Day 3 - API Integration Report – General eCommerce

Introduction

The purpose of this report is to document the process of API integration, schema adjustments, and migration steps performed as part of the project. This exercise focused on ensuring seamless data handling between the backend and frontend using modern tools and best practices.

API Integration Process

1. Overview:

We integrated the provided API into our application to populate the marketplace data. Below is an overview of the API endpoint utilized:

API URL: https://template6-six.vercel.app/api/products

The API provided data such as product titles, images, prices, and descriptions. These details were migrated to Sanity CMS and later fetched for frontend display.

2. Steps Taken:

Environment Variables:

Sensitive data was stored securely in `.env.local` to avoid hardcoding values. Key environment variables used:

NEXT_PUBLIC_SANITY_PROJECT_ID=[Insert Project ID]

NEXT_PUBLIC_SANITY_DATASET=[Insert Dataset]

NEXT_PUBLIC_SANITY_TOKAN=[Insert Token]

Sanity Client Creation:

Configured the Sanity client using the project ID and dataset. Handled sensitive data with care using environment files.

Data Fetching:

GROQ queries were used to fetch products from Sanity CMS. Fetched fields included:

id, title, productImage, price, originalPrice, discountPercentage, description

Data Processing:

Processed fetched data to align with frontend requirements. Generated unique image URLs dynamically for the frontend.

Sanity Documentation Creation:

Created a schema in Sanity CMS to align with the API structure. Key fields included: title, price, originalPrice, discountPercentage, isNew, tags, description.

Error Handling:
Added error handling during API calls and data fetching. Logged errors for debugging and displayed user-friendly messages in the frontend.

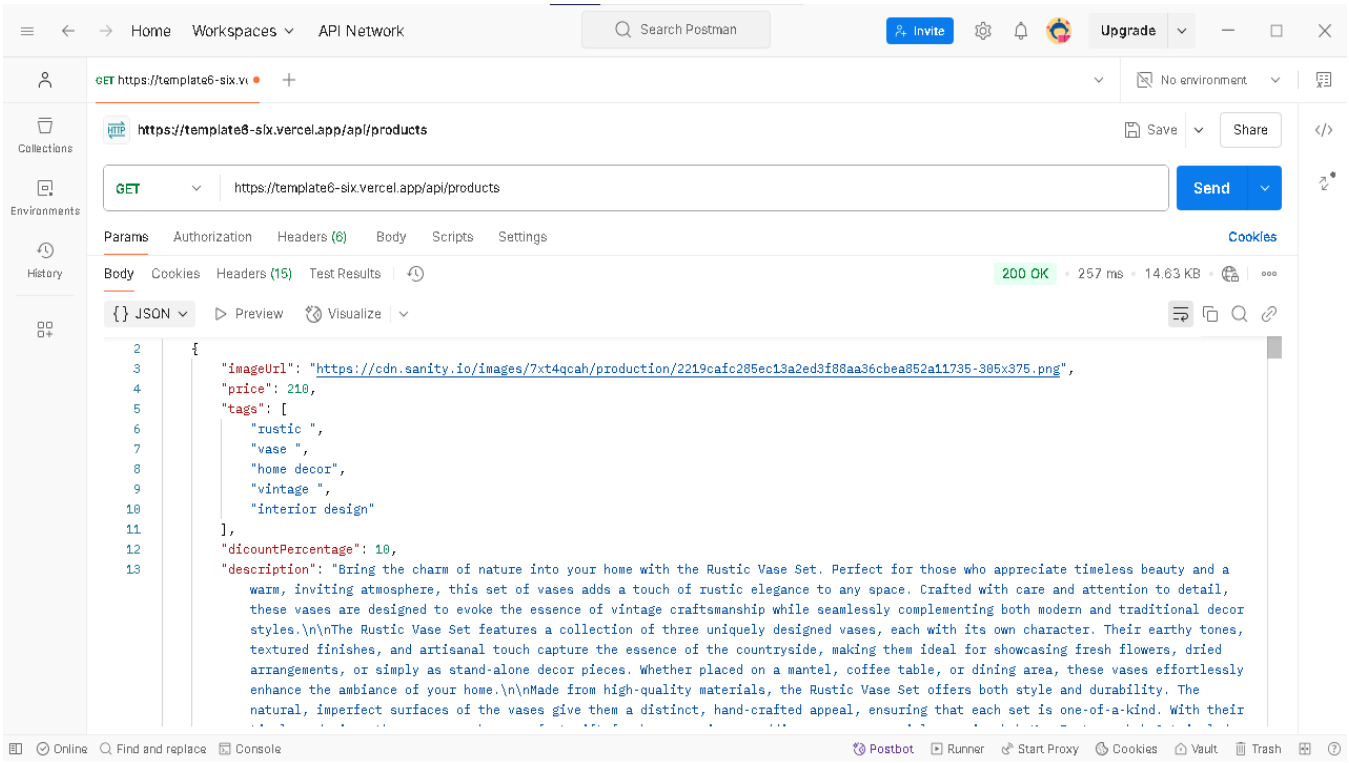
3. Migration Steps and Tools Used:

Migration Script:
Script used to fetch data from the API and populate Sanity CMS programmatically. Verified data accuracy during the migration process.

