DAY 3 - API INTEGRATION AND DATA MIGRATION

Introduction

This documentation outlines the process of integrating a mock API for a rental cars project into a Sanity CMS. It details the steps for setting up a clean Sanity project, defining a schema, importing API data, and fetching it using GROQ queries for local development.

1. Creating the Mock API

Define Your Data Structure:

Create a mock API that represents your rental car data. Each car object should include attributes like id, make, model, year, price, and any other relevant details.

Mock API Platform:

Use a tool like JSON Server, Mockaroo, Mock Api or any custom backend to create and host the mock API.

Test the API:

Ensure your API endpoint is working correctly and returning the expected data structure.

2. Setting Up Environment Variables

Configuring Environment Variables

Begin by setting up your environment variables. If a .env.local file doesn't already exist in your project's root directory, create one. Then, add the following variables:

```
.env.local
1  NEXT_PUBLIC_SANITY_PROJECT_ID="gb0kymx2"
2  NEXT_PUBLIC_SANITY_DATASET="production"
3  SANITY_API_TOKEN="skCdN4OSjrBgxuc7OyRfEh8g2ai81IjMzInxCC7czbAJM1jw4fxwy0Tdgw8CEoLXDpjjBtIdvw4rLJUNCrhytILJ69ko6aAm9y3ycjTGE
   ohI28eMsZ5GUvmpp6Pk3ywM1ljuv5IGQRLMRwBoNPXhSIgO964jjMNxBKXsyWXPJBgAshNx3tA5"SS
```

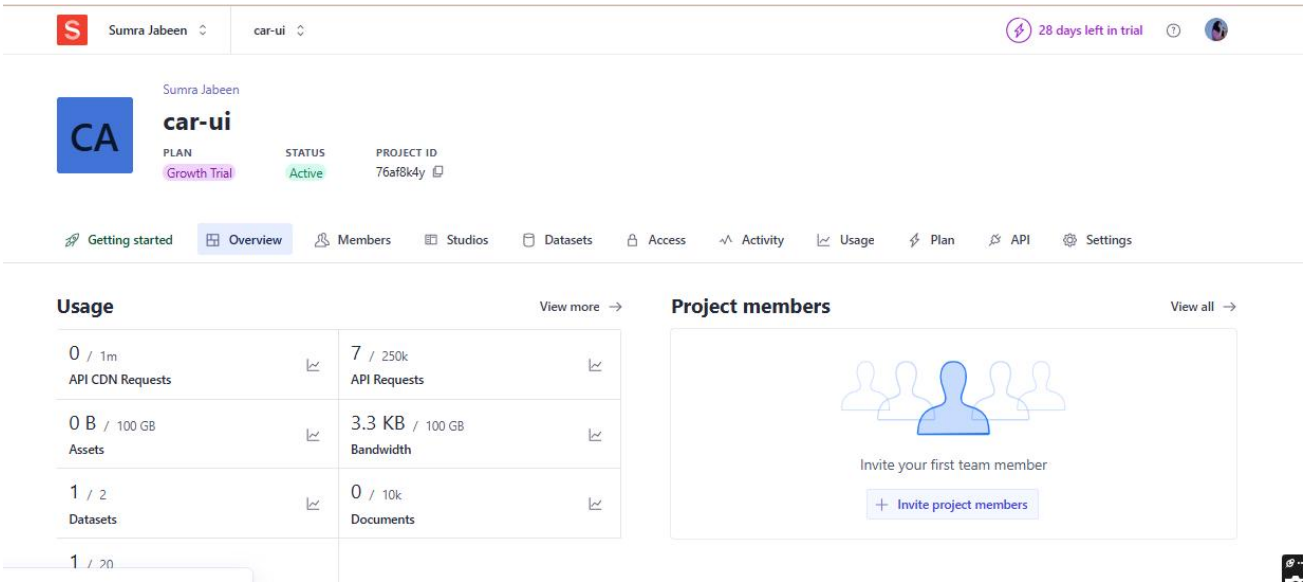
3. Obtaining Sanity Project ID and API Token

