DAY 3 - API INTEGRATION AND DATA MIGRATION

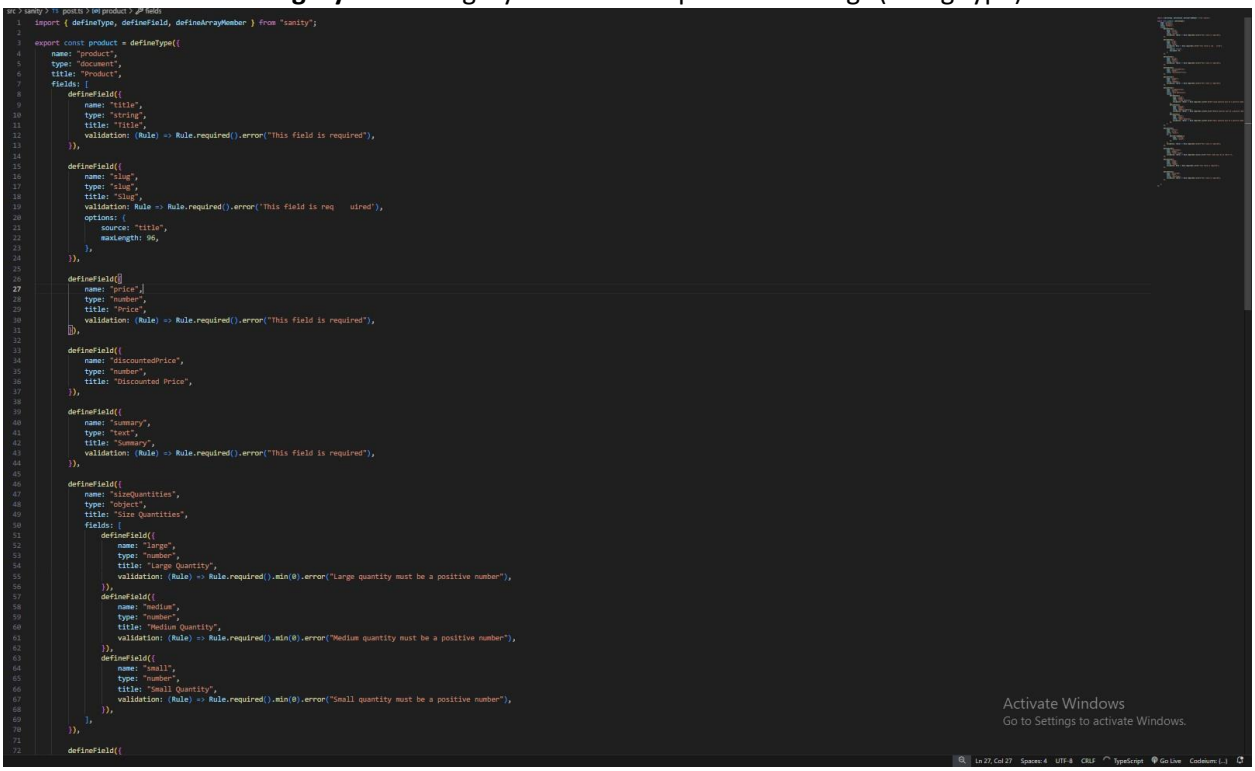
Objective:

The aim for Day 3 was to seamlessly integrate API data into Sanity CMS for the Bandage project. This integration allowed dynamic content updates for the marketplace, replacing the need for manual data entry with a more efficient and scalable solution.

1. Sanity CMS Schema Design:

To efficiently manage product data, I created a schema named **`product`** in Sanity CMS. The schema includes the following fields:

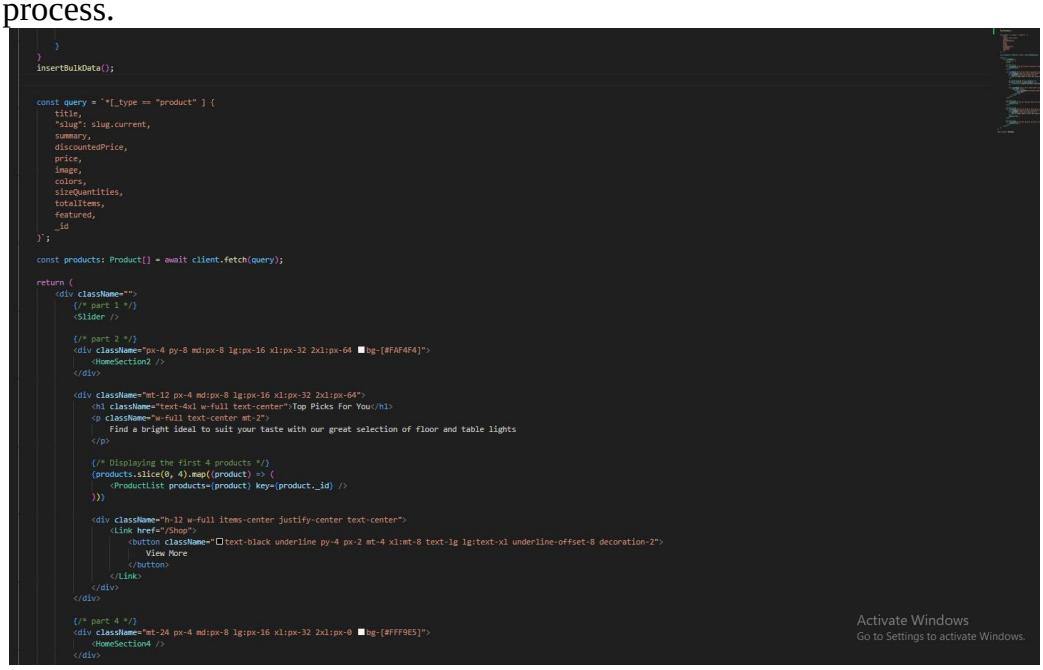
- **Main Fields:** ○ **id**: A unique identifier for the product (string type).
  - **name**: The product's name (string type). ○ **imagePath**: The URL of the product image (URL type).
  - **price**: The price of the product (number type). ○ **description**: A detailed description of the product (text type). ○ **discountPercentage**: The discount percentage applied to the product (number type).
  - **isFeaturedProduct**: A boolean flag indicating if the product is featured (boolean type).
  - **stockLevel**: The quantity of the product available in stock (number type). ○ **category**: The category to which the product belongs (string type).

