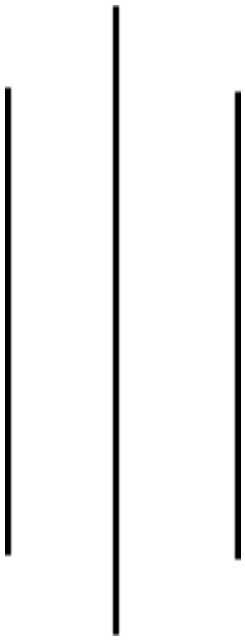
Car Dealership Project

Course: CSD 3103

Full Stack JavaScript



Group Members:

Ananta Poudel (C0913139)

Bibek Shrestha (C0905023)

Bikash Sapkota (C0911133)

Subash Pariyar (C0913543)

Instructor: Mallika Mallika

Date: 13th Dec 2024

Table of Contents

1. Introduction

1.1 Project Objectives

The main aim of this project is to design and implement a full web service intended for car dealerships to make the showcase, filtering, and management of vehicle inventories easier. This service will increase user engagement by introducing a modern, responsive, intuitive interface for casual browsers and serious buyers. The presentation of the cars is done in an organized 4x4 grid layout where one can get a quick overview of vital details regarding the make and model, price, mileage, and more. This way, detailed specifications, fine-quality images, and other helpful information on the selected car can be provided on its dedicated page.

Advanced filtering options are an integral part of the service to find vehicles according to the customer's preference on price range, kilometers driven, or other criteria. Feature pagination enhances this further, with only 16 vehicles per page for better navigation and an overloaded design.

On the technical side, the project leverages React.js for dynamic front-end development, Node.js for a scalable and efficient backend, and MongoDB for secure, flexible data storage. The system is designed with a RESTful approach that allows modularity and ensures easy integration with any third-party service in the future. The web service that will allow scalability and data integrity sets the foundation in this book to implement some add-ons such as user authentication, real-time inventory update capability, and analytics.

Ultimately, this project aims to bridge the gap between car dealerships and their customers through a digital platform that effectively combines usability, functionality, and modern technology. The system will be a tool worth depending on by individual buyers, staff at the dealership, and even prospective partners in managing and exploring car inventories.

1.2 Scope of the Project

This project's scope encompasses creating a robust and user-friendly web service that caters to the operational needs of car dealerships while enhancing the browsing experience for potential customers. This web service is designed with several key functionalities that are integral to achieving its purpose:

1. **Displaying a List of Cars with Abstract Information**:
   The application provides users with a well-structured and visually appealing interface where vehicles are displayed in a grid layout. Each vehicle card includes make, model, year, price, and a representative image. This abstract representation allows users to quickly scan through available options and identify vehicles of interest without being overwhelmed by excessive information.
2. **Filtering Cars by Price or Kilometers (KMs)**:
   To cater to diverse customer preferences, the system integrates advanced filtering capabilities. Users can refine their search based on price ranges or kilometers driven, narrowing down options to meet their specific criteria. This feature ensures efficiency and convenience, particularly for buyers with a clear budget or preference for vehicle usage.
3. **Viewing Detailed Information about a Selected Car**:
   Upon selecting a vehicle, users are redirected to a dedicated details page. This page offers comprehensive information, including specifications such as engine type, fuel efficiency, transmission, seating capacity, and additional features. Multiple high-quality images of the car are also displayed to provide a realistic and detailed view. This feature aids in informed decision-making and enhances user satisfaction.
4. **Efficiently Handling Data Storage and Retrieval Using MongoDB**:
   This project works with vehicle data using a NoSQL database called MongoDB, which is very efficient regarding scalability and flexibility. The schema is specially designed so that data retrieval and storage are efficiently managed with fast response times, even on big sets. Features such as unique VIN (Vehicle

Identification Number) fields maintain the integrity of the data by controlling data duplication.

5. **Seamless Integration of Front-End and Back-End Using RESTful Services**: The application follows a RESTful architecture, ensuring smooth communication between the front-end and back-end parts. The back-end, using Node.js and Express.js, provides APIs that perform data operations, while the React.js-based front-end consumes these APIs to dynamically present data. This modular design makes the system maintainable and scalable, thus fit for enhancements and integrations in the future.

Additional Features:

- **Pagination**: The pagination implemented in the web service allows users to view 16 vehicles at a time per page while preventing information overload. For better usability, large datasets are being chunked into smaller pieces.
- **Cross-Platform Accessibility**: The services are designed to work seamlessly with desktops, tablets, and mobile phones, increasing their reach to potential users of all kinds.