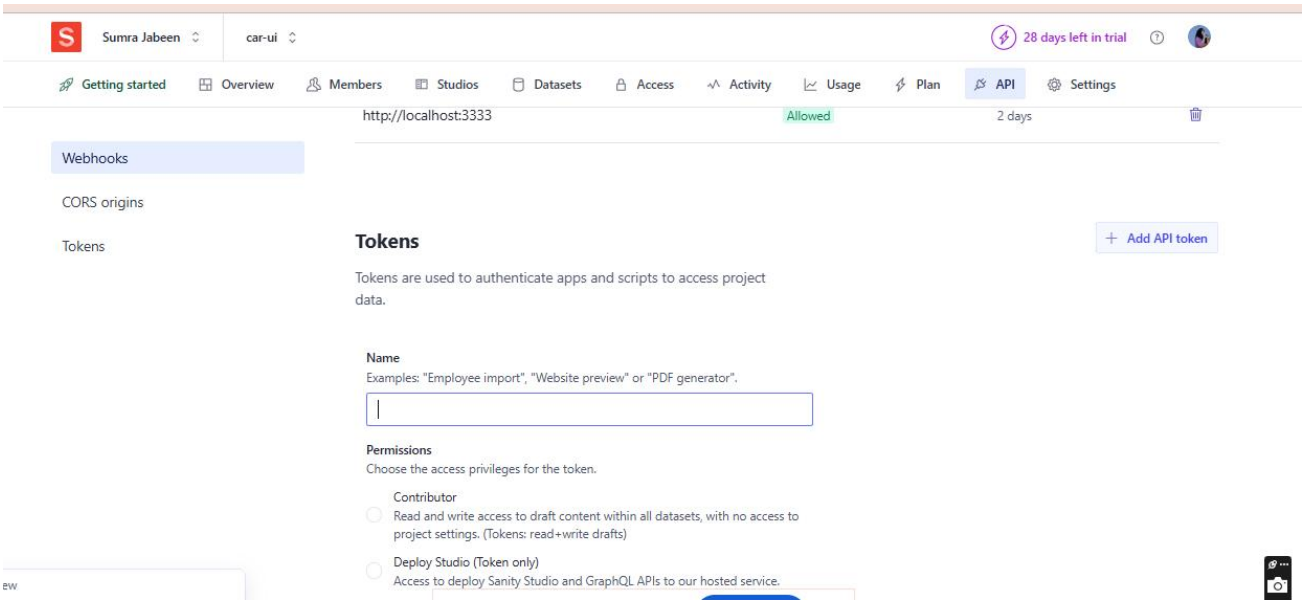
Create a Sanity Project:

- Go to Sanity.io and create a new project.
- Note down your Project ID.

Generate an API Token:

Navigate to Settings > API in your Sanity project dashboard
Generate a new token with read and write permissions.
Add this token to your .env.local file as shown above.





4. Creating the Sanity Schema

Create a cars.ts Schema:

In your Sanity project's schemas directory, create a file named cars.ts. Define your schema as follows:

```ts
import { type SchemaTypeDefinition } from 'sanity'
import cars from './cars'

export const schema: { types: SchemaTypeDefinition[] } = {
  types: [cars],
}
```

```ts
import { apiVersion, dataset, projectId } from '../env'

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  token: process.env.SANITY_API_TOKEN,
  useCdn: true, // Set to false if statically generating pages, using ISR or tag-based revalidation
})
```

```ts
export default {
    name: 'cars',
    title: 'car',
    type: 'document',
    fields: [
      {
        name: 'id',
        title: 'ID',
        type: 'number',
      },
      {
        name: 'name',
        title: 'Name',
        type: 'string',
      },

      {
        name: 'type',
        title: 'Type',
        type: 'string',
      },
      {
        name: 'fuel_capacity',
        title: 'Fuel Capacity',
        type: 'string',
      },
      {
        name: 'transmission',
        title: 'Transmission',
        type: 'string',
      },
      {
        name: 'seating_capacity',
        title: 'Seating Capacity',
        type: 'string',
      },
      {
        name: 'price_per_day',
        title: 'Price Per Day',
        type: 'string',
      },
      {
        name: 'original_price',
        title: 'Original Price',
        type: 'string',
      },
      {
        name: 'image',
        title: 'Car Image',
        type: 'image',

      },
      {
        name: 'tags',
        title: 'Tags',
        type: 'array',
        of: [{ type: 'string' }],
      },

    ],
  };
```

Import the Schema into sanity.mjs:

In the root of your project, create a file named sanity.mjs if it doesn't exist.

