Hackathon 5 - DAY 3
API Integration Report

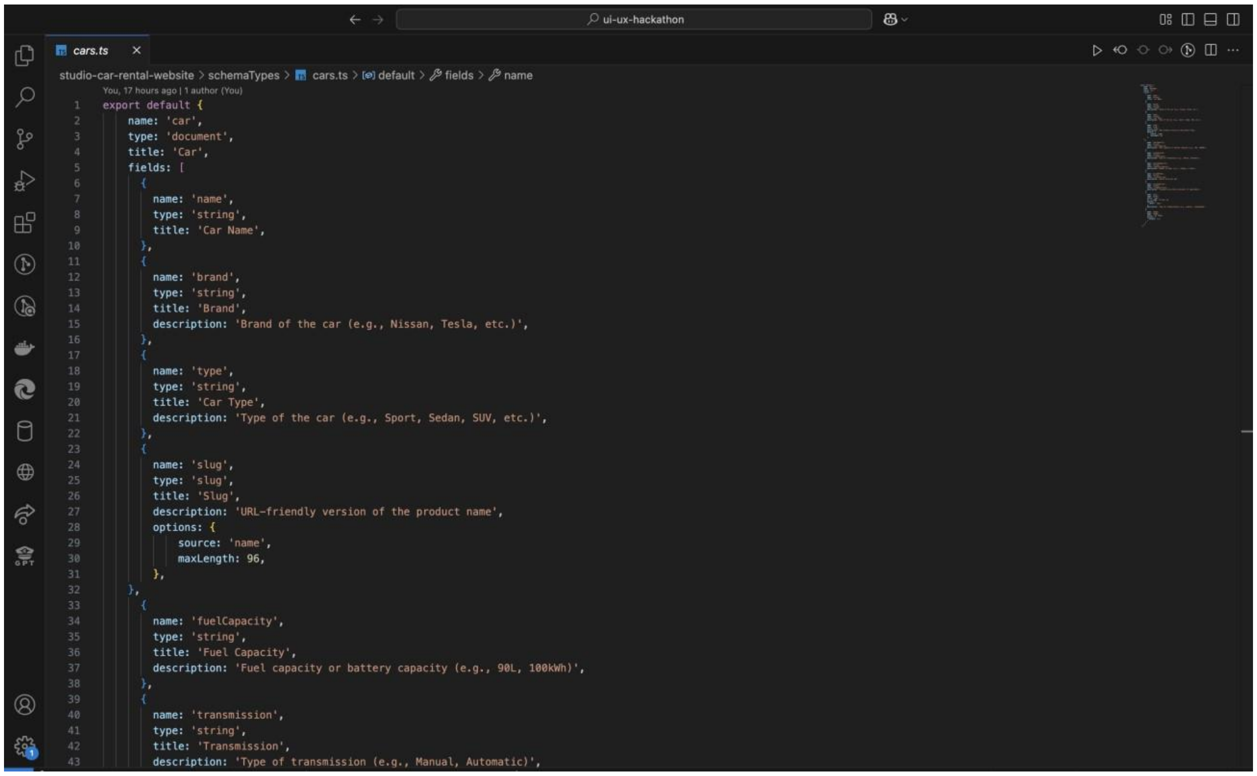
Overview

This report outlines the steps taken to integrate an API with Sanity CMS and how the data was utilized in a Next.js application. The integration process includes setting up the Sanity schema, querying data using GROQ, and rendering data dynamically in Next.js components. Visual aids have been included to provide clarity.

