

# 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:

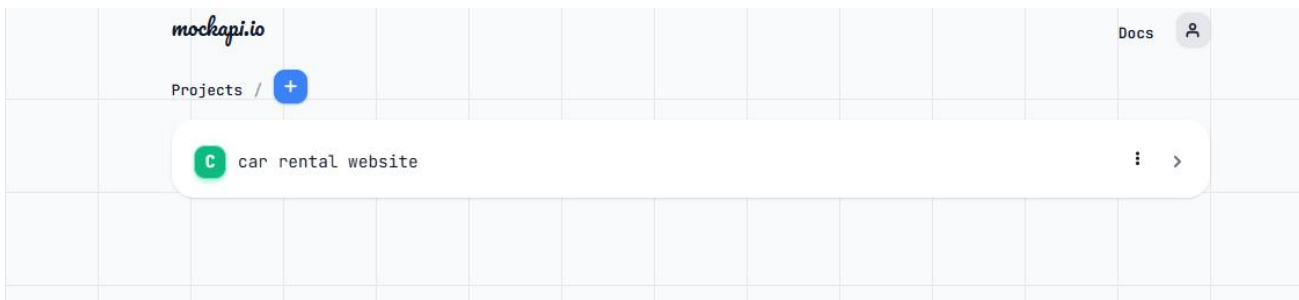
Create a mock API that represents your rental car data. Each car object should include attributes like id, make, model, year, price, and any other relevant details.

#### 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.



Pretty print ☐

```
{
  "id": 1,
  "name": "Koenigsegg",
  "type": "Sport",
  "fuel_capacity": "90L",
  "transmission": "Manual",
  "seating_capacity": "2",
  "People": 2,
  "price_per_day": "$99.00",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar.11698147.jpg&w=640&q=75",
  "tags": ["popular"],
  "id": 2,
  "name": "Nissan GT-R",
  "type": "Sport",
  "fuel_capacity": "80L",
  "transmission": "Manual",
  "seating_capacity": "2",
  "People": 2,
  "price_per_day": "$80.00",
  "original_price": "$100.00",
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(1).cab606a9.jpg&w=640&q=75",
  "tags": ["popular"],
  "id": 3,
  "name": "Rolls-Royce",
  "type": "Sedan",
  "fuel_capacity": "70L",
  "transmission": "Manual",
  "seating_capacity": "4",
  "People": 4,
  "price_per_day": "$96.00",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(2).bd07489a.jpg&w=1200&q=75",
  "tags": ["popular"],
  "id": 4,
  "name": "Nissan GT-R",
  "type": "Sport",
  "fuel_capacity": "80L",
  "transmission": "Manual",
  "seating_capacity": "2",
  "People": 2,
  "price_per_day": "$80.00",
  "original_price": "$100.00",
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(1).cab606a9.jpg&w=1200&q=75",
  "tags": ["popular"],
  "id": 5,
  "name": "Tesla Model 3",
  "type": "Electric",
  "fuel_capacity": "100kWh",
  "transmission": "Manual",
  "seating_capacity": "5",
  "seats": 5,
  "price_per_day": "$100.00/day",
  "original_price": "$100.00",
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(13).37182fc4.jpg&w=1200&q=75",
  "tags": ["recommended"],
  "id": 6,
  "name": "Ford Mustang",
  "type": "Gasoline",
  "fuel_capacity": "60L",
  "transmission": "Manual",
  "seating_capacity": "4",
  "seats": 4,
  "price_per_day": "$80.00/day",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(14).5f4e5799.jpg&w=1200&q=75",
  "tags": ["recommended"],
  "id": 7,
  "name": "BMW X5",
  "type": "Diesel",
  "fuel_capacity": "70L",
  "transmission": "Manual",
  "seating_capacity": "7",
  "seats": 7,
  "price_per_day": "$150.00/day",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(15).5f4e5799.jpg&w=1200&q=75",
  "tags": ["recommended"],
  "id": 8,
  "name": "Audi A6",
  "type": "Hybrid",
  "fuel_capacity": "50L",
  "transmission": "Manual",
  "seating_capacity": "5",
  "seats": 5,
  "price_per_day": "$120.00/day",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(16).fc285c8d.jpg&w=1200&q=75",
  "tags": ["recommended"],
  "id": 9,
  "name": "Mercedes-Benz C-Class",
  "type": "Gasoline",
  "fuel_capacity": "65L",
  "transmission": "Manual",
  "seating_capacity": "5",
  "seats": 5,
  "price_per_day": "$140.00/day",
  "original_price": null,
  "image_url": "https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fcar(17).574834dc.jpg&w=1200&q=75",
  "tags": ["recommended"],
  "id": 10,
  "name": "name 10",
  "type": "type 10",
  "fuel_capacity": "fuel_capacity 10",
  "transmission": "transmission 10",
  "seating_capacity": "seating_capacity 10",
  "price_per_day": "price_per_day 10",
  "original_price": "original_price 10",
  "image_url": "image_url 10",
  "tags": "tags 10"
}
```

