Day3 (Rental Car System)

API INTEGRATION OVERVIEW

API integration is a critical step in building the Rental Car System. It involves connecting the frontend with the backend to dynamically fetch, display, and manage car data.

Key Objectives:

- Adjust schema for rental cars in Sanity CMS.
- Fetch car data from Sanity CMS and display it dynamically.
- Implement sorting functionality for a better user experience.
- Ensure the system is responsive and error-free.

SCHEMA ADJUSTMENTS

Sanity CMS Schema Setup:

1. **Define the Schema:**

    a. Create a schema for rental cars, including fields like `Car Name`, `Price per Day`, `Location`, `Availability`, and `Image`.

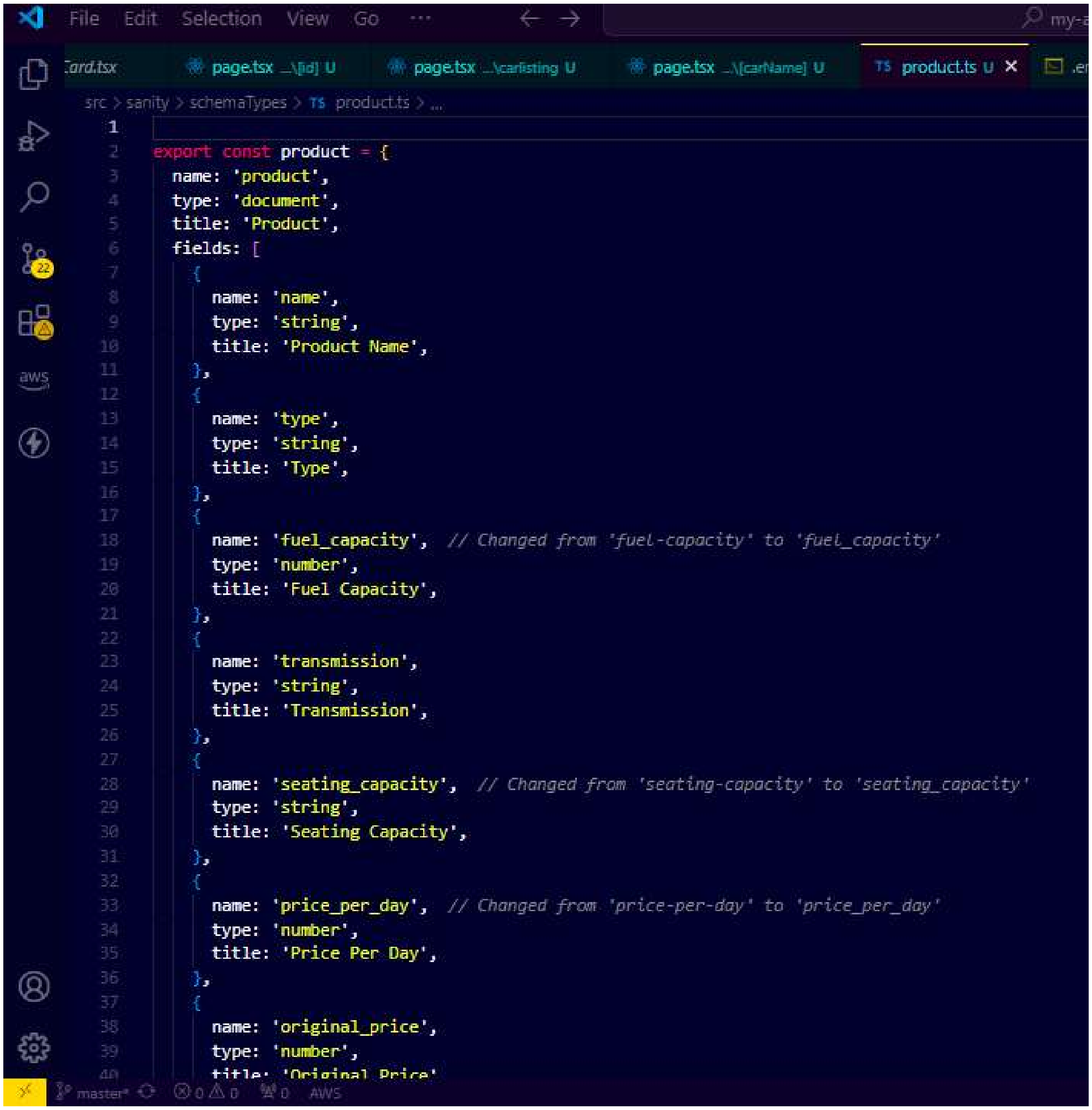
    b. Each field is carefully chosen to ensure the system captures all necessary details about a car.