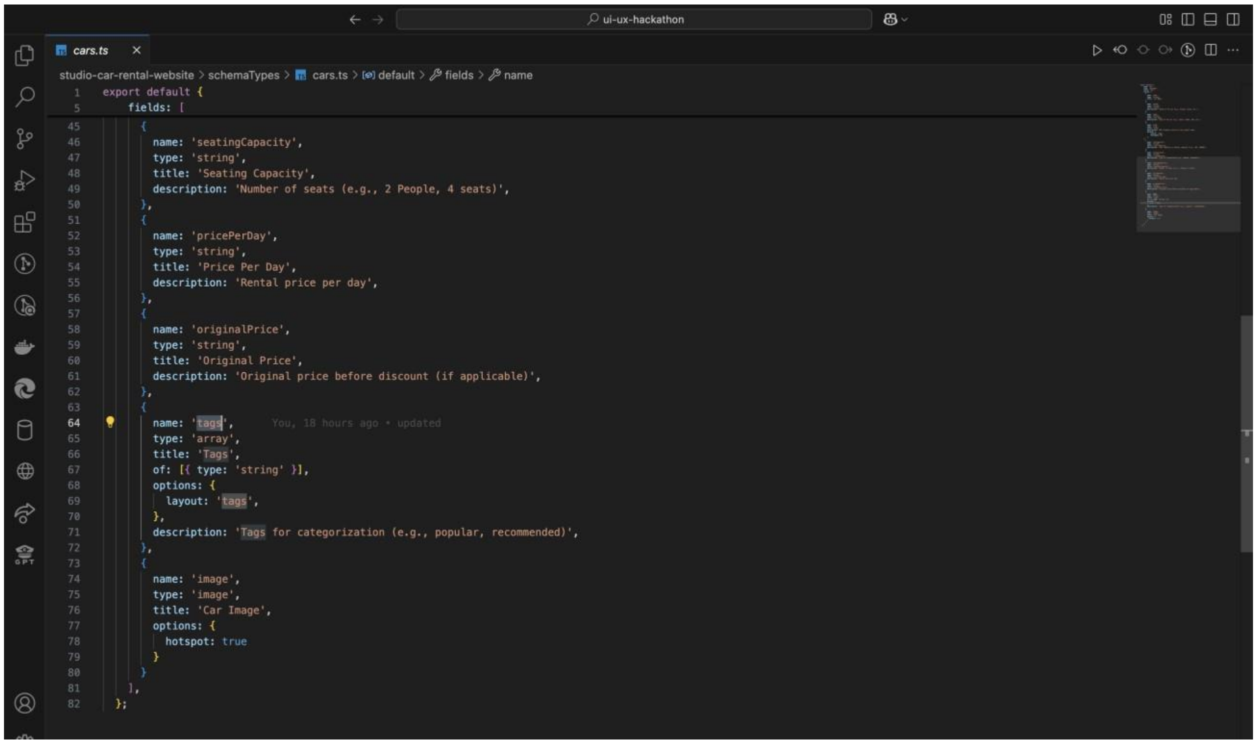
1. API Integration with Sanity CMS

Step 1: Setting Up Sanity CMS

Sanity CMS was configured to serve as the backend for storing car rental data. The following schema was created to define the structure of car details:


ui-ux-hackathon
cars.ts
studio-car-rental-website > schemaTypes > cars.ts > default > fields > name
You, 17 hours ago | 1 author (You)