## 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="skCdN405jrBgxuc70yRfEh8g2ai81IjMzInxCC7czbAJM1jw4fxwy0Tdgw8CEoLXDpj8tIdvw4rLJUNCrhytILJ69ko6aAm9y3ycjTGEohI28eMsZ5GUvmp6Pk3ywM11juv5IGQRLMRwBoNPXhSIg0964jjMNxBKXsyWXPJBgAshNx3tA5"SS
```

## 3. Obtaining Sanity Project ID and API Token


### Create a Sanity Project:

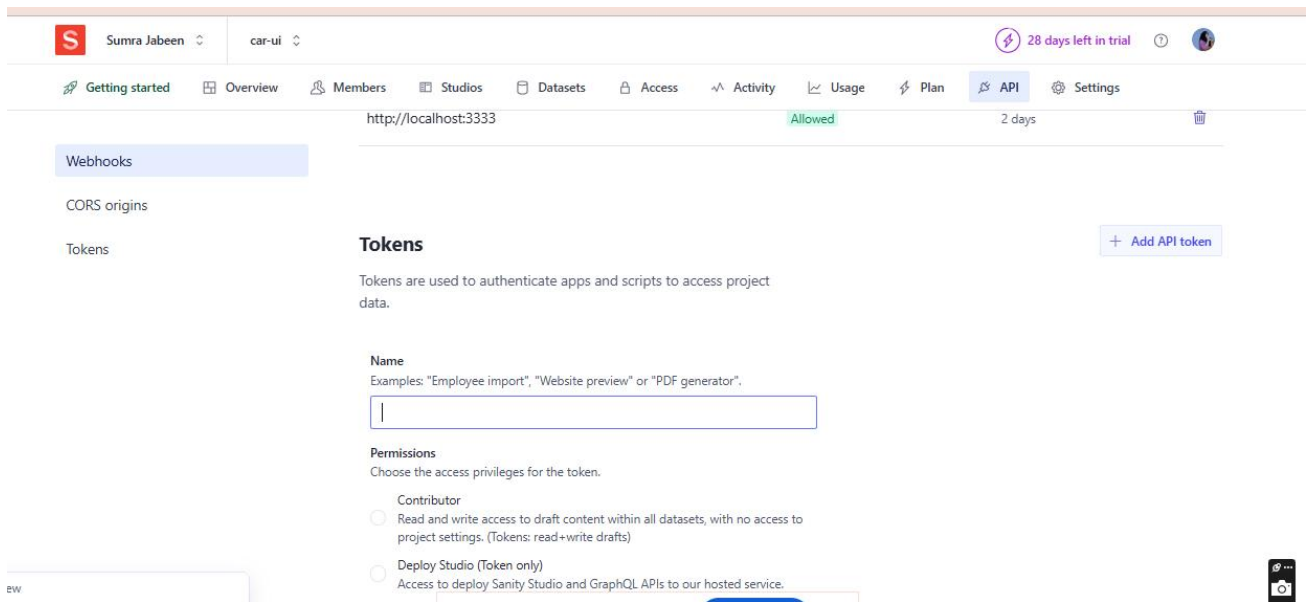
- Go to [Sanity.io](https://sanity.io) and create a new project.
- Note down your Project ID.

### 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.

The screenshot shows the Sanity project dashboard for a project named 'car-ui'. The top navigation bar includes the user 'Sumra Jabeen', the project name 'car-ui', and a trial status '28 days left in trial'. Below the header, the project details are displayed: a blue 'CA' logo, the project name 'car-ui', the plan 'Growth Trial', the status 'Active', and the project ID '76af8k4y'. A navigation menu at the bottom includes links for 'Getting started', 'Overview' (selected), 'Members', 'Studios', 'Datasets', 'Access', 'Activity', 'Usage', 'Plan', 'API', and 'Settings'. The main content area is divided into two sections: 'Usage' and 'Project members'. The 'Usage' section shows a table with metrics for API CDN Requests, Assets, Datasets, API Requests, Bandwidth, and Documents. The 'Project members' section shows a placeholder for team members with a button to 'Invite project members'.

Usage		Project members	
0 / 1m API CDN Requests	7 / 250k API Requests	 Invite your first team member <a href="#">+ Invite project members</a>	
0 B / 100 GB Assets	3.3 KB / 100 GB Bandwidth		
1 / 2 Datasets	0 / 10k Documents		



## 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:

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

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

```
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
})
```

```
1 export default {
2   name: 'cars',
3   title: 'car',
4   type: 'document',
5   fields: [
6     {
7       name: 'id',
8       title: 'ID',
9       type: 'number',
10    },
11    {
12      name: 'name',
13      title: 'Name',
14      type: 'string',
15    },
16    {
17      name: 'type',
18      title: 'Type',
19      type: 'string',
20    },
21    {
22      name: 'fuel_capacity',
23      title: 'Fuel Capacity',
24      type: 'string',
25    },
26    {
27      name: 'transmission',
28      title: 'Transmission',
29      type: 'string',
30    },
31    {
32      name: 'seating_capacity',
33      title: 'Seating Capacity',
34      type: 'string',
35    },
36    {
37      name: 'price_per_day',
38      title: 'Price Per Day',
39      type: 'string',
40    },
41    {
42      name: 'original_price',
43      title: 'Original Price',
44      type: 'string',
45    },
46    {
47      name: 'image',
48      title: 'Car Image',
49      type: 'image',
50    },
51    {
52      name: 'tags',
53      title: 'Tags',
54      type: 'array',
55      of: [{ type: 'string' }],
56    },
57  ],
58 },
59 ],
60 };
61
62
```

## 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:

```
1  import { createClient } from "@sanity/client";
2  import axios from "axios";
3  import dotenv from "dotenv";
4  import { fileURLToPath } from "url";
5  import path from "path";
6
7  // Load environment variables from .env.local
8  const __filename = fileURLToPath(import.meta.url);
9  const __dirname = path.dirname(__filename);
10  dotenv.config({ path: path.resolve(__dirname, "../.env.local") });
11  // Create Sanity client
12  const client = createClient({
13    projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
14    dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
15    useCdn: false,
16    token: process.env.SANITY_API_TOKEN,
17    apiVersion: "2021-08-31",
18  });
19  async function uploadImageToSanity(imageUrl) {
20    try {
21      console.log(`Uploading image: ${imageUrl}`);
22      const response = await axios.get(imageUrl, { responseType: "arraybuffer" });
23      const buffer = Buffer.from(response.data);
24      const asset = await client.assets.upload("image", buffer, {
25        filename: imageUrl.split("/").pop(),
26      });
27      console.log(`Image uploaded successfully: ${asset._id}`);
28      return asset._id;
29    } catch (error) {
30      console.error("Failed to upload image:", imageUrl, error);
31      return null;
32    }
33  }
```

```

46 async function importData() {
47   try {
48     console.log("Fetching products from API...");
49     const response = await axios.get(
50       "https://67820232c51d092c3dcdf34f.mockapi.io/carrental/cars"
51     );
52     const products = response.data;
53     console.log(`Fetched ${products.length} products`);
54
55     for (const product of products) {
56       console.log(`Processing product: ${product.name}`);
57       console.log("API product data:", JSON.stringify(product, null, 2));
58
59       let imageRef = null;
60       if (product.image_url) {
61         imageRef = await uploadImageToSanity(product.image_url);
62       }
63
64       const sanityProduct = {
65         _type: "cars",
66         id: product.id,
67         name: product.name,
68         type: product.type,
69         fuel_capacity: product.fuel_capacity,
70         transmission: product.transmission,
71         seating_capacity: product.seating_capacity,
72         price_per_day: product.price_per_day,
73         original_price: product.original_price,
74         tags: product.tags || [],
75         image: imageRef
76         ? {
77           _type: "image",
78           asset: {
79             _type: "reference",
80             _ref: imageRef,
81           },
82         }
83         : undefined,
84       };
85
86       console.log(
87         "Sanity product data:",
88         JSON.stringify(sanityProduct, null, 2)
89       );
90
91       try {
92         console.log("Uploading product to Sanity:", sanityProduct.name);

```



```

90
91     try {
92         console.log("Uploading product to Sanity:", sanityProduct.name);
93         const result = await client.create(sanityProduct);
94         console.log(`Product uploaded successfully: ${result._id}`);
95     } catch (error) {
96         console.error("Error creating product in Sanity:", error);
97         console.error(
98             "Failed product data:",
99             JSON.stringify(sanityProduct, null, 2)
100         );
101     }
102 }
103
104 console.log("Data import completed successfully!");
105 } catch (error) {
106     console.error("Error importing data:", error);
107 }
108 }
109
110 importData();
111

```

## 6. Fetching Data Locally with GROQ

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

### 5. Running the Import Script

```

1  {
2    "name": "next-sanity",
3    "version": "0.1.0",
4    "private": true,
5    "scripts": {
6      "dev": "next dev --turbo",
7      "build": "next build",
8      "start": "next start",
9      "lint": "eslint .",
10     "import-data": "node scripts/importSanityData.js"
11   },
12   "dependencies": {
13     "@sanity/client": "^4.25.2",
14     "@sanity/image-url": "^1.1.0",
15     "@sanity/preview-kit": "^3.0.0",
16     "axios": "^1.7.0",
17     "dotenv": "^16.4.7",
18     "next": "15.1.4",
19     "next-sanity": "^9.0.35",
20     "react": "19.0.0",
21     "react-dom": "19.0.0",
22     "sanity": "^3.69.0",
23     "styled-components": "^6.1.14"
24   },
25   "devDependencies": {
26     "@types/node": "20",
27     "@types/react": "19",
28     "@types/react-dom": "19",
29     "eslint": "8",
30     "eslint-config-next": "15.1.4",
31     "postcss": "8",
32     "tailwindcss": "3.4.17",
33     "typescript": "5"
34   }
35 }

```

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.

```
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>
  )
}
```

```
// 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.

Fetches and displays 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