2. **Deploy Schema:**

    b. After defining the schema, deploy it to Sanity CMS using the `sanity deploy` command.

    a. This ensures the schema is available in the CMS for data management.

Here is a code snippet:



```
export const product = {
  name: 'product',
  type: 'document',
  title: 'Product',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Product Name',
    },
    {
      name: 'type',
      type: 'string',
      title: 'Type',
    },
    {
      name: 'fuel_capacity',  // Changed from 'fuel-capacity' to 'fuel_capacity'
      type: 'number',
      title: 'Fuel Capacity',
    },
    {
      name: 'transmission',
      type: 'string',
      title: 'Transmission',
    },
    {
      name: 'seating_capacity',  // Changed from 'seating-capacity' to 'seating_capacity'
      type: 'string',
      title: 'Seating Capacity',
    },
    {
      name: 'price_per_day',  // Changed from 'price-per-day' to 'price_per_day'
      type: 'number',
      title: 'Price Per Day',
    },
    {
      name: 'original_price',
      type: 'number',
      title: 'Original Price'
```

FETCHING DATA FROM APIS

Fetching data from APIs is an essential step in integrating the backend with the frontend. This process enables dynamic display of data, like rental car details, directly on the website.

Use Axios for API Calls:

- Install Axios:

```
npm install axios
```

```
Fetch the data using axios.get() in your frontend code. Example:

axios.get('https://example.com/api/cars')
  .then(response => {
    console.log(response.data); // Log the data for debugging
  })
  .catch(error => {
    console.error('Error fetching data:', error); // Handle API errors
  });
```

Benefits of Fetching API Data:

- Provides a **real-time experience** by displaying updated information dynamically.
- Reduces the need for hardcoding, making the system scalable.
- Enhances user experience with features like **filtering and sorting** based on fetched data.

Connecting the Frontend and Backend:

1. **Create an API Endpoint:**
    a. Set up an API route in the Next.js project to fetch rental car data from Sanity CMS.

```
  https://sanity-nextjs-application.vercel.app/api/hackathon/template7
```

2. **Fetch Data Dynamically:**
    a. Use Axios in the frontend to call the API and retrieve car data.
    b. Ensure proper error handling to display fallback content if the API call fails.

3. **Environment Variables:**
    a. Store the Sanity project ID and API token in a `.env.local` file for security.

MIGRATION AND TOOLS USED

After defining the schema, the next step is **data migration** and utilizing appropriate tools to ensure the data is properly structured, transferred, and integrated with the application. This is crucial when switching between environments or updating schemas.

What is Data Migration?

Migration involves moving or adapting data from one schema structure to another to reflect new requirements or changes. In this project, data migration ensures the updated schema fields (e.g., adding new fields like "fuel type" or "car type") are reflected across the backend.

