Hackathon Day 3
Day 3 - API Integration And Migration

1. API Integration Overview:

API integration is a key part of modern web development, especially when using a headless CMS like **Sanity** to manage content dynamically. By integrating APIs into your Next.js application, you can fetch data such as rental car listings, availability, pricing, and other relevant information directly from the backend to display it on the frontend.

The primary objectives of API integration in this context are:

- Fetch data dynamically from Sanity CMS (or any external data source).
- Allow the frontend to display real-time data (car availability, pricing).
- Adjust schema structures to align with the data being fetched.
- Ensure smooth interaction between the frontend (Next.js) and the backend (Sanity CMS or custom backend).

2. Key Objectives of API Integration:

- **Seamless Data Fetching**: Fetch car-related data (make, model, price, availability) from Sanity or external APIs dynamically.
- **Dynamic Data Rendering**: Render data in real-time on the frontend to reflect changes in car availability or pricing.
- **Error Handling**: Implement effective error handling to manage failed API calls or data inconsistencies.
- **API Call Optimization**: Use Axios or other HTTP clients for efficient API calls.
- **Frontend and Backend Communication**: Establish a clear communication pathway between frontend and backend via API endpoints.

3. Schema Adjustment:

The Sanity CMS schema defines how data is structured within the CMS, which must align with the data fetched from APIs. Schema adjustments ensure that the data from APIs maps correctly to the fields in your CMS, avoiding inconsistencies.

Steps for Schema Adjustment:

1. **Define Sanity Schema**: Create schema files that match the data structure you expect from the API (e.g., car model, availability).
2. **Map Fields**: Ensure that the field names in your API responses match the field names in your Sanity schema. If necessary, adjust or create new fields to accommodate the API data.
3. **Validate Schema**: Ensure the data types and relationships in Sanity (string, number, boolean) are correct and align with your API's response.

4. Sanity CMS Schema Setup:

To store data in Sanity CMS, you'll first need to define schemas that reflect your data model (for a rental car system, this might include cars, name, brand, type, fuel capacity, transmission, seating capacity, price perday,week , image).

Example Schema for Car (Car.js):

```
export default {
  name: 'carDataTypes',
  type: 'document',
  title: 'Car Data Types',
  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: 'fuelCapacity',
      type: 'string',
      title: 'Fuel Capacity',
      description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
      initialValue: 'Not specified',  // Default value
    },
    {
      name: 'transmission',
      type: 'string',
      title: 'Transmission',
      description: 'Type of transmission (e.g., Manual, Automatic)',
    },
    {
      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',
      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
      }
    }
  ],
};
```

This schema defines the structure for car data, including make, model, price per day, and availability.

5. Deploying the Schema:

After defining the schema in Sanity Studio, you can deploy it by following these steps:

1. **Save Schema Changes**: After making adjustments to your schema, save and commit them to your Sanity Studio project.
2. **Deploy to Sanity Studio**: Run the command to deploy your changes and push them to your live Sanity project.

Example command:

bash
Copy
sanity deploy

6. Data Migration Overview:

Data migration refers to the process of transferring data from one system to another. In this case, it refers to migrating car data into Sanity CMS. There are two common ways to migrate data:

1. **Manual Import**: Data is manually entered into the Sanity CMS interface.
2. **Automated Migration Scripts**: Data is imported programmatically via scripts (e.g., using Node.js or external APIs).

Steps for Data Migration:

1. **Extract Data**: Fetch data from your source (e.g., external APIs or CSV files).
2. **Transform Data**: If necessary, format or adjust the data to match your Sanity schema.
3. **Load Data**: Use Sanity's import tools or write scripts to load the data into Sanity CMS.