Import and add the schema:

5. Setting Up the Data Import Script

Create an Import Script:

Create a file named importsanityData.mjs in your project root and add the following:

```
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;
  }
}
```

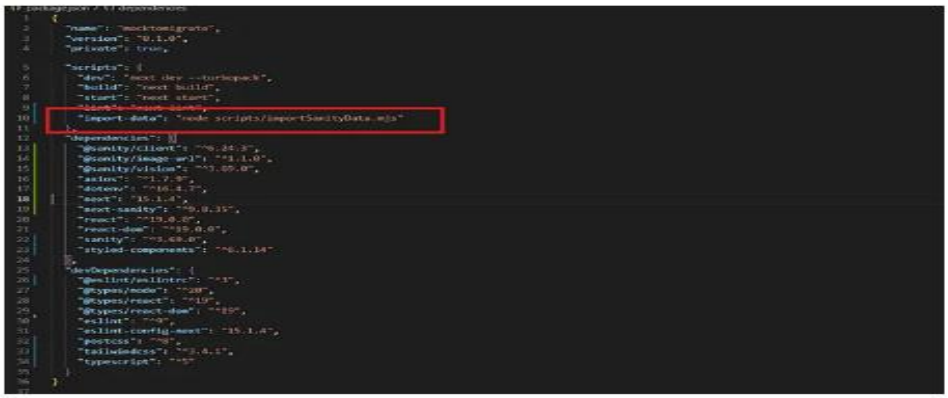
```
async function importData() {
  try {
    console.log("Fetching products from API...");
    const response = await axios.get(
      "https://67820232c51d092c3dcdf34f.mockapi.io/carrental/cars"
    );
    const products = response.data;
    console.log(`Fetched ${products.length} products`);

    for (const product of products) {
      console.log(`Processing product: ${product.name}`);
      console.log("API product data:", JSON.stringify(product, null, 2));

      let imageRef = null;
      if (product.image_url) {
        imageRef = await uploadImageToSanity(product.image_url);
      }

      const sanityProduct = {
        _type: "cars",
        id: product.id,
        name: product.name,
        type: product.type,
        fuel_capacity: product.fuel_capacity,
        transmission: product.transmission,
        seating_capacity: product.seating_capacity,
        price_per_day: product.price_per_day,
        original_price: product.original_price,
        tags: product.tags || [],
        image: imageRef
          ? {
              _type: "image",
              asset: {
                _type: "reference",
                _ref: imageRef,
              },
            }
          : undefined,
      };

      console.log(
        "Sanity product data:",
        JSON.stringify(sanityProduct, null, 2)
      );

      try {
        console.log("Uploading product to Sanity:", sanityProduct.name);
```

```
      try {
        console.log("Uploading product to Sanity:", sanityProduct.name);
        const result = await client.create(sanityProduct);
        console.log(`Product uploaded successfully: ${result._id}`);
      } catch (error) {
        console.error("Error creating product in Sanity:", error);
        console.error(
          "Failed product data:",
          JSON.stringify(sanityProduct, null, 2)
        );
      }
    }

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

importData();
```

6. Fetching Data Locally with GROQ

```
npm install @sanity/client axios dotenv
```

5. Running the Import Script



To run the import script, we need to add a new script to our `package.json` file. Open your `package.json` and add the following to the `"scripts"` section:

Run the Query in Your Project:

Integrate this query into your local project (e.g., within a React or Next.js app) to display the fetched car data.

Test on Localhost:

Start your development server and verify that the data is being fetched and displayed correctly.

```tsx
const CarList: React.FC = () => {
  const [cars, setCars] = useState<CarCardProps[]>([])

  useEffect(() => {
    const fetchCars = async () => {
      const carsData = await client.fetch(`*[_type == "cars"]{
        name,
        type,
        "image": image.asset->url,
        "fuelCapacity": fuel_capacity,
        "transmission": transmission,
        "capacity": seating_capacity,
        "price": price_per_day,
        "discountedPrice": original_price,
        "isFavorite": false
      }`)
      setCars(carsData)
    }
    fetchCars()
  }, [])

  return (
    <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
      {cars.map((car) => (
        <CarCard key={car.name} {...car} />
      ))}
    </div>
  )
}
```

```ts
// export default nextConfig;
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'cdn.sanity.io',
      },
    ],
    },
};

export default nextConfig;
```

Conclusion

By following these steps, you have successfully:

Created a mock API for your rental car data.

Set up a clean Sanity CMS project.

Defined and imported a custom schema for cars.

Imported external mock API data into Sanity.

Fetched and displayed the data locally using GROQ queries.

This workflow ensures a seamless connection between your mock API and Sanity, allowing you to manage and display your data effectively.

Result