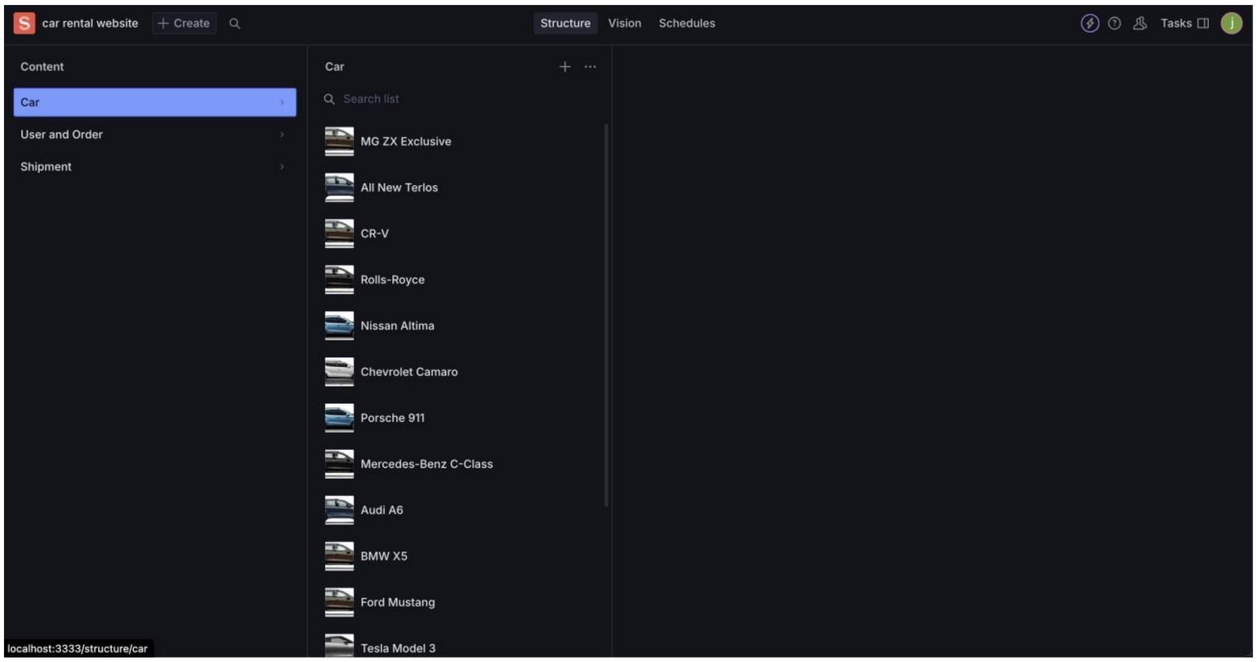
```
export default {
    name: 'car',
    type: 'document',
    title: 'Car',
    fields: [
      {
        name: 'name',
        type: 'string',
        title: 'Car Name',
      },
      {
        name: 'brand',
        type: 'string',
        title: 'Brand',
        description: 'Brand of the car (e.g., Nissan, Tesla, etc.)',
      },
      {
        name: 'type',
        type: 'string',
        title: 'Car Type',
        description: 'Type of the car (e.g., Sport, Sedan, SUV, etc.)',
      },
      {
        name: 'slug',
        type: 'slug',
        title: 'Slug',
        description: 'URL-friendly version of the product name',
        options: {
            source: 'name',
            maxLength: 96,
        },
    },
      {
        name: 'fuelCapacity',
        type: 'string',
        title: 'Fuel Capacity',
        description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
      },
      {
        name: 'transmission',
        type: 'string',
        title: 'Transmission',
        description: 'Type of transmission (e.g., Manual, Automatic)',
```


ui-ux-hackathon
cars.ts
studio-car-rental-website > schemaTypes > cars.ts > default > fields > name

```
export default {
    fields: [
      {
        name: 'seatingCapacity',
        type: 'string',
        title: 'Seating Capacity',
        description: 'Number of seats (e.g., 2 People, 4 seats)',
      },
      {
        name: 'pricePerDay',
        type: 'string',
        title: 'Price Per Day',
        description: 'Rental price per day',
      },
      {
        name: 'originalPrice',
        type: 'string',
        title: 'Original Price',
        description: 'Original price before discount (if applicable)',
      },
      {
        name: 'tags',        You, 18 hours ago • updated
        type: 'array',
        title: 'Tags',
        of: [{ type: 'string' }],
        options: {
          layout: 'tags',
        },
        description: 'Tags for categorization (e.g., popular, recommended)',
      },
      {
        name: 'image',
        type: 'image',
        title: 'Car Image',
        options: {
          hotspot: true
        }
      }
    ],
  };
```

Step 2: Populating Data in Sanity Studio

The schema was used to populate car details in the Sanity Studio interface. Each document was assigned a unique slug, which serves as the identifier for querying.