2. Design and Architecture

2.1 System Architecture

The system architecture for your project follows a **client-server model**. Here's a breakdown of each component:

1. Front-End (Client-Side):

➔ **Technology**: React.js is styled with Tailwind CSS.

➔ **Responsibilities**:

◆ Display the vehicle list to the user.

- ◆ Implement the vehicle filter functionality (by price or KMs).
- ◆ Use React components to structure the UI, making the interface dynamic and responsive.
- ◆ Pagination: Display 16 vehicles per page in a 4x4 grid layout.
- ◆ When selecting a vehicle, navigate to a detailed view showing complete vehicle information and an image.

2. Back-End (Server-Side):

➔ **Technology**: Node.js and Express.js.

➔ **Responsibilities**:

- ◆ Handle incoming requests from the front end (such as fetching the list of vehicles, applying filters, or retrieving detailed vehicle information).
- ◆ Implement RESTful API routes:
  - **GET /vehicles**: Retrieves a list of vehicles with optional filters for price and kilometers.
  - **GET /vehicle/:id**: Retrieves detailed information about a single vehicle based on its unique ID.
- ◆ Server-side logic for pagination to return a fixed number of vehicles per page (16).
- ◆ Handle any additional back-end functionalities, such as vehicle search or sorting.

3. Database:

➔ **Technology**: MongoDB (NoSQL database).

➔ **Responsibilities**:

- ◆ Store vehicle information, including fields like vehicle ID, name, model, price, kilometers, year, and image URL.
- ◆ Support efficient querying for filtered vehicle data (price, kilometers) and pagination.
- ◆ Use MongoDB's flexible schema to handle varying vehicle data.

Flow Overview:

1. **User Interaction**: The user interacts with the front end, filtering the list of vehicles by price or kilometers.
2. **API Request**: The React front-end sends an API request to the Node.js back-end.
3. **Back-End Processing**: The back-end queries the MongoDB database for the relevant data (filtered by user input) and returns the result.
4. **Rendering Data**: The React front-end displays the filtered list of vehicles (16 per page), and each vehicle has an image and brief information.
5. **Vehicle Detail View**: When a user selects a vehicle, the front end fetches additional details from the back end and displays the complete information on a new page.

2.2 Database Design

The database for the vehicle listing application uses MongoDB as the data storage solution. The system consists of a single collection, **Dealership**, which stores all the vehicle information. Below is a detailed explanation of the collection schema and its fields:

Collection: dealership

The **dealership** collection is designed to store the essential details of each vehicle, including attributes such as make, model, price, and images. Each document in this collection represents a single vehicle.

Schema Fields:

- **Name** (String): Name of the vehicle
- **model** (String): The specific model of the vehicle (e.g., Corolla, Mustang).
- **year** (Number): The manufacturing year of the vehicle (e.g., 2020).
- **color** (String): The color of the vehicle (e.g., Red, Blue).
- **km** (Number): The number of kilometers the vehicle has been driven.

- **VIN** (String, unique): The Vehicle Identification Number (VIN), a unique identifier for each vehicle.
- **price** (Number): The price of the vehicle.
- **images** (Array of Strings): An array containing up to four image URLs representing different views of the vehicle.

2.3 Front-End Design

This section describes the design and structure of the front end of your vehicle listing application.

- UI/UX Design:
    - Layout: Explain how the interface is laid out, including the vehicle listing page, filter options, pagination controls, and the details page for individual vehicles. Mention how the design ensures that users can easily browse through vehicles.
    - User Flow: Detail how the user navigates from the vehicle list to the vehicle details page, how they apply filters, and how the pagination system works. Ensure that the navigation is intuitive and easy to follow.
    - Responsiveness: Discuss how the design adapts to different screen sizes (e.g., desktop, tablet, mobile) using responsive design principles with Tailwind CSS. Mention any breakpoints used to adjust the layout for smaller devices.
- Technologies Used:
    - React.js: A JavaScript library for building user interfaces, allowing for efficient vehicle listing rendering.
    - Tailwind CSS: A utility-first CSS framework that makes it easier to design the front end with minimal custom CSS.
- Design Considerations:
    - User Experience: Focus on the ease of use. For example, explain how you ensured users could quickly find vehicles based on price, kilometers, etc.

- Visual Design: Discuss how you used Tailwind CSS to apply colors, spacing, typography, and other design elements to create an aesthetically pleasing UI.