Example Data Migration

```
// export default Page;
"use client"
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv'; // You can still use dotenv for server-side operations
```

```
// Load environment variables from .env.local (Optional, since Next.js automatically loads them)
dotenv.config(); // Only needed if you are using a custom server setup
```

```
// Create Sanity client
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID, // Exposed as NEXT_PUBLIC_SANITY_PROJECT_ID
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,      // Exposed as NEXT_PUBLIC_SANITY_DATASET
  token: process.env.NEXT_PUBLIC_SANITY_API_TOKEN,      // Use the token from .env.local
  apiVersion: '2023-01-17', // You can use the current date or a fixed version
  useCdn: true, // Optionally set this to `false` during development for more accurate results
});
```

```
async function uploadImageToSanity(imageUrl : any) {
  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() {
  try {
    console.log('Fetching car data from API...');
```

```
    // API endpoint containing car data
    const response = await axios.get('https://sanity-nextjs-
application.vercel.app/api/hackathon/template7');
    const cars = response.data;
```

```
    console.log(`Fetched ${cars.length} cars`);
```

```
    for (const car of cars) {
      console.log(`Processing car: ${car.name}`);
```

```
      let imageRef = null;
      if (car.image_url) {
        imageRef = await uploadImageToSanity(car.image_url);
      }
      const sanityCar = {
        _type: 'car',
        name: car.name,
        brand: car.brand || null,
        type: car.type,
        fuelCapacity: car.fuel_capacity,
        transmission: car.transmission,
        seatingCapacity: car.seating_capacity,
        pricePerDay: car.price_per_day,
        originalPrice: car.original_price || null,
        tags: car.tags || [],
        image: imageRef ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        } : undefined,
      };
```

```
      console.log('Uploading car to Sanity:', sanityCar.name);
      const result = await client.create(sanityCar);
      console.log(`Car uploaded successfully: ${result._id}`);
    }
```

```
    console.log('Data import completed successfully!');
  } catch (error) {
    console.error('Error importing data:', error);
  }
}
```

```
importData();
```

7. Fetching Data from APIs (Using Axios):

To fetch data from an API (Sanity CMS or an external API), Axios is a great tool that simplifies HTTP requests. Here’s how you can use Axios to fetch car data from the API:

Installing Axios:

npm install axios

Fetching Data Example in Next.js:

```
j'use client'; // Mark this as a client component

import React, { useState, useEffect } from "react";
import { client } from "@/sanity/lib/client"; // Import Sanity client
import Link from "next/link";
```

```
const CarRecommendationPage = () => {
  const [cars, setCars] = useState<any[]>([]); // Add proper type annotation for cars array
  const [loading, setLoading] = useState<boolean>(true); // State for loading indication
```

```
  // Fetch car data from Sanity CMS
  useEffect(() => {
    const fetchCars = async () => {
      // Corrected Sanity query to fetch recommended cars
      const query = `*[_type == "carData" && "recommended" in tags] {
        _id,
        name,
        type,
        fuelCapacity,
        transmission,
        seatingCapacity,
        pricePerDay,
        "image_url": image.asset->url
      }`;
```

```
      try {
        const data = await client.fetch(query); // Fetch data from Sanity
        console.log('Fetched cars data:', data); // Debugging step to see fetched data
        setCars(data); // Set the fetched data to the state
        setLoading(false); // Stop loading when data is fetched
      } catch (error) {
        console.error("Error fetching cars data: ", error); // Log error in case of failure
        setLoading(false); // Stop loading even if there's an error
      }
    };
```

```
    fetchCars(); // Call the function to fetch data after the component mounts
  }, []); // Empty dependency array ensures this effect runs only once
```

```
  // State for managing favorites (heart icon toggle)
  const [favorites, setFavorites] = useState<{ [key: string]: boolean }>({});
```

```
  const handleFavoriteToggle = (carId: string) => {
    setFavorites((prevFavorites) => ({
      ...prevFavorites,
      [carId]: !prevFavorites[carId],
    }));
  };
```

```
  return (
    <div className="p-6 bg-gray-100">
      <h2 className="text-xl font-bold text-slate-400 text-left ml-4 mb-8">Recommended Cars</h2>
```

```
      {/* Show loading message or car grid */}
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 xl:grid-cols-4
2xl:grid-cols-5 gap-6">
        {loading ? (
          <p className="text-center text-gray-500">Loading cars...</p> // Show loading text while fetching
        ) : (
          cars.length > 0 ? (
            cars.map((car) => (
              <div
                key={car._id} // Use the car _id as the key for each element
                className="border rounded-lg p-4 bg-white shadow-lg"
              >
                <div className="flex justify-between items-center">
                    <h3 className="text-lg font-semibold mt-4">{car.name}</h3>
                  <div
                    className="cursor-pointer"
                    onClick={() => handleFavoriteToggle(car._id)}
                  >
                    <img
                      src={favorites[car._id] ? "/love4.png" : "/love5.png"} // Toggle heart image
                      alt="Heart Icon"
                      className="w-6 h-6"
                    />
                  </div>
                  {favorites[car._id] && <span className="text-sm text-red-500 font-
semibold">Favourite</span>}
                </div>
                <p className="text-sm text-gray-600">{car.type}</p>
                <img src={car.image_url} alt={car.name} className="w-full h-22 object-cover rounded-t-lg"
/>


                {/* Heart Icon Toggle */}
```

```
                <div className="flex gap-3 items-center justify-center">
                  <div className="flex">
```

```
                    <img
                      loading="lazy"
                      src="https://cdn.builder.io/api/v1/image/assets/TEMP/bb9f5fa088a33a8329469c11ed8f42f7df3e0fd11b9aa0921af94d8d3307f051?placeholderIfAbsent=true&apiKey=5967db0a3a5740a580d3441f6f0ec2df"
                      alt="Fuel Icon"
                      className="object-contain shrink-0 w-6 aspect-square"
                    />
                    <p className="text-sm text-gray-600"> {car.fuelCapacity}</p>
                  </div>
                  <div className="flex">
                    <img
                      loading="lazy"
                      src="https://cdn.builder.io/api/v1/image/assets/TEMP/563fd9367e8be9e271233fa362e88c8b2205c920475aad51a787f2599d87477e?placeholderIfAbsent=true&apiKey=5967db0a3a5740a580d3441f6f0ec2df"
                      alt="Transmission Icon"
                      className="object-contain shrink-0 w-6 aspect-square"
                    />
                    <p className="text-sm text-gray-600">{car.transmission}</p>
                  </div>
                  <div className="flex">
                    <img
                      loading="lazy"
                      src="https://cdn.builder.io/api/v1/image/assets/TEMP/fd12c9762ffaa585959a2bb1c514f631f14a3524f88d9c2bd9d3da13bf9fa3d9?placeholderIfAbsent=true&apiKey=5967db0a3a5740a580d3441f6f0ec2df"
                      alt="Capacity Icon"
                      className="object-contain shrink-0 w-6 aspect-square"
                    />
                    <p className="text-sm text-gray-600">{car.seatingCapacity}</p>
                  </div>
                </div>
                <div>
                  <p className="text-sm text-black font-bold">Price per Day: {car.pricePerDay}</p>
                  <Link href="/payment">
                    <button
                      className="gap-2 self-start px-6 py-3 mt-1 text-base font-medium tracking-tight text-center text-white bg-[#3563E9] rounded min-h-[10px] w-[130px] whitespace-nowrap"
                      aria-label={`Rent ${car.name} now`}
                    >
                      Rent Now
                    </button>
                  </Link>
                </div>
              </div>
            ))
          ) : (
            <p className="text-center text-gray-500">No recommended cars available.</p> // Message when no cars are available
          )
        )}
      </div>
```

```
      <div className="flex justify-center items-start pt-4">
        <button className="h-10 w-40 text-white bg-blue-500 rounded">
          Show More Cars
        </button>
      </div>
    </div>
  );
};
```

```
export default CarRecommendationPage;
```