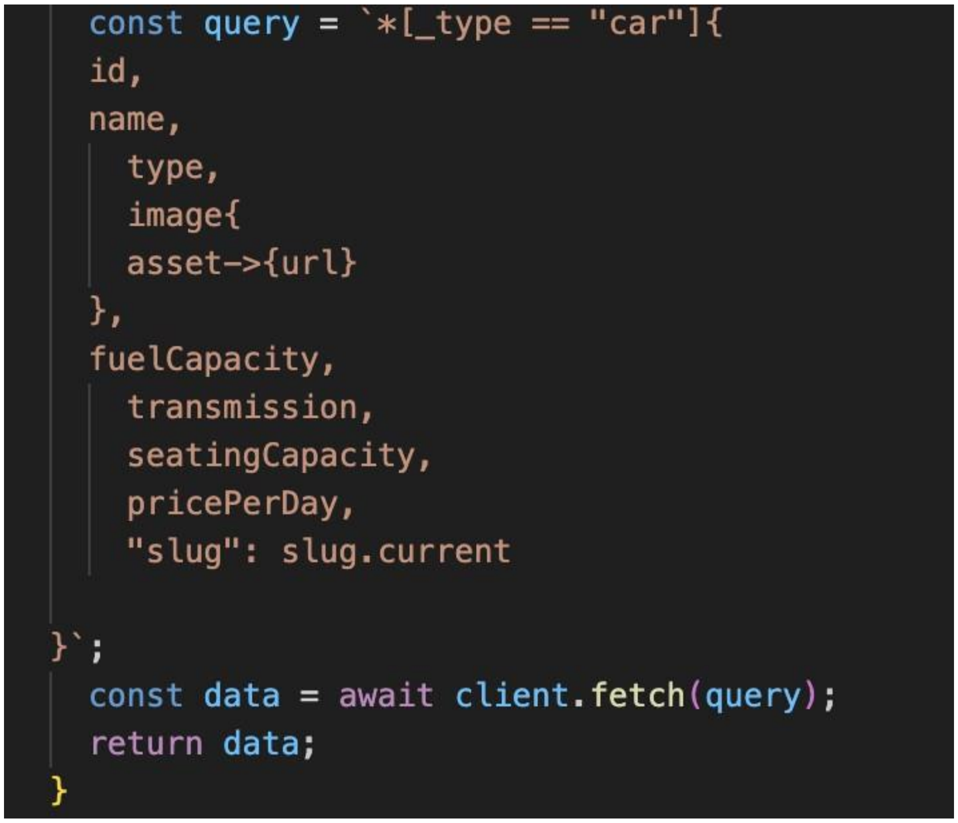


Car

- MG ZX Exclusive
- All New Terlos
- CR-V
- Rolls-Royce
- Nissan Altima
- Chevrolet Camaro
- Porsche 911
- Mercedes-Benz C-Class
- Audi A6
- BMW X5
- Ford Mustang
- Tesla Model 3