2.4 Back-End Design

This section explains the architecture and design of the back-end system.

- Server-Side Setup:
  - The server uses Node.js for asynchronous handling and Express.js for routing and middleware. Describe the server configuration, setting up the environment, and managing routes.
- API Design:
  - RESTful API Endpoints: Describe the key API endpoints, such as:
    - GET /api/cars: Retrieves the list of vehicles.
    - GET /api/cars/:id: Fetches detailed information about a specific vehicle.
    - POST /api/cars: Allows adding new vehicles (if required for admin purposes).
    - PUT /api/cars/:id: Allows updating vehicle details.
    - DELETE /api/cars/:id: Allows deleting a vehicle (if this functionality is provided).
  - Query Parameters for Filtering: The API should support query parameters like price, km, and year for filtering the vehicle listings, as well as support pagination via parameters like page and limit.
- Database Interaction:
  - The back-end communicates with MongoDB to store, retrieve, and modify vehicle data. Mongoose is used to create the schema for User and Car.

- Security & Validation:
  - Added input validations in all fields from the UI side and duplication validation for VIN in the Car model.

3. Implementation

This section outlines the actual implementation process of our vehicle listing application.

- System Architecture:
    - It’s a client-server model where the front end (React.js) communicates with the back end (Node.js/Express.js) via API requests. The front end fetches data from MongoDB through the back-end API.
- Code Structure:
    - Front-End: Describe the structure of your React project, such as the folder organization (e.g., components/, pages/, services/ for API calls).
    - Back-End: The structure of our Node.js/Express app includes folders for routes, controllers, and models.

3.1 Technologies Used

These list all the technologies and tools used to build your application.

- React.js: For building the dynamic user interface with a component-based architecture.
- Tailwind CSS: They offer utility-first CSS classes for styling the front end for faster design.
- Node.js: JavaScript runtime for the back-end to handle requests asynchronously.
- Express.js: Web framework for Node.js to simplify routing and middleware.
- MongoDB: NoSQL database for storing vehicle information. MongoDB was chosen for its flexibility and scalability, especially for handling unstructured data like vehicle details and images.

3.2 RESTful Web Services

This section explains how the RESTful API was implemented to interact with the database and provide data to the front end.

- API Endpoints:
    - GET /api/cars: Retrieves a paginated list of vehicles with optional filters for price and kilometers.
    - GET api/cars/:id: Retrieves detailed information about a single vehicle based on the vehicle ID.
- Request and Response Format:
    - Requests are made in JSON format, and the server responds with JSON. For instance, when retrieving vehicle details, the response would include the vehicle's make, model, year, price, etc.
- Authentication: Currently, users must register and log in to view all cars. Then, they can add and edit cars and see all other pages.

3.3 Front-End Implementation

Here is the description of how we implemented the front-end features.

- Components:
    - Carlist: Displays the list of cars with pagination.
    - CarCard: A card displaying each vehicle's summary details (like image, price, and title).
    - CarDetails: A page that shows detailed information about a selected vehicle, including images and full specs.
- State Management:
    - Use React hooks (useState, useEffect) to manage the component state, including vehicle data, filter options, and pagination state.
- API Integration:
    - The front end uses fetch or axios to send requests to the back end API, retrieve vehicle listing details, and apply filters using a js file called api.js.

3.4 Back-End Implementation

Here is the description of how we implemented the back-end features.

- Routes and Controllers:
    - Car Route: Handles HTTP requests for car data, including fetching, adding, updating, and deleting cars.
    - Controller Logic: Explains how the controller processes requests, communicates with the database and returns the appropriate response.
- Database Interaction:
    - MongoDB is queried using the Mongoose library, which is how vehicle data is manipulated.
- Error Handling:
    - Errors are caught and returned to the client (e.g., validation errors, database errors).

4. Features and Functionalities

This section outlines the core features of our application and how they work.

- Car Listings: The application shows a list of vehicles with key details (name, model, price, etc.) and images.
- Search and Filter: Users can filter vehicles by price, kilometers, year, or other attributes.
- Pagination: Cars are displayed in paginated grids of 16 per page.
- Car Details: Users can click on a vehicle to view more detailed information about it.

4.1 Vehicle Filtering

- Filtering Options: Users can filter vehicles by price, kilometers driven, etc.
- Implementation: Filters are applied on the front end (for UI updates) and the back end (to query the database).

