

**Name: Noor Ul Sehar**

**Roll Number : 00349943**

**Class : Friday ( 9 am to 12 pm )**

## **Day 3 - API Integration and Data Migration**

### **Overview:**

Day 3 was dedicated to integrating APIs, particularly with Sanity CMS, utilizing GROQ queries, and refining the schema to accommodate dynamic data needs. Below is a breakdown of the tasks completed:

### **Key Objectives:**

1. Establish a connection with Sanity CMS to manage product data effectively.
2. Implement GROQ queries to dynamically retrieve structured data from Sanity.
3. Update the Sanity schema to meet the evolving requirements of the API.
4. Migrate product data, including images, from an external API to Sanity CMS.
5. Test and validate the integration to ensure seamless data flow between the backend and frontend.

# 1. Adjustments made to schemas

Align the Sanity CMS schema with the application's dynamic data requirements, several adjustments were made to the product schema. These changes ensure the schema can handle additional fields, validations, and data structures effectively. Below are the key adjustments:

## Products Schema :

The **product schema** defines product data structure with fields like id, name, brand, category, price, stock, image, and description, ensuring organized and validated data storage.

## Product Schema Details :

```
export const productSchema = {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'productName',
      title: 'Product Name',
      type: 'string',
    },
    {
      name: 'category',
      title: 'Category',
      type: 'string',
    },
    {
      name: 'price',
      title: 'Price',
      type: 'number',
    },
    {
      name: 'inventory',
      title: 'Inventory',
      type: 'number',
    },
  ],
}
```

```

    name: 'colors',
    title: 'Colors',
    type: 'array',
    of: [{ type: 'string' }]],
  },
  {
    name: 'status',
    title: 'Status',
    type: 'string',
  },
  {
    name: 'image',
    title: 'Image',
    type: 'image',
    options: {
      hotspot: true,
    },
  },
  {
    name: 'description',
    title: 'Description',
    type: 'text',
  },
],
}

```

## Old Schema:

The old schema contained the following fields:

- ❖ **ID:** A unique identifier for each product.
- ❖ **Name:** The name of the product.
- ❖ **Brand:** The brand of the product.
- ❖ **Category:** The category of the product.
- ❖ **Price:** The price of the product.
- ❖ **Stock:** The available stock of the product.
- ❖ **Description:** A detailed description of the product.

- ❖ **Image:** An image of the product.

## New Schema:

The new schema expands on the old one, with these fields:

- ❖ **ID (product id):** Retained as a unique identifier for each product.
- ❖ **Product Name (product Name):** Renamed from `name` for clarity.
- ❖ **Brand (brand):** Retained for brand-specific filtering.
- ❖ **Category (category):** Retained for product classification.
- ❖ **Price (price):** Retained for product pricing.
- ❖ **Inventory (inventory):** Renamed from `stock` to track available stock.
- ❖ **Colors (colors):** Added to store available color options as an array.
- ❖ **Status (status):** Added to track the product's status (e.g., "available", "out of stock").
- ❖ **Image (image):** Enhanced to use Sanity's built-in image type with hotspot support.
- ❖ **Description (description):** Retained for product details.

## 2. API Integration Process:

- ❖ **Identify API Requirements:**

Understand the API's purpose, endpoints, and data format (e.g., JSON, XML).

- ❖ **Set Up API Credentials:**

Obtain API keys, tokens, or authentication details.

- ❖ **Install Required Libraries:**

Use tools like `axios` (for HTTP requests) or SDKs provided by the API.

❖ **Configure API Client:**

Set up the base URL, headers, and authentication.

❖ **Make API Requests:**

Use GET, POST, PUT, or DELETE methods to interact with the API.

❖ **Handle Responses:**

Process and validate the API response data.

❖ **Error Handling:**

Implement error handling for failed requests or invalid responses.

❖ **Test Integration:**

Verify functionality and ensure seamless data flow.

❖ **Deploy and Monitor:**

Deploy the integration and monitor for performance or errors.

## 3. API Data Migration to Sanity

### Objective:

Migrate product data, including images, from an external API to Sanity CMS.

### Steps Taken:

#### 1. Fetch Product Data:

- ❖ Utilized Axios to retrieve product data from an external API.

- ❖ Example API Endpoint: ``https://template-03-api.vercel.app/api/products``.

## 2. Upload Images to Sanity:

- ❖ Uploaded product images using the ``client.assets.upload`` method.
- ❖ Ensured each image was correctly linked to its respective product.

## 3. Create Product Documents in Sanity:

- ❖ Iterated through the fetched data and created corresponding product documents in Sanity using the ``client.create()`` method.