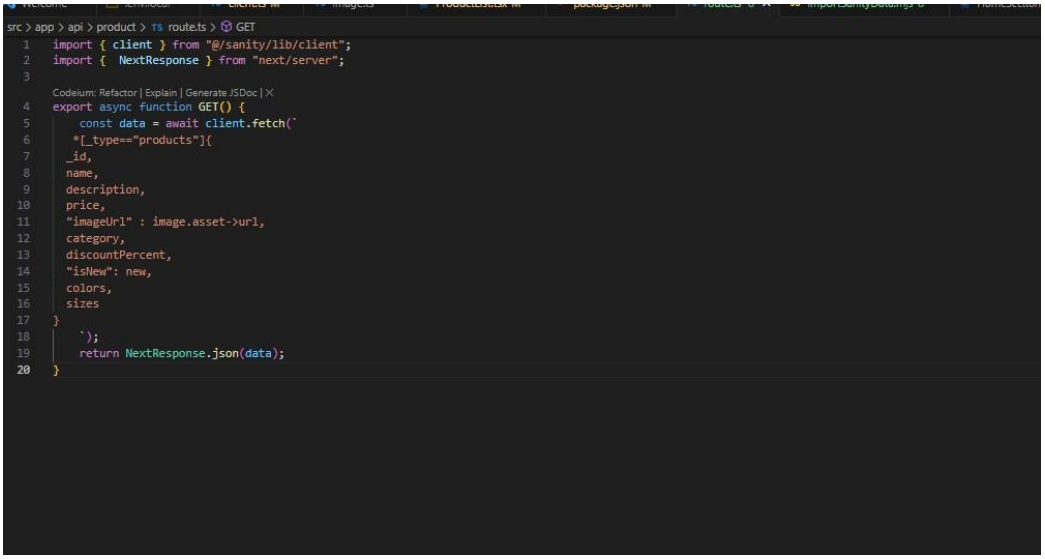


```
import { defineType, defineField, defineArrayMember } from "sanity";

export const product = defineType({
    name: "product",
    type: "document",
    title: "Product",
    fields: [
        defineField({
            name: "title",
            type: "string",
            title: "Title",
            validation: (Rule) => Rule.required().error("This field is required"),
        }),

        defineField({
            name: "slug",
            type: "slug",
            title: "Slug",
            validation: Rule => Rule.required().error('This field is req    uired'),
            options: {
                source: "title",
                maxLength: 96,
            },
        }),

        defineField({
            name: "price",
            type: "number",
            title: "Price",
            validation: (Rule) => Rule.required().error("This field is required"),
        }),

        defineField({
            name: "discountedPrice",
            type: "number",
            title: "Discounted Price",
        }),

        defineField({
            name: "summary",
            type: "text",
            title: "Summary",
            validation: (Rule) => Rule.required().error("This field is required"),
        }),

        defineField({
            name: "sizeQuantities",
            type: "object",
            title: "Size Quantities",
            fields: [
                defineField({
                    name: "large",
                    type: "number",
                    title: "Large Quantity",
                    validation: (Rule) => Rule.required().min(0).error("Large quantity must be a positive number"),
                }),
                defineField({
                    name: "medium",
                    type: "number",
                    title: "Medium Quantity",
                    validation: (Rule) => Rule.required().min(0).error("Medium quantity must be a positive number"),
                }),
                defineField({
                    name: "small",
                    type: "number",
                    title: "Small Quantity",
                    validation: (Rule) => Rule.required().min(0).error("Small quantity must be a positive number"),
                }),
            ],
        }),

        defineField({
```


2. API Integration and Data Migration:

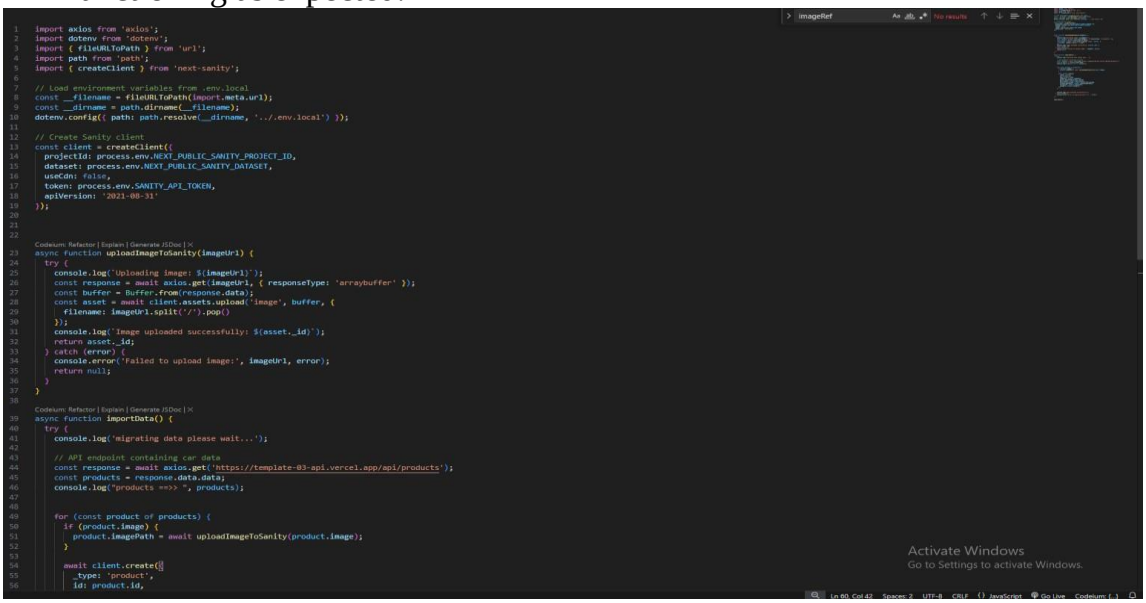
- **API Data Fetching:**
  I retrieved product data from an external API, which included key details such as images, titles, descriptions, prices, and tags. This data was then mapped to the relevant fields in the **Sanity CMS** schema.
- **Data Population in Sanity CMS:**
  After fetching the API data, I dynamically populated the product fields in Sanity CMS. This approach enabled the automated input of product information, ensuring consistency and accuracy across the platform while reducing manual effort.
- **Data Migration:**
  Utilizing the **Sanity CLI**, I exported the dataset from Sanity CMS for backup purposes and later re-imported it during testing. This ensured that the data remained well-structured and displayed correctly throughout the migration process.