Steps for Migration

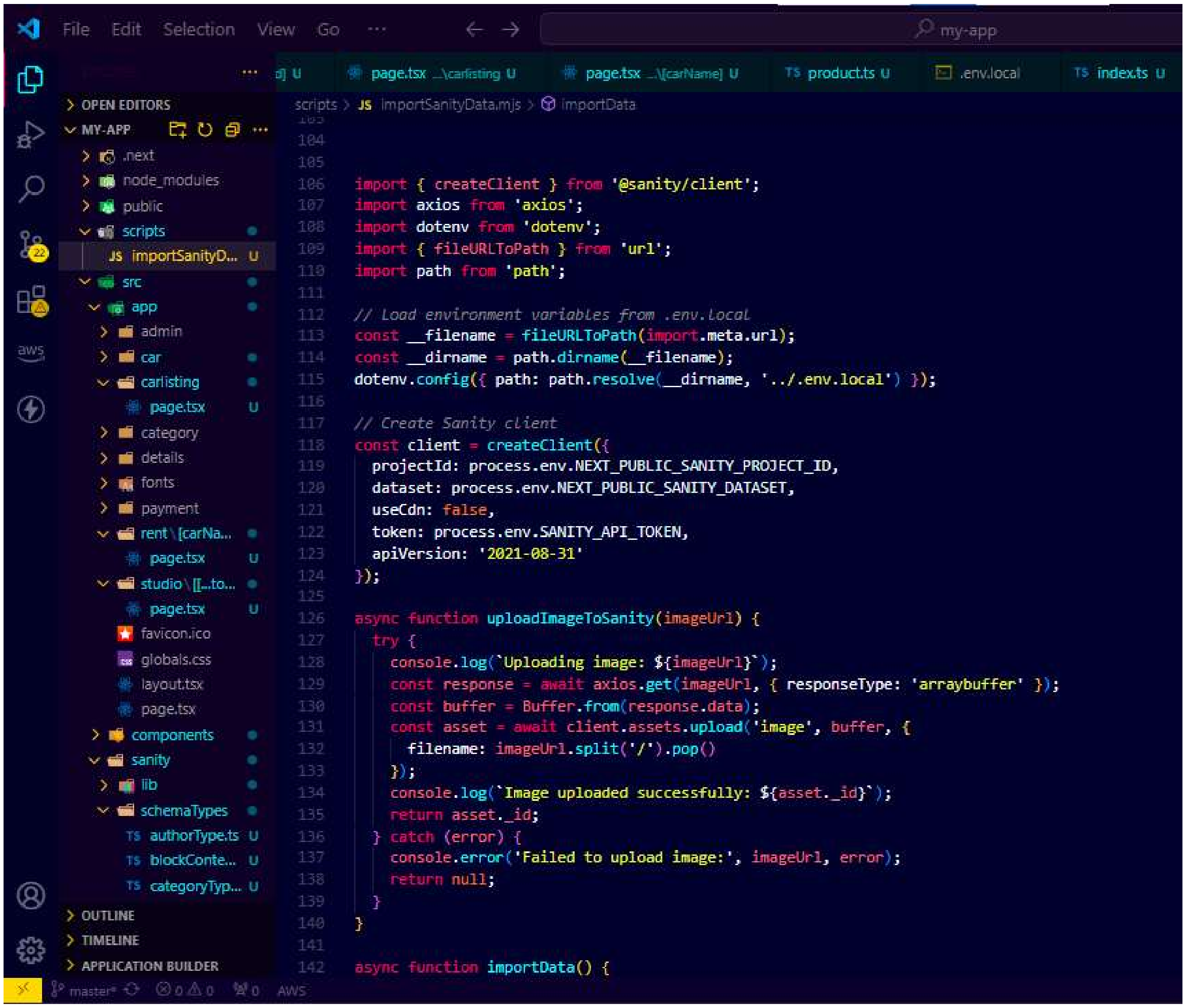
1. **Update Schema in Sanity CMS:**
    a. Modify or add fields in the schema file.
    b. Example: Add a new field `fuelType` (e.g., Petrol, Diesel, Electric).
2. **Deploy Schema Changes:**
    a. Deploy the updated schema to Sanity:

```
sanity deploy
```

    b. This updates the CMS structure to match the new schema.