### ❖ Sanity CMS Integration

#### ➤ Connection Setup:

- Integrated the ``@sanity/client`` library into the project to enable communication with Sanity CMS.
- Configured the client with project-specific credentials, including ``projectId``, ``dataset``, and ``apiVersion``.

### ❖ Purpose:

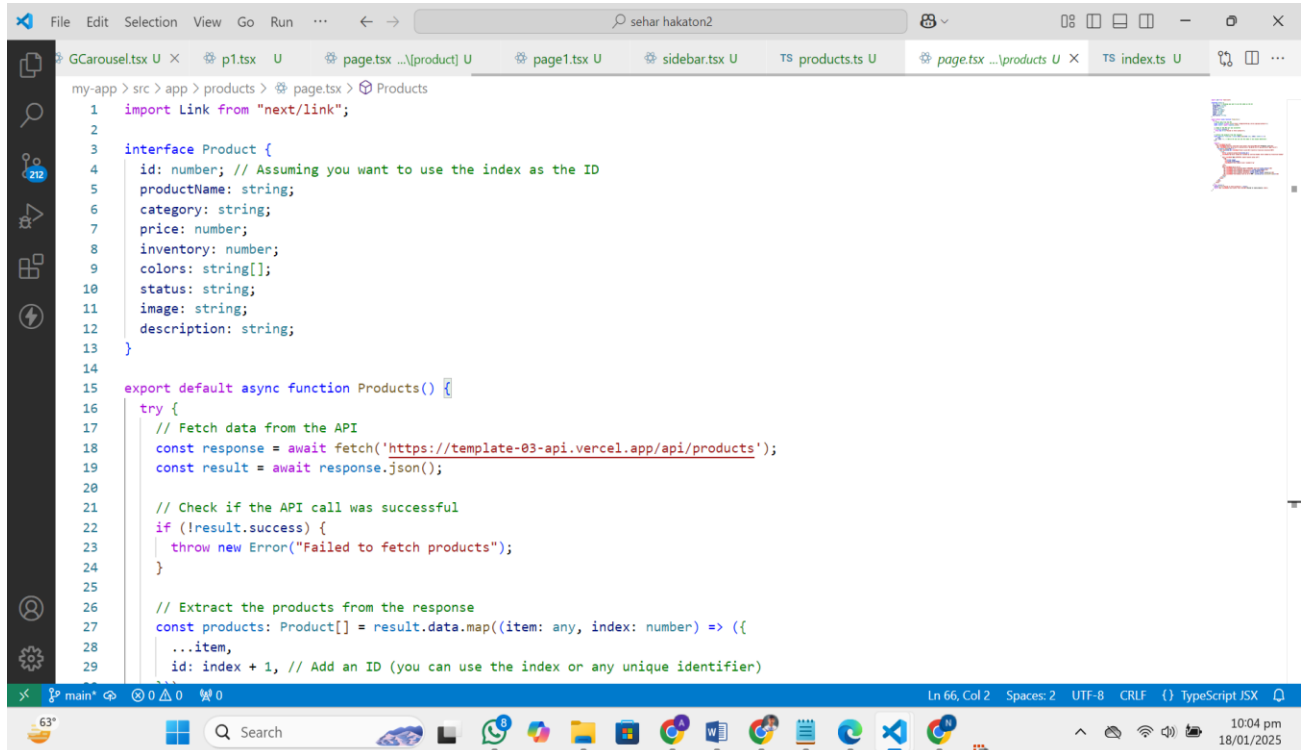
- To centralize product management and leverage Sanity's features, such as real-time updates, robust content modeling, and optimized image handling.