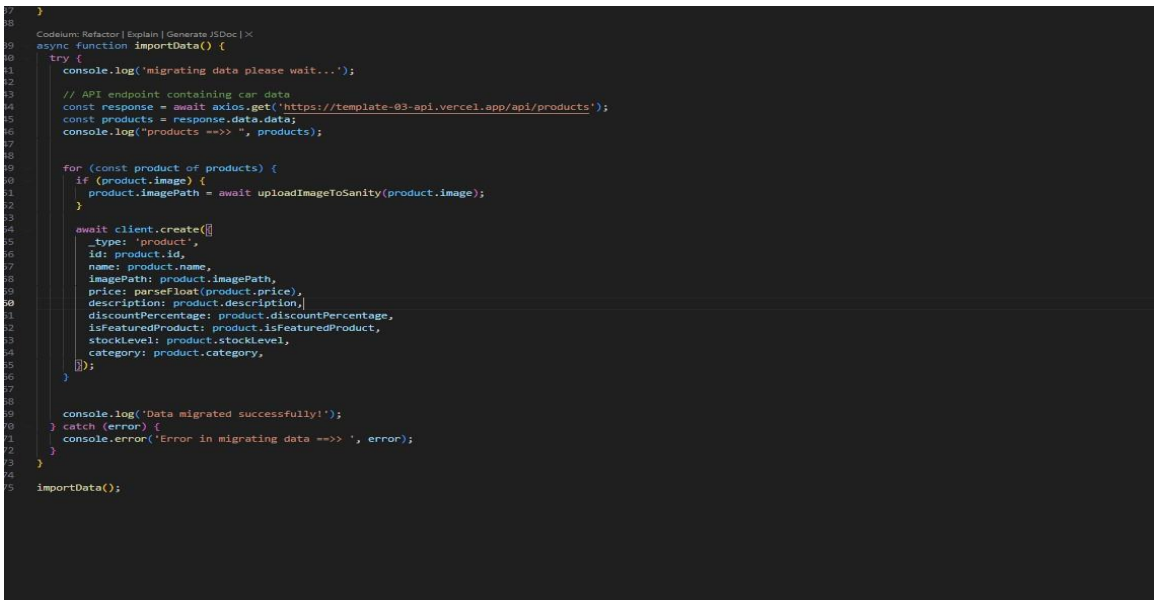


3. Steps Taken for Data Migration:

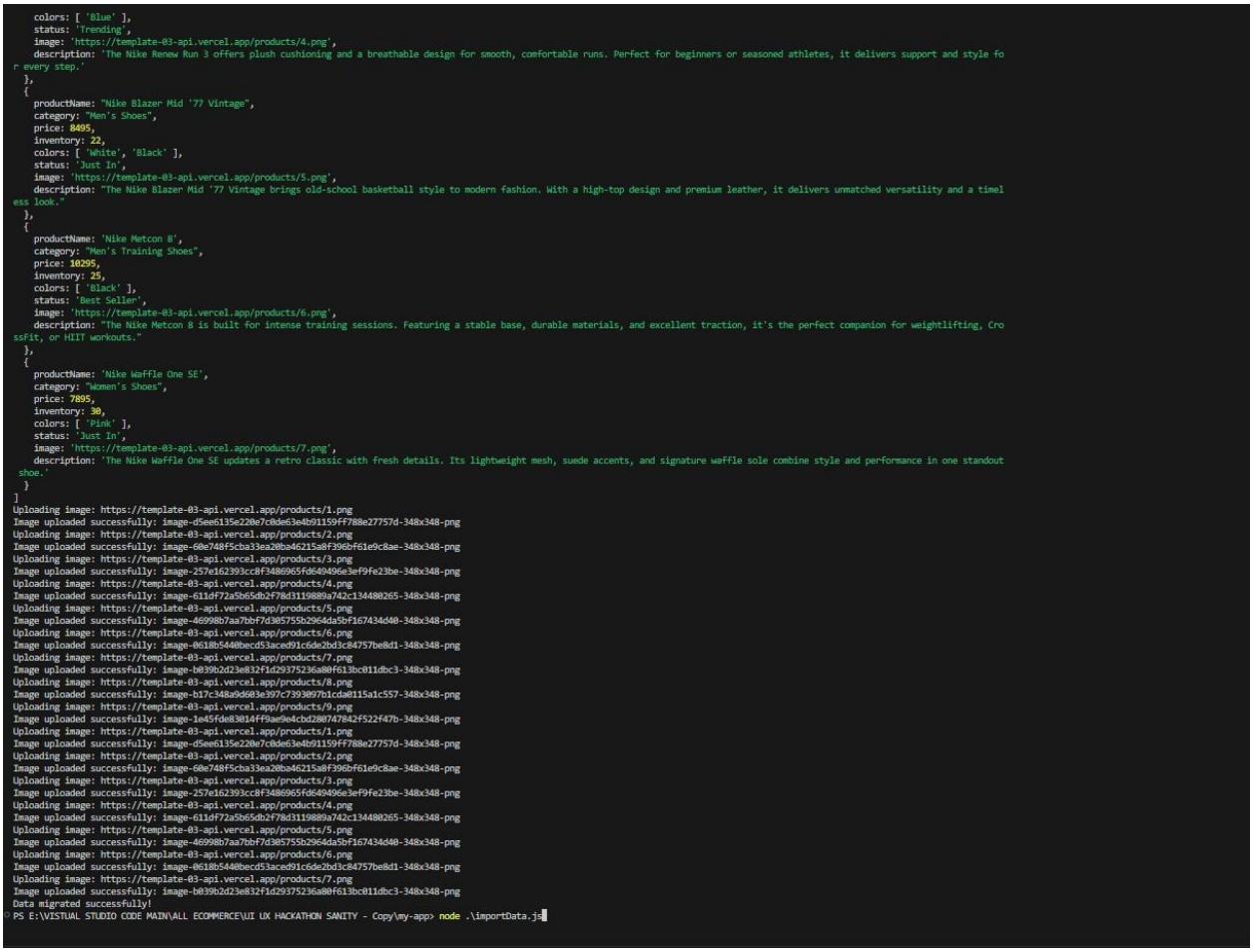
- **Exporting Data:**
  The first step involved exporting the data from **Sanity CMS** using the Sanity CLI. This ensured that all product data was securely backed up before performing any further operations.
- **Verification of Data:**
  The exported JSON structure was carefully reviewed to confirm that all fields were correctly populated. This step was critical to ensure the data would display accurately when fetched and rendered on the frontend.
- **Re-importing Data:**
  Once verified, the dataset was re-imported into **Sanity CMS**. This process validated that the data migration was successful and confirmed the system was functioning as expected.



```
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';
import { createClient } from 'next-sanity';

// 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'
});


Codeium: Refactor | Explain | Generate JSDoc | ×
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;
  }
}

Codeium: Refactor | Explain | Generate JSDoc | ×
async function importData() {
  try {
    console.log('migrating data please wait...');

    // API endpoint containing car data
    const response = await axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    console.log("products ==>> ", products);


    for (const product of products) {
      if (product.image) {
        product.imagePath = await uploadImageToSanity(product.image);
      }

      await client.create({
        _type: 'product',
        id: product.id,
```

