

Hackathon: Day 3

API Integration & Data Migration.

- **Let me tell you what we did in this task.**

1.Sanity Setup: You define schemas in Sanity Studio to structure your data (e.g., products with title, price, description, image, etc.).

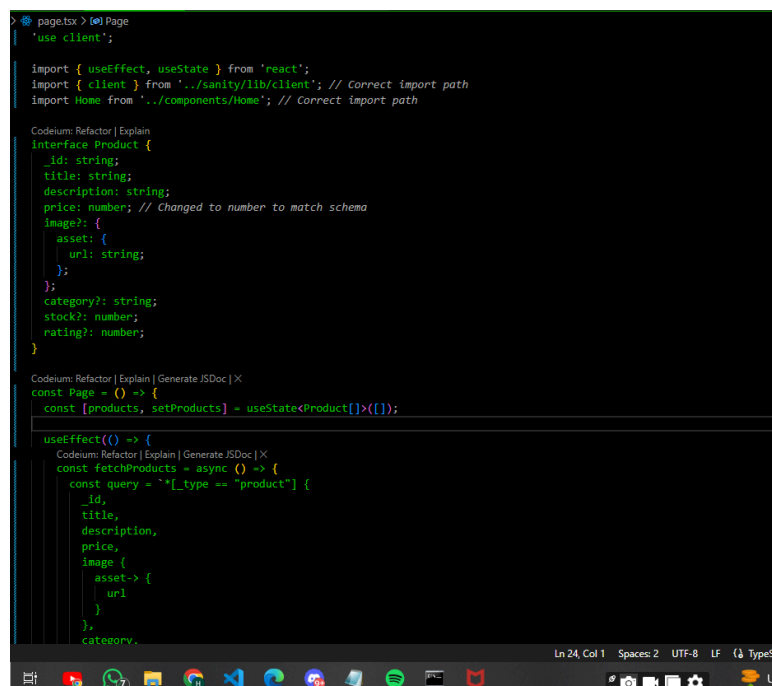
2.Data Entry: Populate the fields in Sanity Studio manually or via migration scripts.

3.API Integration: Use GROQ queries to fetch data from Sanity's backend. These queries retrieve specific fields like product details.

4.Frontend Display: The fetched data is passed to your React/Next.js components to render on the UI dynamically.

- **API Integration Process.**

For this project, I integrated Sanity CMS to manage product data dynamically. Instead of using external APIs for data migration, I utilized Sanity's intuitive interface to manually populate the required fields. GROQ queries were used to fetch the data into the application.



```

> page.tsx > Page
| 'use client';

import { useEffect, useState } from 'react';
import { client } from '../sanity/lib/client'; // Correct import path
import Home from '../components/Home'; // Correct import path

Codeium: Refactor | Explain
interface Product {
  _id: string;
  title: string;
  description: string;
  price: number; // Changed to number to match schema
  image?: {
    asset: {
      url: string;
    };
  };
  category?: string;
  stock?: number;
  rating?: number;
}

Codeium: Refactor | Explain | Generate JSDoc | X
const Page = () => {
  const [products, setProducts] = useState<Product[]>([]);

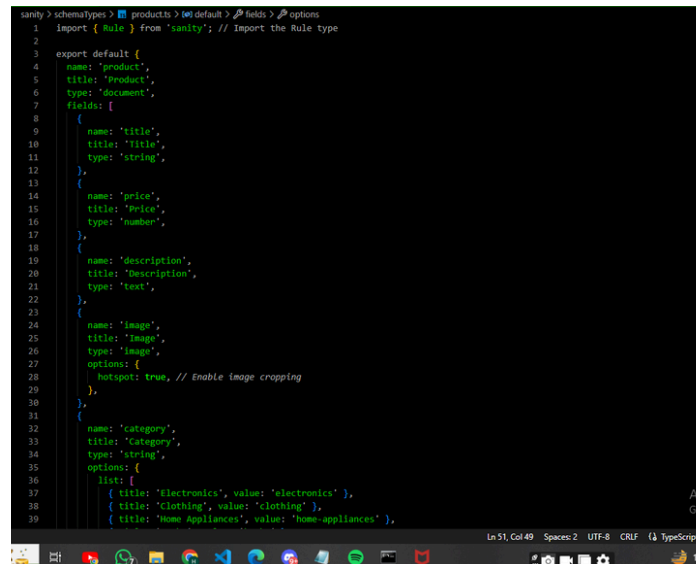
  useEffect(() => {
    Codeium: Refactor | Explain | Generate JSDoc | X
    const fetchProducts = async () => {
      const query = `*_type == "product" {
        _id,
        title,
        description,
        price,
        image {
          asset-> {
            url
          }
        },
        category,