3. **Run Migration Script:**

    a. Write a migration script to update existing data to fit the new schema.



```
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

// Load environment variables from .env.local
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: path.resolve(__dirname, '../.env.local') });

// Create Sanity client
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: false,
  token: process.env.SANITY_API_TOKEN,
  apiVersion: '2021-08-31'
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop()
    });
    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}

async function importData() {
```

Tools Used

1. **Sanity CMS:**
    a. Used to manage the schema and backend data dynamically.
    b. Tools within Sanity:
        i. **Studio:** To manage and deploy schema changes.
        ii. **API Client:** To fetch and update data.
        iii. **Dataset Management:** To store and organize data.
2. **Axios:**
    a. For making API requests to fetch or update data on the frontend.
3. **Node.js (For Scripts):**
    a. Write migration scripts to handle data updates programmatically.
4. **Postman (Optional):**
    a. Test API endpoints to ensure proper data flow and debugging.
5. **Git Version Control:**
    a. Track changes to schema files and migration scripts.

Backend Display Details

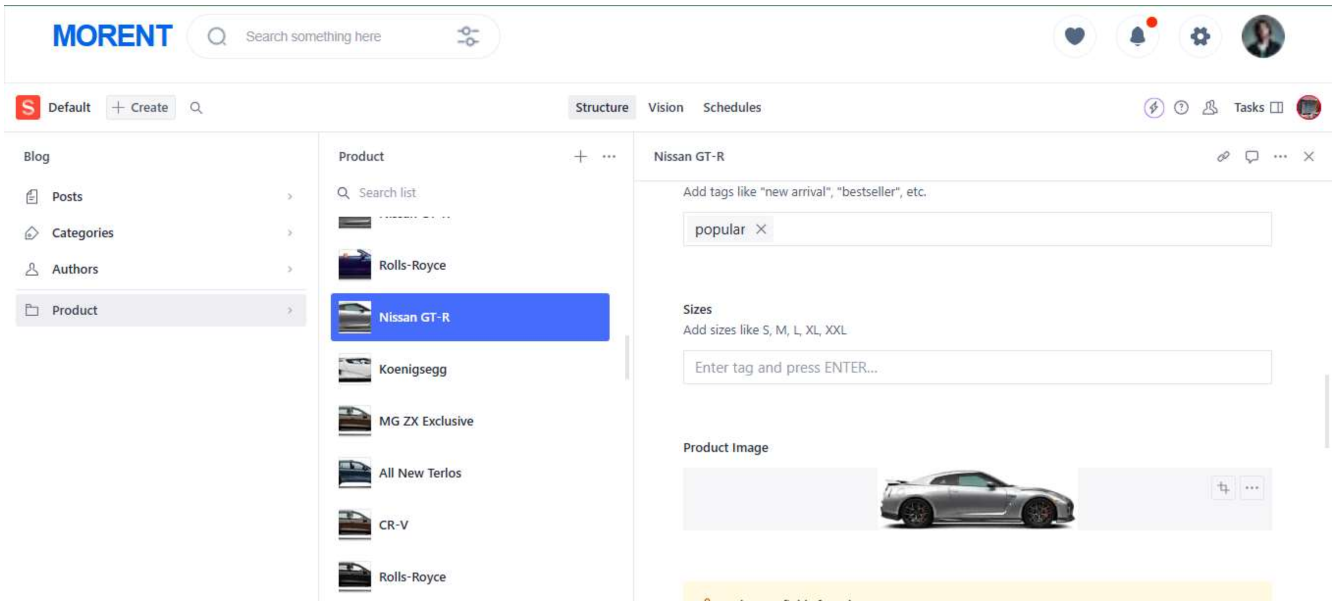
The backend in this project uses **Sanity CMS** to manage car data, which is then served to the frontend via APIs. The backend is responsible for:

1. **Storing Data**:

2. Schema-defined rental car data, including car name, price, location, image URL, etc.

3. **API Integration**:

Endpoints are exposed to fetch this data dynamically.