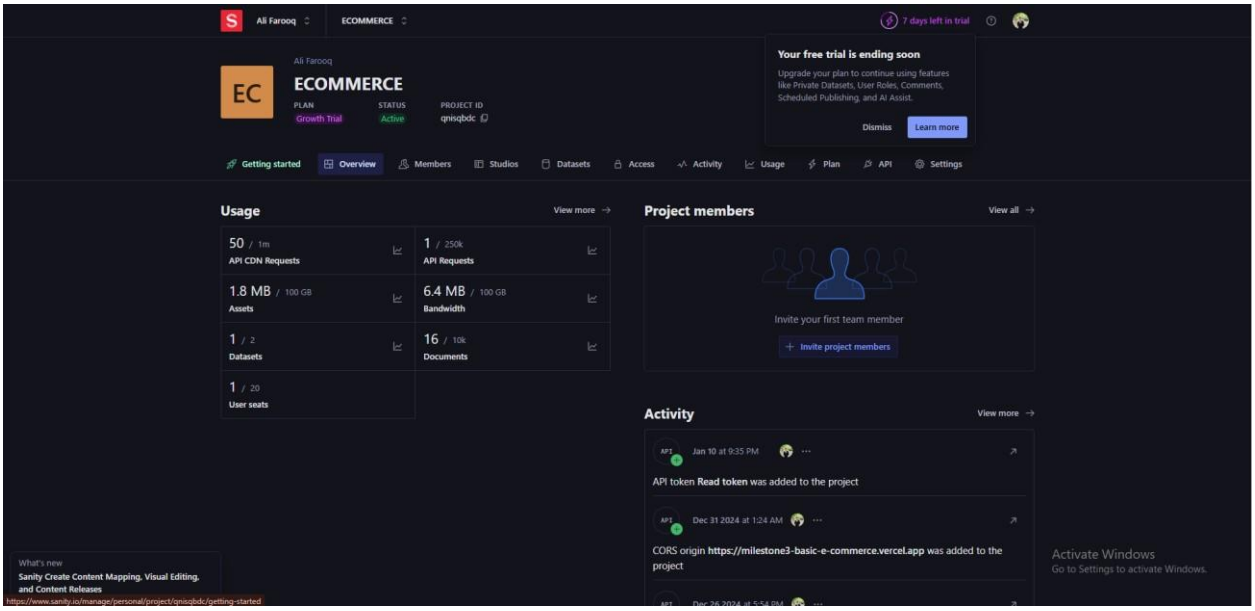


```
}

Codeium: Refactor | Explain | Generate JSDoc | ×
async function importData() {
  try {
    console.log('migrating data please wait...');

    // API endpoint containing car data
    const response = await axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    console.log("products ==>> ", products);


    for (const product of products) {
      if (product.image) {
        product.imagePath = await uploadImageToSanity(product.image);
      }

      await client.create({
        _type: 'product',
        id: product.id,
        name: product.name,
        imagePath: product.imagePath,
        price: parseFloat(product.price),
        description: product.description,
        discountPercentage: product.discountPercentage,
        isFeaturedProduct: product.isFeaturedProduct,
        stockLevel: product.stockLevel,
        category: product.category,
      });
    }


    console.log('Data migrated successfully!');
  } catch (error) {
    console.error('Error in migrating data ==>> ', error);
  }
}

importData();
```