4.2 Vehicle Details Page

- Detailed View: Once a user selects a car, the application redirects them to a detailed page where they can see all the information about the car, including images, specs, prices, and more.
- Implementation: React routes handle the navigation to this page and how the data is fetched from the server.

5. Testing and Validation

Here, you explain how you ensured the system worked as expected.

- Api Tests: Api testing using Postman is done.
- Integration Testing: Added console logs to the different parts of the system (front-end, back-end, and database) and were tested together.

5.1 Test Scenarios

- Vehicle Filtering: Tested if the filter shows the correct vehicles based on user input.
- Pagination: Tested if pagination works correctly and users can navigate between pages.
- Data Integrity: Ensure that data retrieved from the back-end matches the database.

5.2 Results

These were the results of our tests:

- Test Results: The application works as expected. Were there some issues or bugs encountered? Later, it was fixed with better code.
- Fixes: Used Stack overflow and documentation from official websites.

6. Conclusion and Future Enhancements

These are the conclusions and future improvements.:

- Successes: Overall, it was a great experience working with the front end, back end, and database. We learned middleware usages, css libraries like Tailwind Css, etc. We used Postman to test the API, and some bugs were fixed using the debugging technique.
- Future Improvements: These are the future enhancements, such as adding user authentication or search filters.

7. References

- **React.js Documentation**: https://reactjs.org/docs/getting-started.html
- **Node.js Documentation**: https://nodejs.org/en/docs/
- **Express.js Documentation**: https://expressjs.com/en/starter/installing.html
- **MongoDB Documentation**: https://www.mongodb.com/docs/

8. Appendices

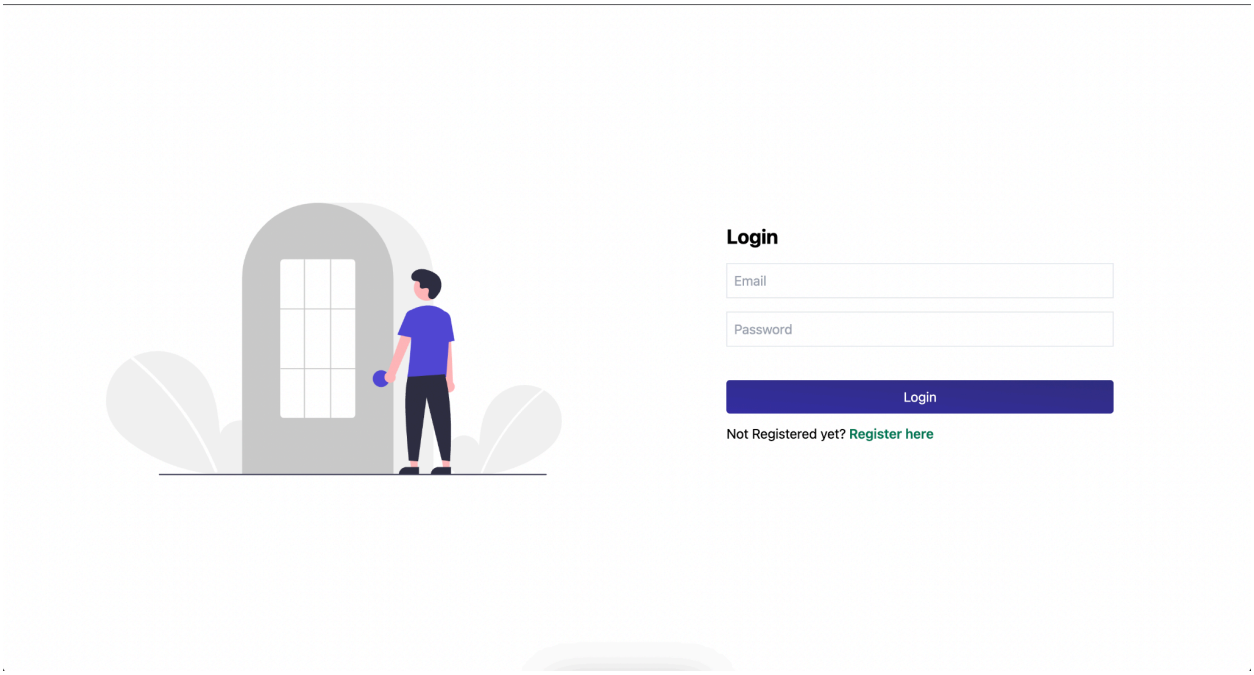
8.1 Code Snippets:

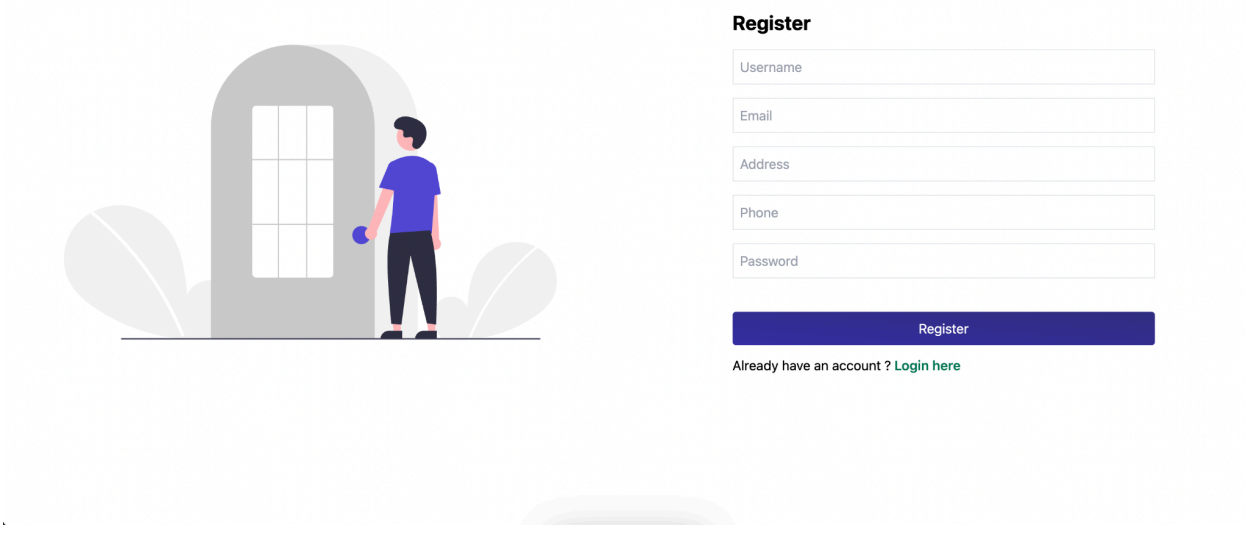
Backend app.js code:

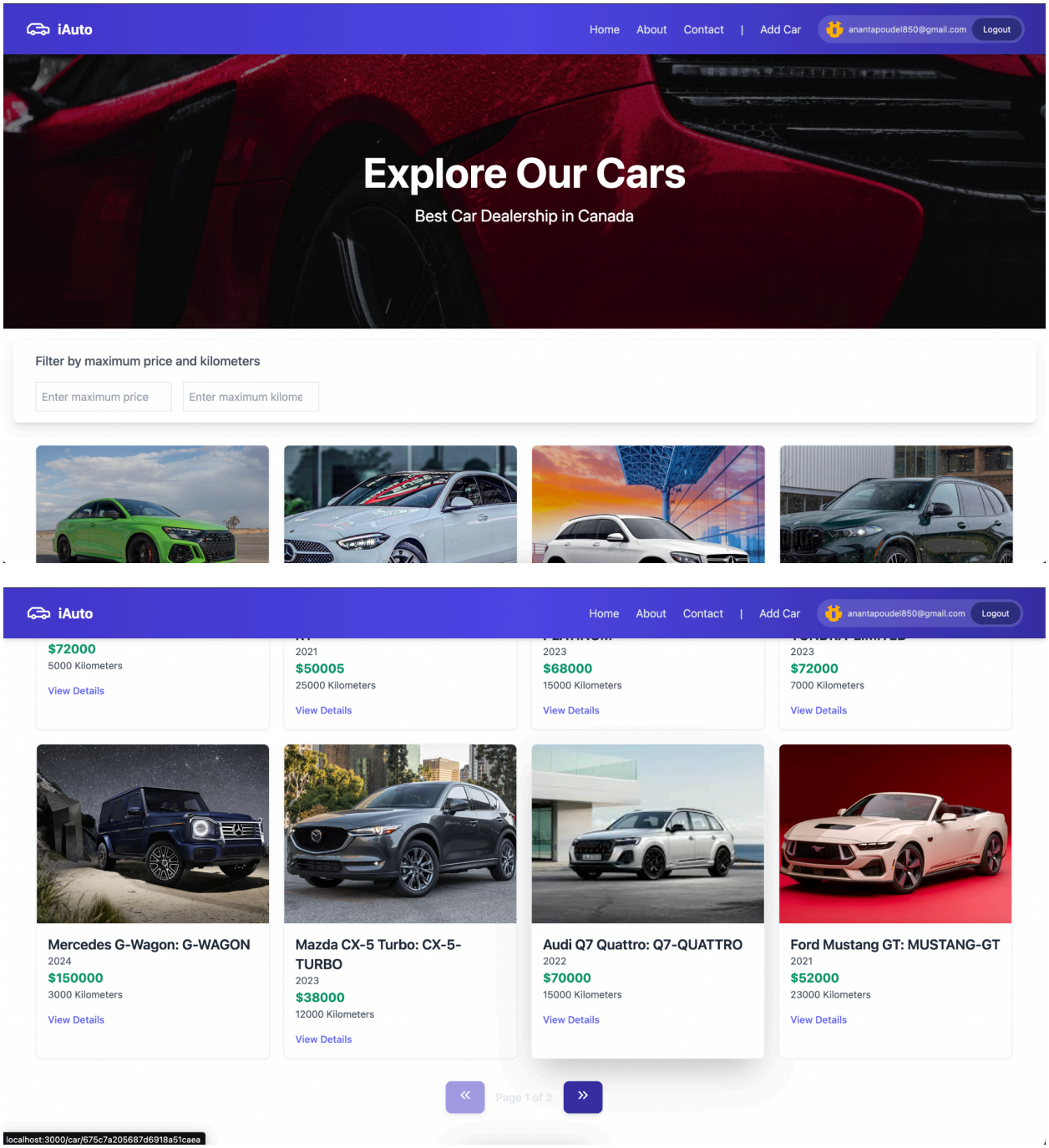
```
const express = require("express");
const cors = require("cors");
const mongoose = require("mongoose");

const userRoutes = require("./routes/userRoute");
const carRoutes = require("./routes/carRoutes");
const jwt = require("jsonwebtoken");

const app = express();

// MongoDB connection
const mongoURI = "mongodb://localhost:27017/dealership";
mongoose
 .connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })
 .then(() => console.log("MongoDB connected..."))
 .catch((err) => console.error(err));

//Middlewaress
app.use(express.json());
app.use(cors());
```