### ❖ Schema Enhancements in Sanity CMS

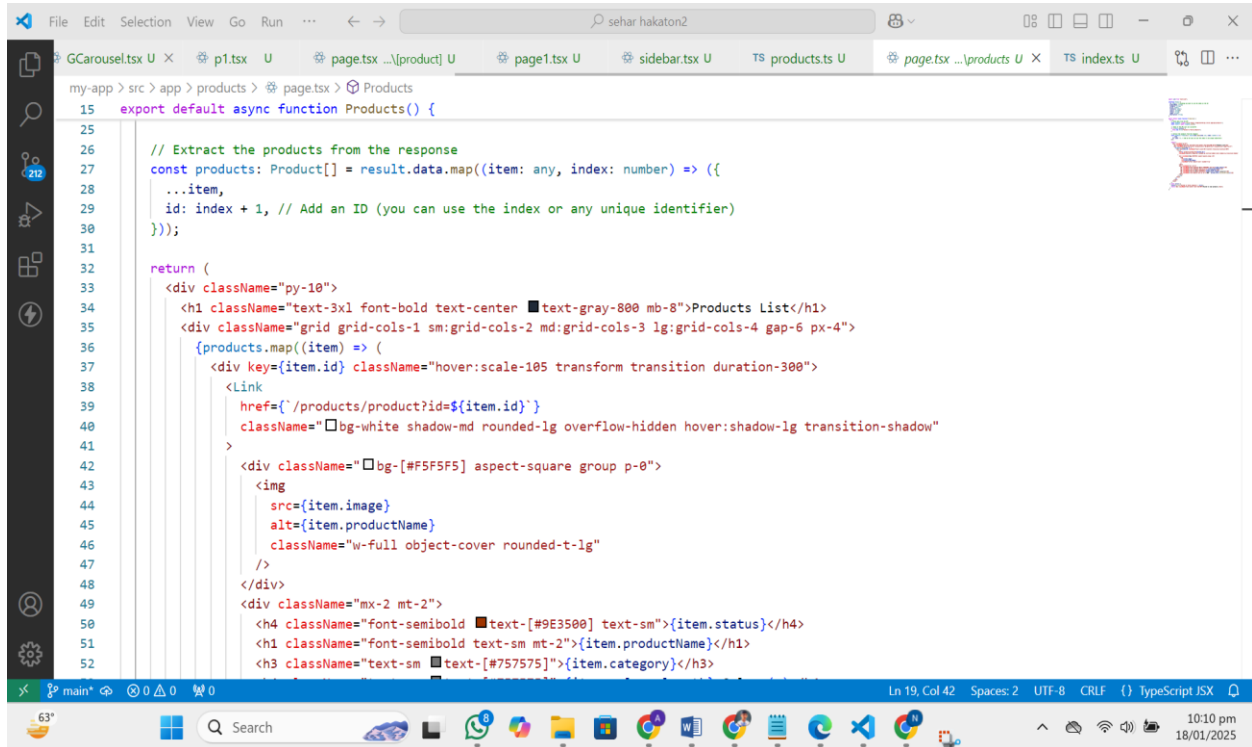
- Updated the product schema to align with the application's requirements, ensuring it could handle new fields and validations.
- Fields Added/Updated:
  - **inventory**: Tracks stock levels for products.
  - **colors**: Stores available color options as an array.