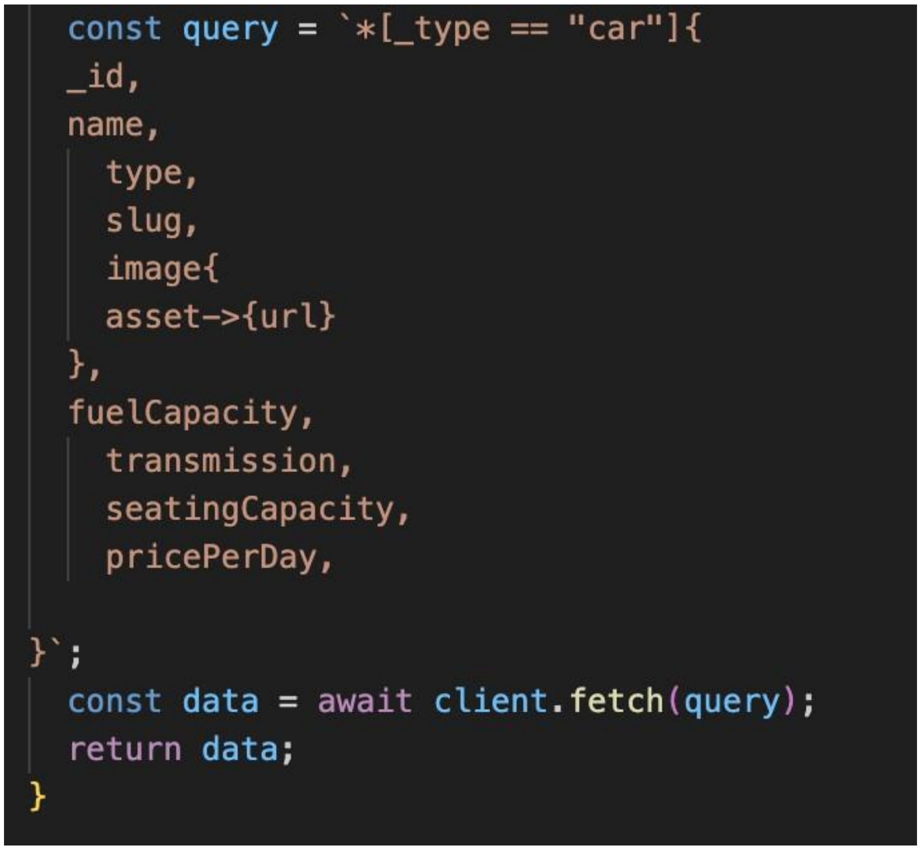
Step 3: Fetching Data from Sanity API

Sanity's GROQ (Graph-Relational Object Queries) was used to fetch data. A query was written to retrieve car details based on the provided slug:



```
  const query = `*[_type == "car"]{
  id,
  name,
    type,
    image{
    asset->{url}
  },
  fuelCapacity,
    transmission,
    seatingCapacity,
    pricePerDay,
    "slug": slug.current

}`;
  const data = await client.fetch(query);
  return data;
}
```



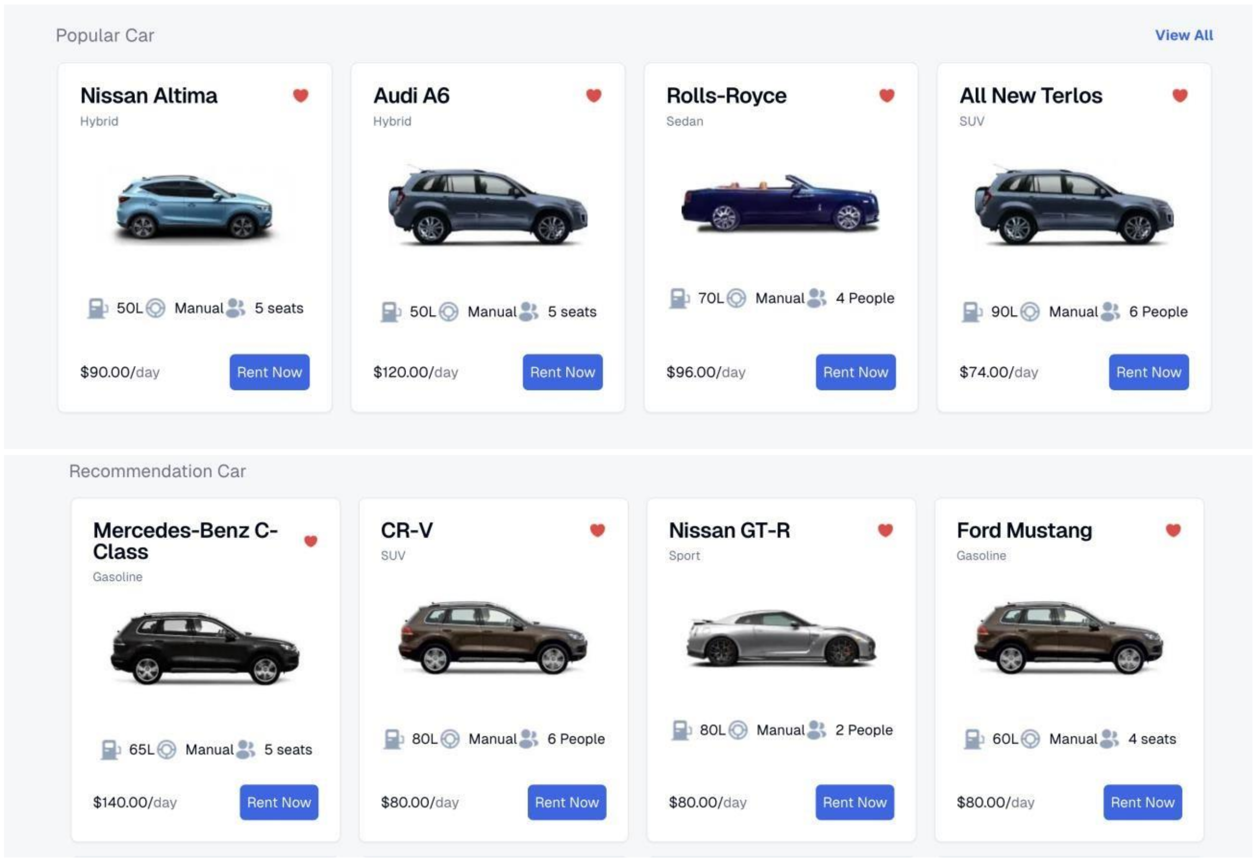
```
  const query = `*[_type == "car"]{
  _id,
  name,
    type,
    slug,
    image{
    asset->{url}
  },
  fuelCapacity,
    transmission,
    seatingCapacity,
    pricePerDay,

}`;
  const data = await client.fetch(query);
  return data;
}
```