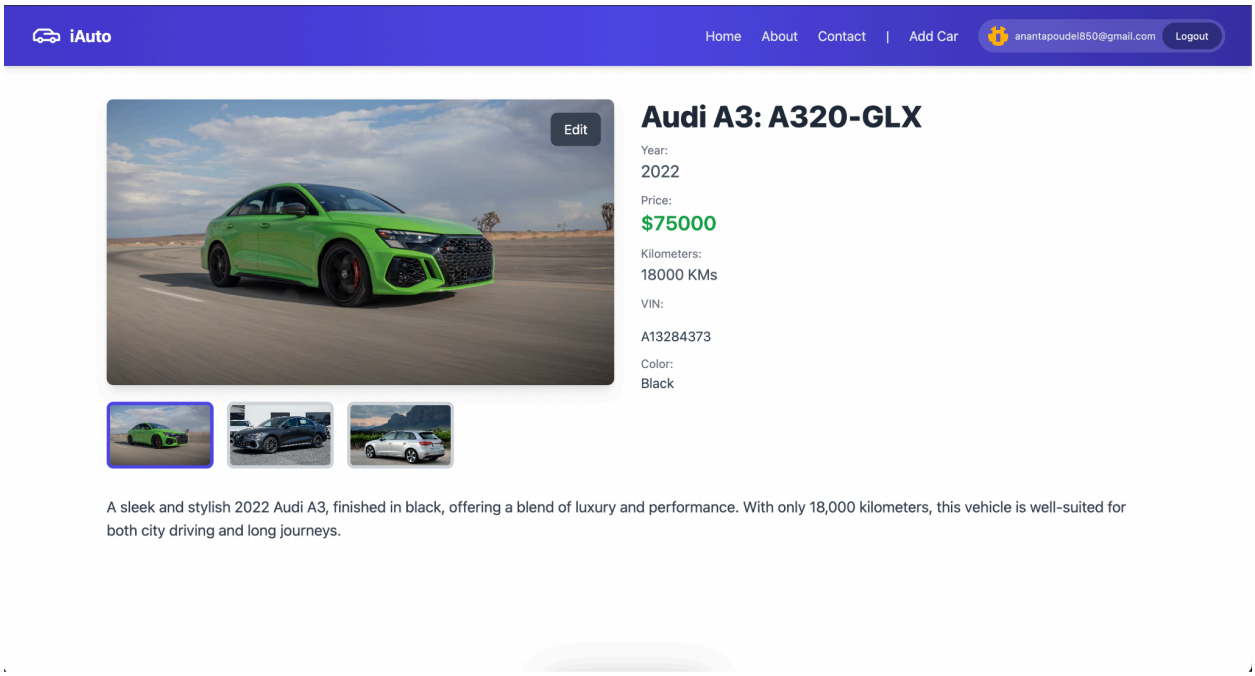
```
const authenticateToken = (req, res, next) => {

 const authHeader = req.header("Authorization");

 const token = authHeader && authHeader.split(" ")[1];

 if (!token)

   return res

     .status(401)

     .json({ message: "Access denied. No token provided." });



 try {

   const decoded = jwt.verify(token, "DealerShipSecretKey");

   req.user = decoded;

   next();

 } catch (ex) {

   res.status(400).json({ message: "Invalid token." });

 }

};



// Routes

app.use("/api/auth", userRoutes);

app.use("/api/cars", authenticateToken, carRoutes);



// Start server

const PORT = 5500;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

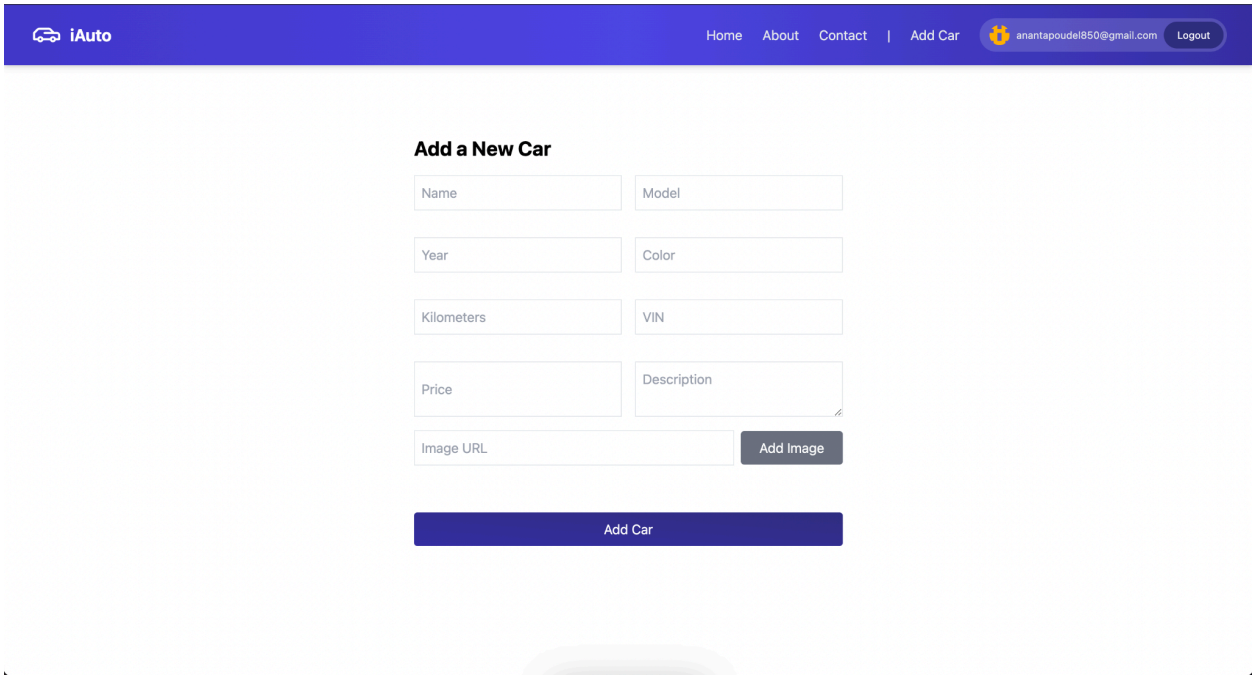
8.2 Screenshots:

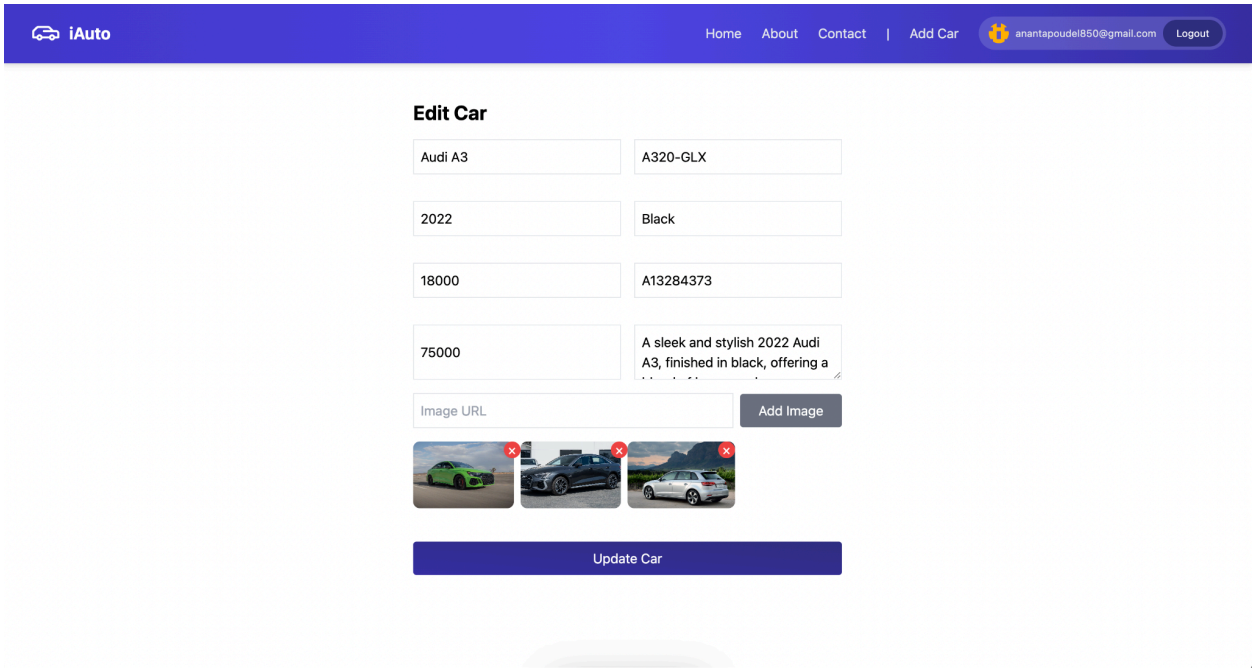





iAuto
Home
About
Contact
|
Add Car
anantapoudel850@gmail.com
Logout
Explore Our Cars
Best Car Dealership in Canada
Filter by maximum price and kilometers
Enter maximum price
Enter maximum kilome
$72000
5000 Kilometers
View Details
2021
$50005
25000 Kilometers
View Details
2023
$68000
15000 Kilometers
View Details
2023
$72000
7000 Kilometers
View Details
Mercedes G-Wagon: G-WAGON
2024
$150000
3000 Kilometers
View Details
Mazda CX-5 Turbo: CX-5-TURBO