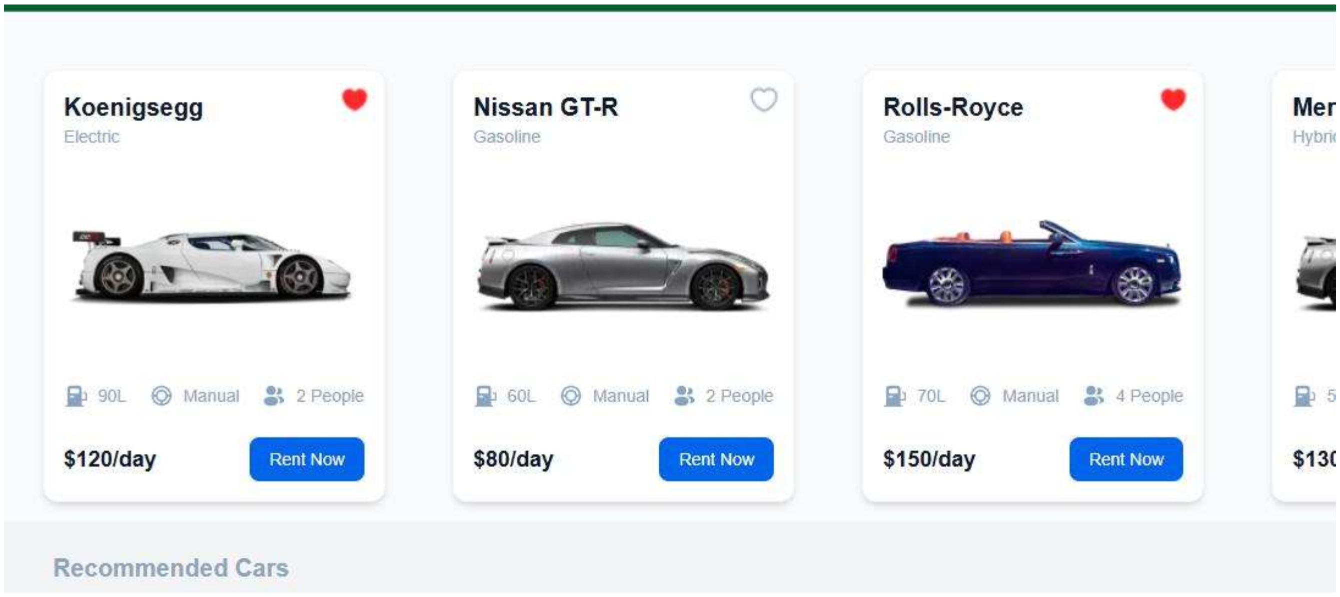


Frontend Display (Details)

The frontend part of the rental car application is responsible for rendering the car data fetched from the backend API dynamically

Output Example

- **Car Cards**: Each car card will show the image, name, location, price, and a button to view details.
- **Error Handling**: If the data fails to load, a message will notify the user.
- **Loading State**: Displays a loading message until the cars are fetched from the backend.



Koenigsegg
Electric
90L Manual 2 People
$120/day

Nissan GT-R
Gasoline
60L Manual 2 People
$80/day

Rolls-Royce
Gasoline
70L Manual 4 People
$150/day

Mer
Hybri
5
$130

Recommended Cars

Task Category	Task Description	Status
	Backend Checklist	
API Setup	Set up API to fetch car data from the backend and ensure proper endpoints are created.	✓
API Integration	Integrate API with frontend using Axios/Fetch. Ensure correct response and error handling.	✓
Data Migration	Migrate product data to the backend and verify data consistency.	✓
	Frontend Checklist	
Fetching Data	Set up data fetching using Axios/Fetch API. Handle loading states and errors.	✓
Dynamic Data Display	Map over fetched data to display car info (image, name, price, location).	✓
Styling & Responsiveness	Style the page using Tailwind CSS. Make the layout responsive.	✓
Error Handling	Display loading spinners and error messages when necessary.	✓
	General Checklist	
Testing	Test API calls, frontend display, error handling, routing, and responsiveness across devices.	✓
Version Control	Commit regularly and ensure the repository is organized and clean.	✓
Deployment	Deploy to platforms like Vercel or Netlify, test the live version, and set up environment variables.	✓