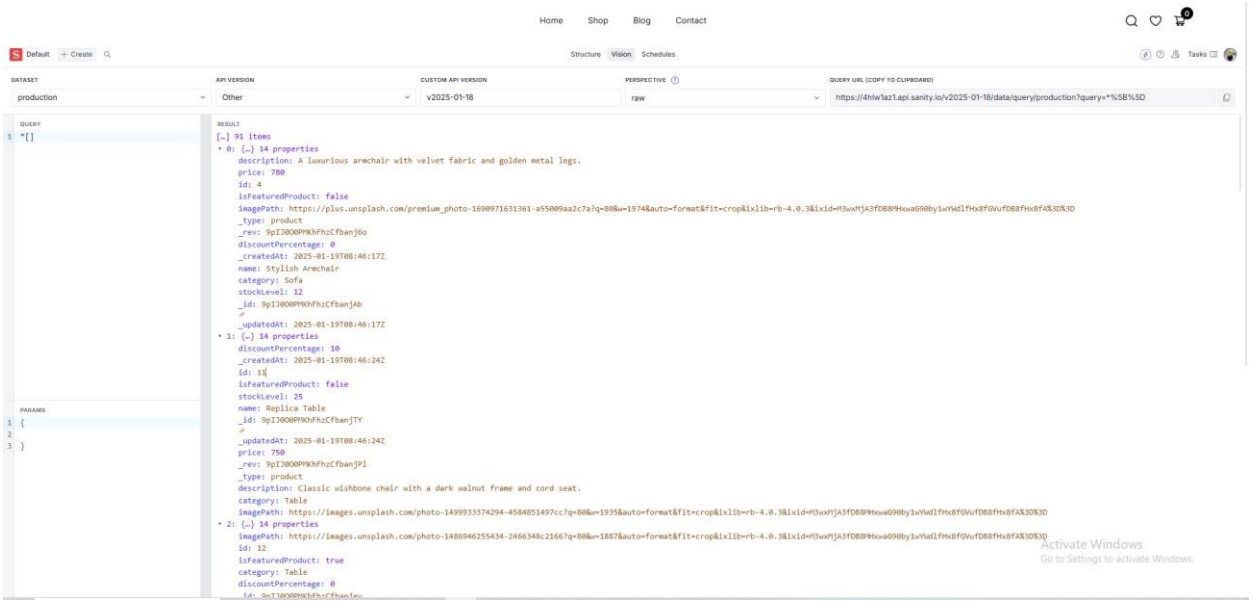


4. Tools Used:

Sanity Studio: Used for schema creation, content management, and displaying product data



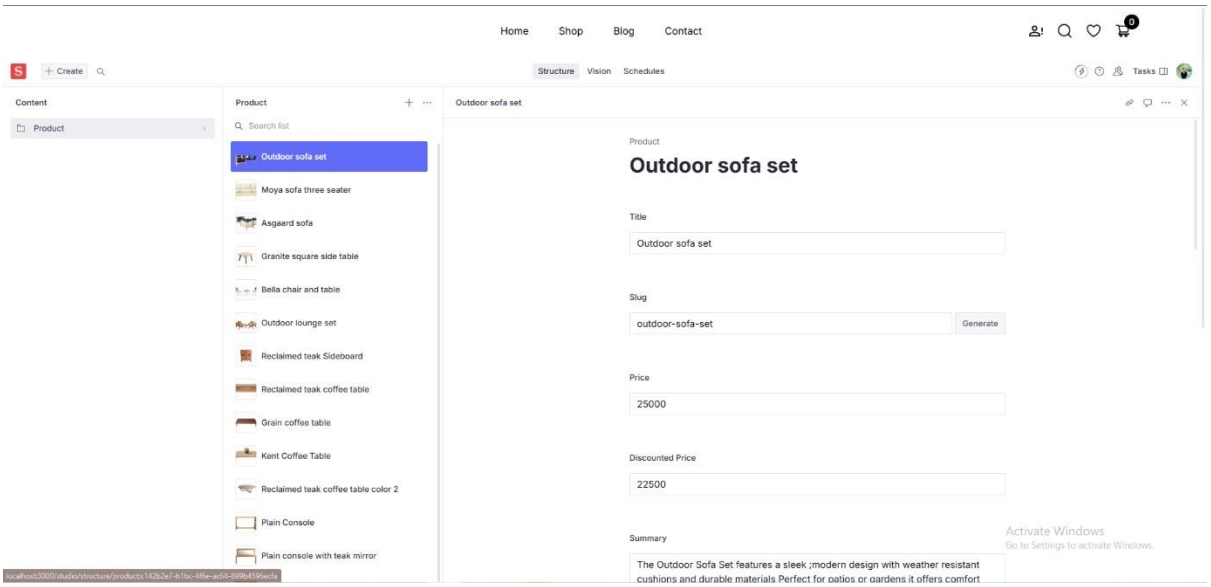
Sanity Vision:



Sanity CLI: Utilized for exporting and importing the dataset, ensuring data consistency and backup.