# Screenshots :

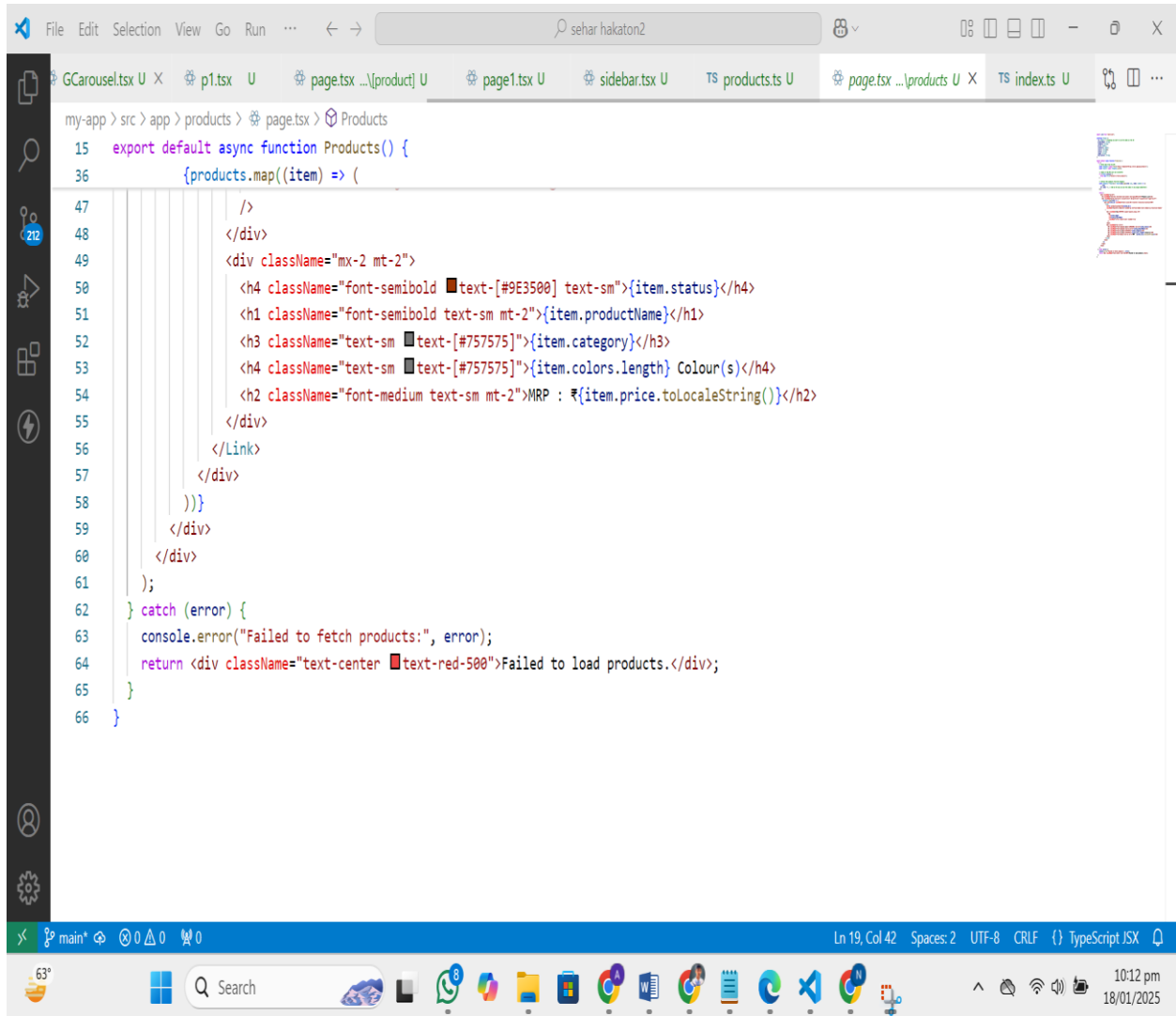
## API calling :



```
1 import Link from "next/link";
2
3 interface Product {
4   id: number; // Assuming you want to use the index as the ID
5   productName: string;
6   category: string;
7   price: number;
8   inventory: number;
9   colors: string[];
10  status: string;
11  image: string;
12  description: string;
13 }
14
15 export default async function Products() {
16   try {
17     // Fetch data from the API
18     const response = await fetch('https://template-03-api.vercel.app/api/products');
19     const result = await response.json();
20
21     // Check if the API call was successful
22     if (!result.success) {
23       throw new Error("Failed to fetch products");
24     }
25
26     // Extract the products from the response
27     const products: Product[] = result.data.map((item: any, index: number) => ({
28       ...item,
29       id: index + 1, // Add an ID (you can use the index or any unique identifier)
30     }));
31   } catch (error) {
32     console.error(error);
33   }
34 }
```



```
15 export default async function Products() {
16   // ... (API call logic from previous screenshot)
17
18   return (
19     <div className="py-10">
20       <h1 className="text-3xl font-bold text-center text-gray-800 mb-8">Products List</h1>
21       <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6 px-4">
22         {products.map((item) => (
23           <div key={item.id} className="hover:scale-105 transform transition duration-300">
24             <Link
25               href={`/products/product?id=${item.id}`}
26               className="bg-white shadow-md rounded-lg overflow-hidden hover:shadow-lg transition-shadow"
27             >
28               <div className="bg-[#F5F5F5] aspect-square group p-0">
29                 <img
30                   src={item.image}
31                   alt={item.productName}
32                   className="w-full object-cover rounded-t-lg"
33                 />
34               </div>
35               <div className="mx-2 mt-2">
36                 <h4 className="font-semibold text-[#9E3500] text-sm">{item.status}</h4>
37                 <h1 className="font-semibold text-sm mt-2">{item.productName}</h1>
38                 <h3 className="text-sm text-[#757575]">{item.category}</h3>
39               </div>
40             </Link>
41           </div>
42         ))}
43       </div>
44     </div>
45   );
46 }
```





Data successfully displayed in the frontend:

Products List



Just In

Nike Air Force 1 Mid '07

Men's Shoes

1 Colour(s)

MRP : ₹10,795



Just In

Nike Court Vision Low Next Nature

Men's Shoes

1 Colour(s)

MRP : ₹4,995



Just In

Nike Air Force 1 PLT.AF.ORM

Women's Shoes

1 Colour(s)

MRP : ₹8,695



Just In

Nike Air Force 1 React

Men's Shoes

1 Colour(s)

MRP : ₹13,295



Promo Exclusion

Air Jordan 4 Elements Low



Just In

Nike Standard Issue Basketball



Promo Exclusion

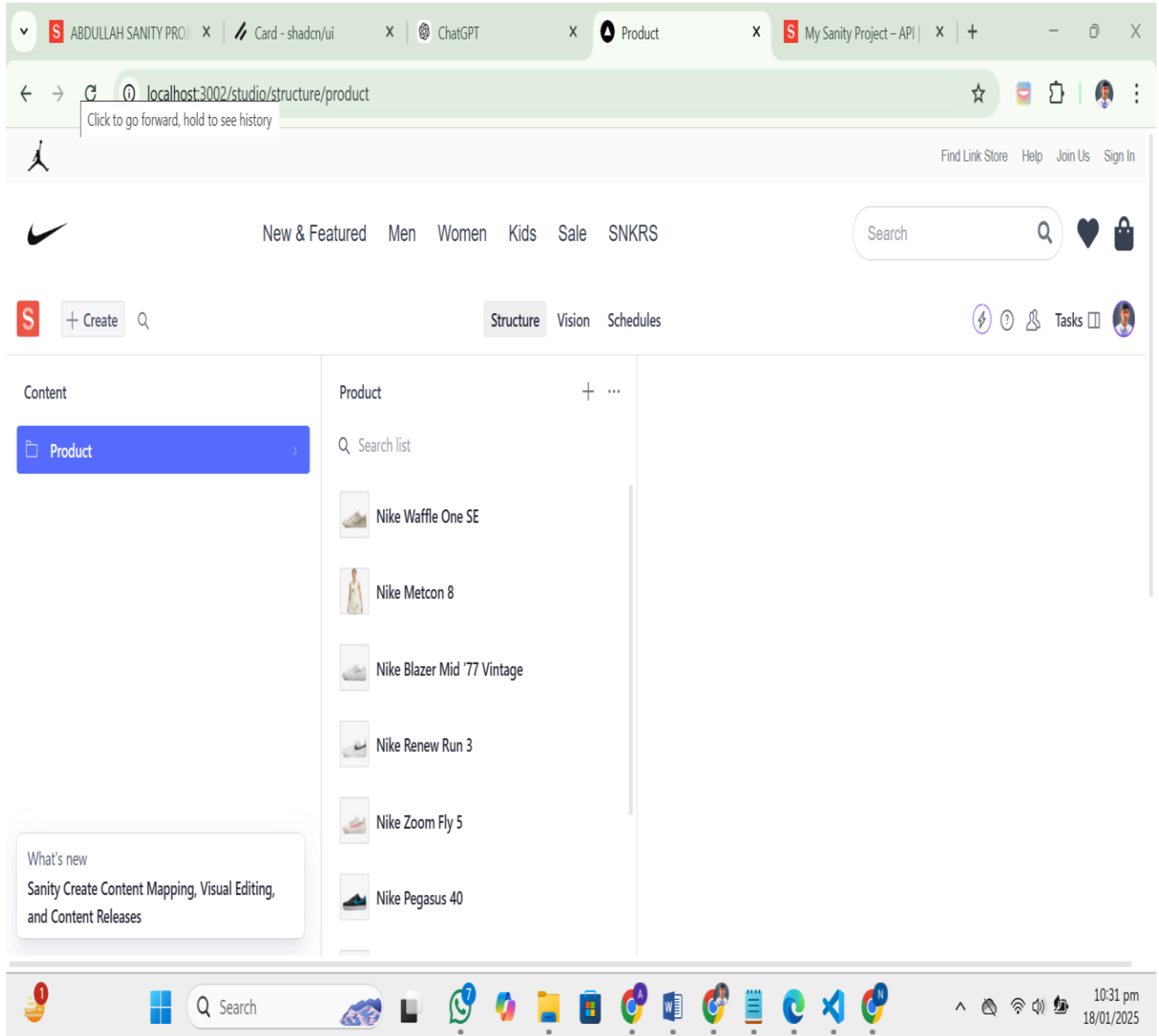
Nike Dunk Low Retro SE



Sustainable Materials

Nike Def Flyknit Running

# Populated Sanity CMS fields:



## Self-Validation Checklist:

➤	API Understanding	✓
➤	Schema Validation	✓
➤	Data Migration	✓
➤	Submission Preparation	✓
➤	API Integration in Next.js	✓

## Outcomes of Day 3:

- Successfully integrated Sanity CMS with the application.
- Implemented dynamic GROQ queries to fetch and display product data.
- Enhanced the Sanity schema to support extended product attributes.
- Migrated data and images from an external API to Sanity CMS.
- Validated and tested the integration to ensure seamless backend-to-frontend data flow.

As I wrap up my Day 3 work at the Hackathon, I, Noor Ul Sehar, am excited to continue pushing forward with determination and creativity to bring these innovative features to life!