```

● Schema Adjustments

I created and adjusted the following schema in Sanity Studio:

- **Product Schema:** Included fields for title, description, price, image, category, stock, and rating.



```
sanity > schematypes > products > default > fields > options
1 import { Rule } from 'sanity'; // Import the Rule type
2
3 export default {
4   name: 'product',
5   title: 'Product',
6   type: 'document',
7   fields: [
8     {
9       name: 'title',
10      title: 'Title',
11      type: 'string',
12    },
13    {
14      name: 'price',
15      title: 'Price',
16      type: 'number',
17    },
18    {
19      name: 'description',
20      title: 'Description',
21      type: 'text',
22    },
23    {
24      name: 'image',
25      title: 'Image',
26      type: 'image',
27      options: {
28        hotspot: true, // Enable image cropping
29      },
30    },
31    {
32      name: 'category',
33      title: 'Category',
34      type: 'string',
35      options: {
36        list: [
37          { title: 'Electronics', value: 'electronics' },
38          { title: 'Clothing', value: 'clothing' },
39          { title: 'Home Appliances', value: 'home-appliances' },
40        ],
41      },
42    },
43  ],
44 }
```

● Migration Steps:

1. Sanity Studio:

- Manually added data entries in Sanity Studio for each product.
- Verified populated fields using the "Content" section in the Studio.

```
1: {...} 5 properties
  price: 2000
  ▼ image: {...} 1 property
    ▼ asset: {...} 1 property
      url: https://cdn.sanity.io/images/vcqvpo7/production/2ac413c1917ac944454e358f73d0af67e1f1b74e-4024x3270.png
      _id: 4f8fd369-2b23-4c76-b1ad-ad999142ed98
    title: sofa
    description: a comfy sofa
```

● Front-end Integration

In the frontend integration section, you'll explain how you fetched data from Sanity CMS and displayed it in your frontend using React/Next.js. This includes:

1. **Fetching Data:** Using GROQ queries to fetch product data from Sanity.
2. **Displaying Data:** Passing the fetched data to components (e.g., Home) and rendering it in the UI.

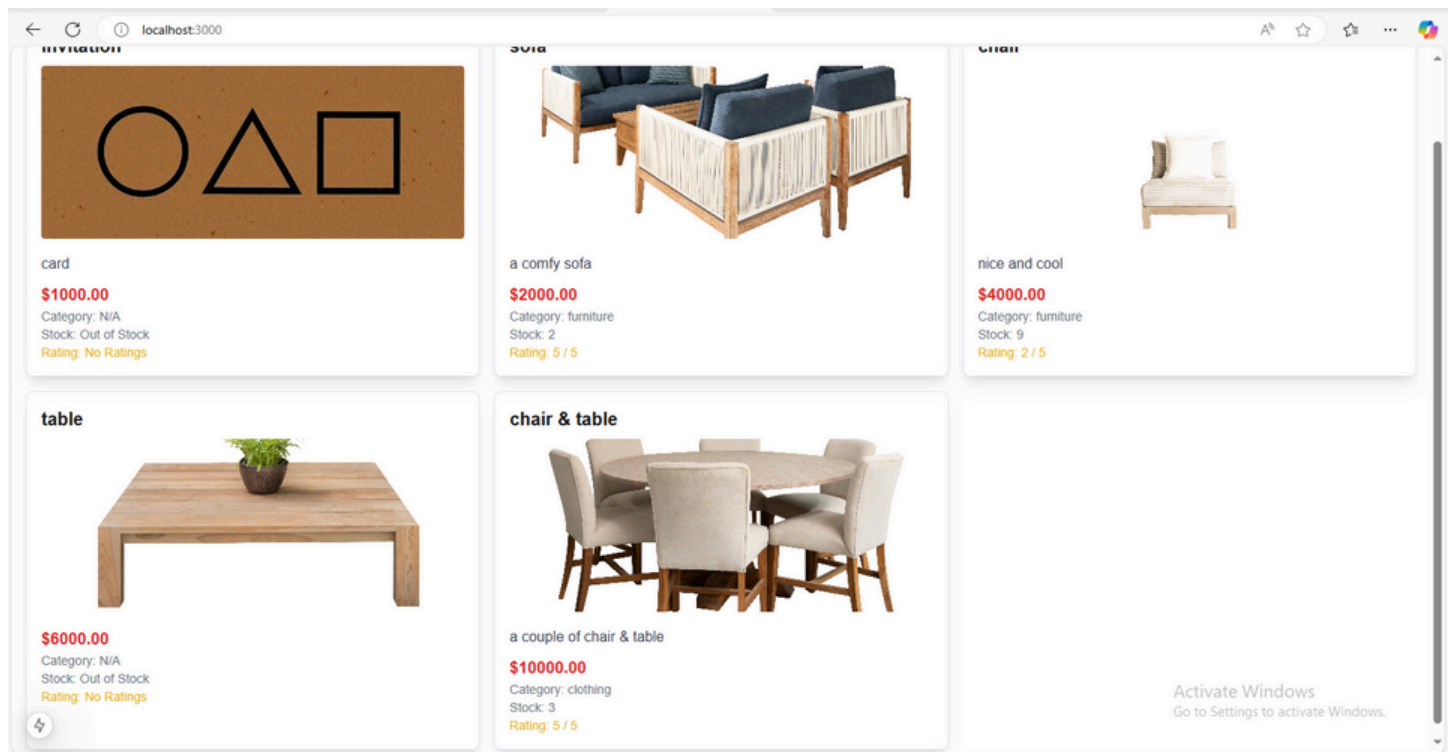
3. **Code Snippets:** Provide code examples like how you used `client.fetch()` to retrieve data and how it was rendered.

```
Codeium: Refactor | Explain | Generate JSDoc | ✕
const Page = () => {
  const [products, setProducts] = useState<Product[]>([]);

  useEffect(() => {
    Codeium: Refactor | Explain | Generate JSDoc | ✕
    const fetchProducts = async () => {
      const query = `*[_type == "product"] {
        _id,
        title,
        description,
        price,
        image {
          asset-> {
            url
          }
        },
        category,
        stock,
        rating
      }`;
    };
  });
}
```