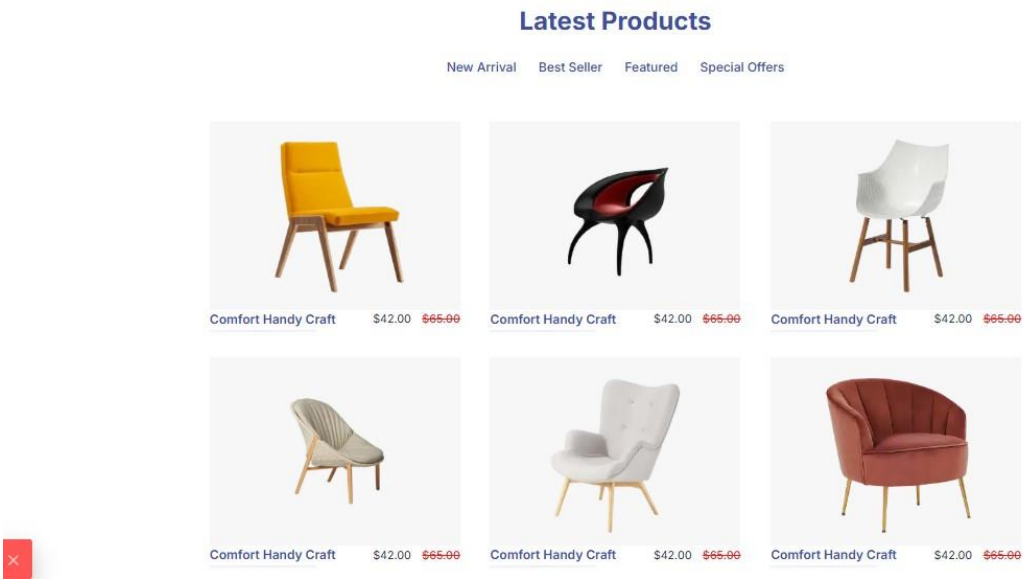
5. Screenshots and Frontend Display:

- **Sanity CMS Fields:**
  A screenshot showcasing the populated fields in **Sanity Studio**, displaying product details such as images, descriptions, and prices.



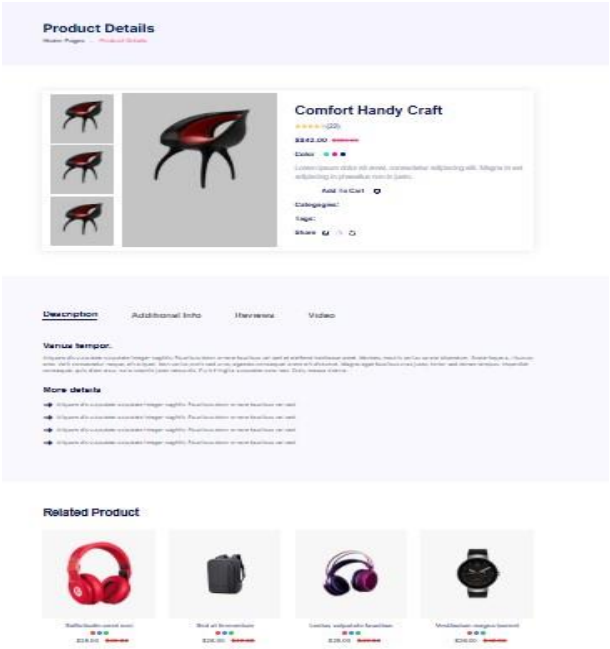
Frontend Display:

A screenshot illustrating how the product data is dynamically rendered on the marketplace frontend.



Product Detail Page Display:

Visual representation of the product detail page, highlighting the seamless integration of API data with the frontend design.



Conclusion:

The **API integration** and **data migration** processes were successfully implemented, significantly improving the efficiency and scalability of the Bandage project. The integration eliminated manual data entry, streamlining the addition and updating of product information. Meanwhile, the migration steps ensured that the data remained consistent and accurate across the system.

This setup makes the Bandage project more dynamic, robust, and easier to maintain, laying the foundation for continued growth and scalability.

Hackathon Day-3 Prepared By: Faraz Alam