2. Using the API in Next.js

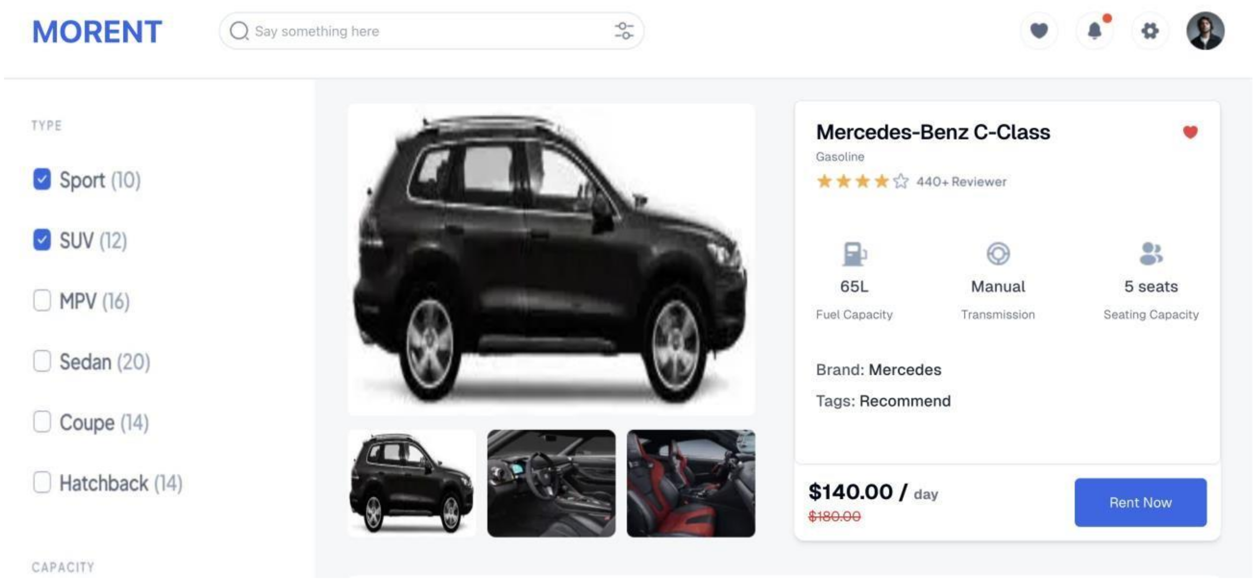
Step 1: Fetching Data in Next.js

The API was integrated into Next.js using Sanity's client library.



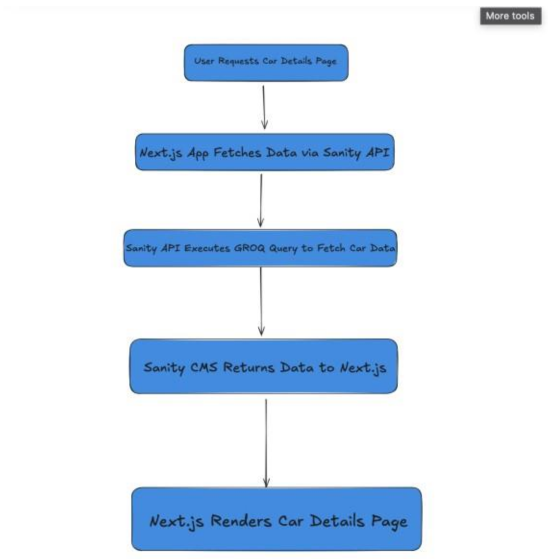
Step 2: Dynamic Routing in Next.js

The car details page was set up with a dynamic route



3. Diagram: Data Flow

Below is a visual representation of the data flow from Sanity CMS to the Next.js application:


More tools
User Requests Car Details Page
Next.js App Fetches Data via Sanity API
Sanity API Executes GROQ Query to Fetch Car Data
Sanity CMS Returns Data to Next.js
Next.js Renders Car Details Page