointment' button. Below the button is a 'View All Appointments' button.

Catalog View Feature (Casper) - This feature allows users to view the dealership's car catalog in a format that displays each car as a card and search for cars based on key. If a user would like more information about a car and more images they are able to click the view car button on a card.



Catalog Edit Feature (Casper) - This feature allows users to add and delete cars based upon if the user is considered an admin. The add function has a support page with input boxes that allow users to create information for a car listing in the database. Once the user clicks add car, it is added to the database and viewable in the catalog. The delete function displays the catalog in a separate page with a delete button attached to each car card. Once clicked the car is deleted from the database and unavailable on the catalog.





6. Installation and Setup Instructions

Step-by-step instructions to run the frontend and backend. Mention environment variables and DB setup.

- The server was made for this project in MongoDB
- database was made, with a collection in the database to have our data for accounts, appointments, and vehicle catalog in a JSON format.
- Build the frontend first with good Tailwind CSS styling, React components, and nice-looking cards to house our data, as well as buttons for navigating to different pages.
- Build the backend and connect with the MongoDB database. Write the frontend URL in app.use for CORS and “credentials: true” so the backend can reference it.
- Add a GET request for getting things from the database
- Add a POST request to add things to the database
- Add a PUT request to edit things that are already in the database
- Add a DELETE request to delete things from the database.
- Send a request from the frontend to the backend from the “handleSubmit” method in the frontend

7. Contribution Overview

A table or list mapping each feature to the responsible team member.

Ayan	Casper
Login/signup frontend and backend	Navbar/Footer
Appointments frontend and backend	Home Page
Brands page	Vehicle Catalog frontend and backend

About page	About page
------------	------------

8. Challenges Faced

List 2–3 significant development/debugging issues and how you addressed them.

- Appointments frontend/backend communication
 - Problem: The frontend and backend code were made for GET and POST requests for adding an appointment. The GET request was working, but adding an appointment was not working.
 - Sat with Jabir and Pranava for 4 ½ hours, troubleshooting the issue
 - The solution that we found is that I need to have “credentials: include” in my request from the frontend for it to actually add an appointment. I was also running the frontend on 5175 instead of 5173, where the backend is supposed to recognize the frontend.
- Debugging backend communication for the catalog
 - Had issues implementing the POST and DELETE backend functionality to add and delete cars. The POST issue was that I was unable to input characters into the input values, and the request wasn’t correctly formatted. The delete backend was not sending a successful status code, and the catalog wasn’t updating dynamically.
 - The POST issue I successfully fixed by adding the value attribute to the input tags and surrounding the input tags and button tag with the form tag.
 - The DELETE issue I fixed was by adding a successful status message to display when the delete goes through to the backend correctly.

9. Final Reflections

- Overall, this project was a good learning experience for learning React, MongoDB, [Node.js](#), and more. This project will help us gain a better understanding of the world of web development in the industry. However, it came with some challenges as we had spent many hours debugging in order for our website to work properly. We ended up solving these problems learned many things about React and Node from those experiences.