2023
$38000
12000 Kilometers
View Details
Audi Q7 Quattro: Q7-QUATTRO
2022
$70000
15000 Kilometers
View Details
Ford Mustang GT: MUSTANG-GT
2021
$52000
23000 Kilometers
View Details
Page 1 of 2
localhost:3000/car/675c7a205687d6918a51caea


iAuto
Home
About
Contact
|
Add Car
anantapoudel850@gmail.com
Logout
Edit
Audi A3: A320-GLX
Year:
2022
Price:
$75000
Kilometers:
18000 KMs
VIN:
A13284373
Color:
Black
A sleek and stylish 2022 Audi A3, finished in black, offering a blend of luxury and performance. With only 18,000 kilometers, this vehicle is well-suited for both city driving and long journeys.


iAuto
Home
About
Contact
|
Add Car
anantapoudel850@gmail.com
Logout
Add a New Car
Name
Model
Year
Color
Kilometers
VIN
Price
Description
Image URL
Add Image
Add Car


iAuto
Home
About
Contact
|
Add Car
anantapoudel850@gmail.com
Logout
Edit Car
Audi A3
A320-GLX
2022
Black
18000
A13284373
75000
A sleek and stylish 2022 Audi A3, finished in black, offering a
Image URL
Add Image
Update Car

Our Team

Bibek Shrestha
Lead Developer

Subash Pariyar
Software Developer

Ananta Poudel
Product Manager

Bikash Sapkota
UI/UX Designer

Our Vision

To revolutionize car management solutions, paving the way for a smarter and more connected automotive future.