- **Front-end display Data**

This is how it shows on front-end



- Sanity Studio Fields

Title: The name of the product (e.g., "Sofa").

Description: A short description of the product (e.g., "A comfy sofa").

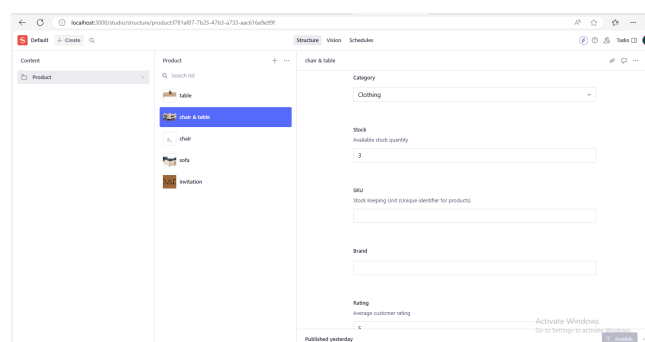
Price: The cost of the product (e.g., 2000).

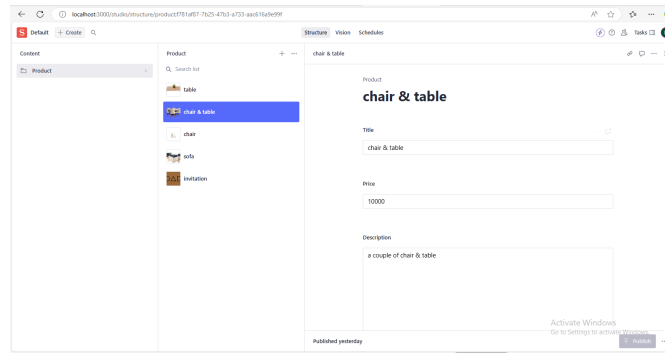
Image: The product image URL (e.g., URL of the sofa image).

Category: The category the product belongs to (optional).

Stock: Quantity of the product in stock (optional).

Rating: Rating of the product (optional).





● Closing Remarks

At the end, the process of integrating Sanity CMS with a React/Next.js project involves setting up the schema, adding products manually through Sanity Studio, and using GROQ queries to fetch the data. The data is then displayed on the frontend, and the integration ensures the seamless flow of product details like title, description, price, and images. This integration simplifies content management and allows for dynamic updates through the CMS, enhancing the flexibility and scalability of the marketplace.

● Available API Endpoints:

1. Fetch All Products

- **Endpoint:** /product
- **Method:** GET
- **Description:** Retrieves a list of all products from the Sanity CMS.

2. Fetch Single Product

- **Endpoint:** /product/:id
- **Method:** GET
- **Description:** Retrieves a specific product by its ID.

3. Create Product

- **Endpoint:** /product
- **Method:** POST
- **Description:** Adds a new product to the CMS.

4. Update Product

- **Endpoint:** /product/:id
- **Method:** PUT
- **Description:** Updates an existing product by its ID.