Benefits of Fetching Data with APIs:

1. **Real-Time Data**: Always get up-to-date information from the backend.

2. **Scalability**: API integration allows you to scale your application easily by connecting to different backends.
3. **Dynamic Data**: API fetching allows you to dynamically change content on the frontend without re-deploying the site.

8. Using Environment Variables:

Environment variables are essential for managing sensitive data such as API keys. Create a .env.local file in your Next.js project to store environment variables securely.

Example of Environment Variables:

In your .env.local file:

```
makefile
Copy
SANITY_PROJECT_ID=your_project_id
SANITY_DATASET=production
```

Then, use these variables in your code:

```
javascript


Copy
const client = sanityClient({
  projectId: process.env.SANITY_PROJECT_ID,
  dataset: process.env.SANITY_DATASET,
  useCdn: true,
});
```

Table: How to Work with API Integration and Migration

Task Ctegory	Task Description	Details
Schema Setup	Define the structure of data in Sanity CMS	Create schemas that represent the expected data (e.g., car data schema).
Schema Adjustment	Adjust Sanity schema based on API data	Modify or create new fields to align the data structure between API and Sanity CMS.
Data Migration	Migrate data into Sanity CMS	Use either manual imports or automated migration scripts (using Node.js or APIs).
API Integration	Integrate external or Sanity APIs to fetch data dynamically	Use Axios to fetch data from APIs and display it in your application.
Environment Variables	Securely store sensitive information like API keys	Store credentials in .env.local for use in your Next.js application.
Frontend Communicati on	Communicate between frontend (Next.js) and backend (Sanity CMS or APIs)	Fetch and display real-time data in the frontend application.