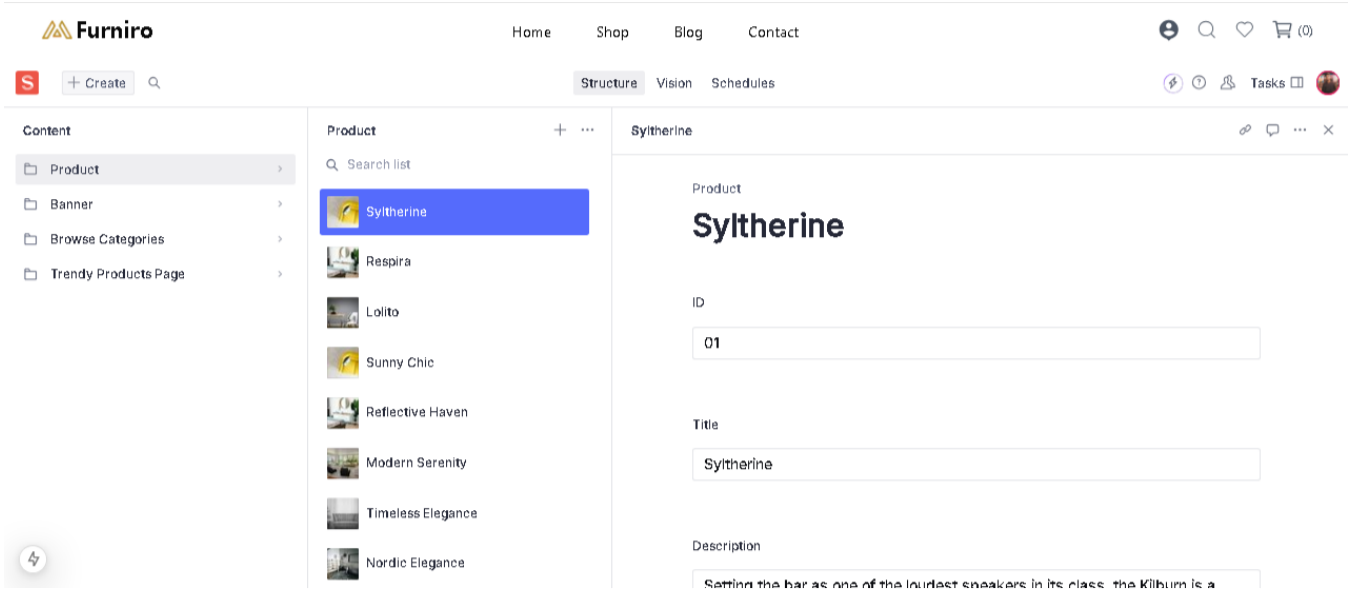
Sanity Schema:
Adjusted schema to match the API fields for seamless integration.

Screenshots:

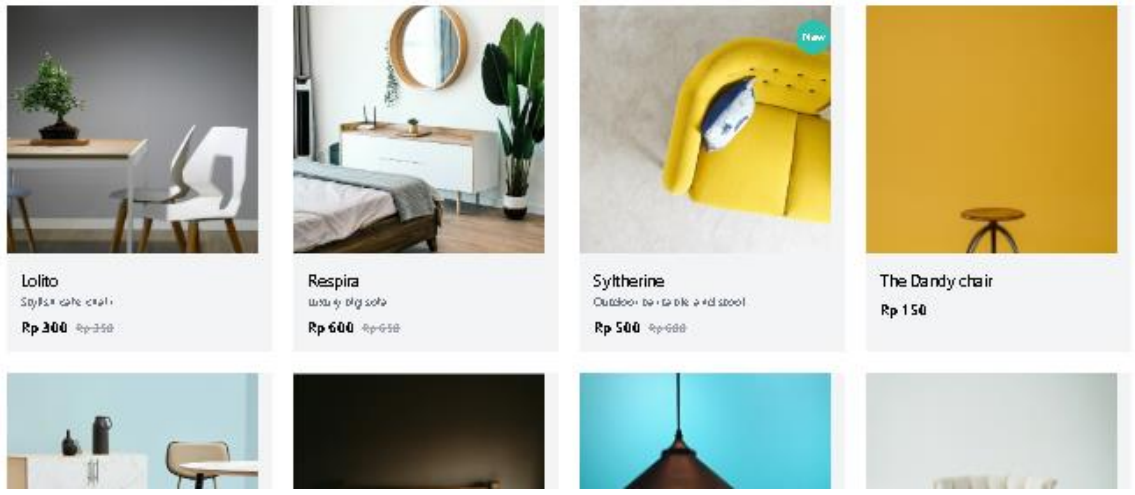
1. API Calls:



2. Populated Sanity CMS Fields:

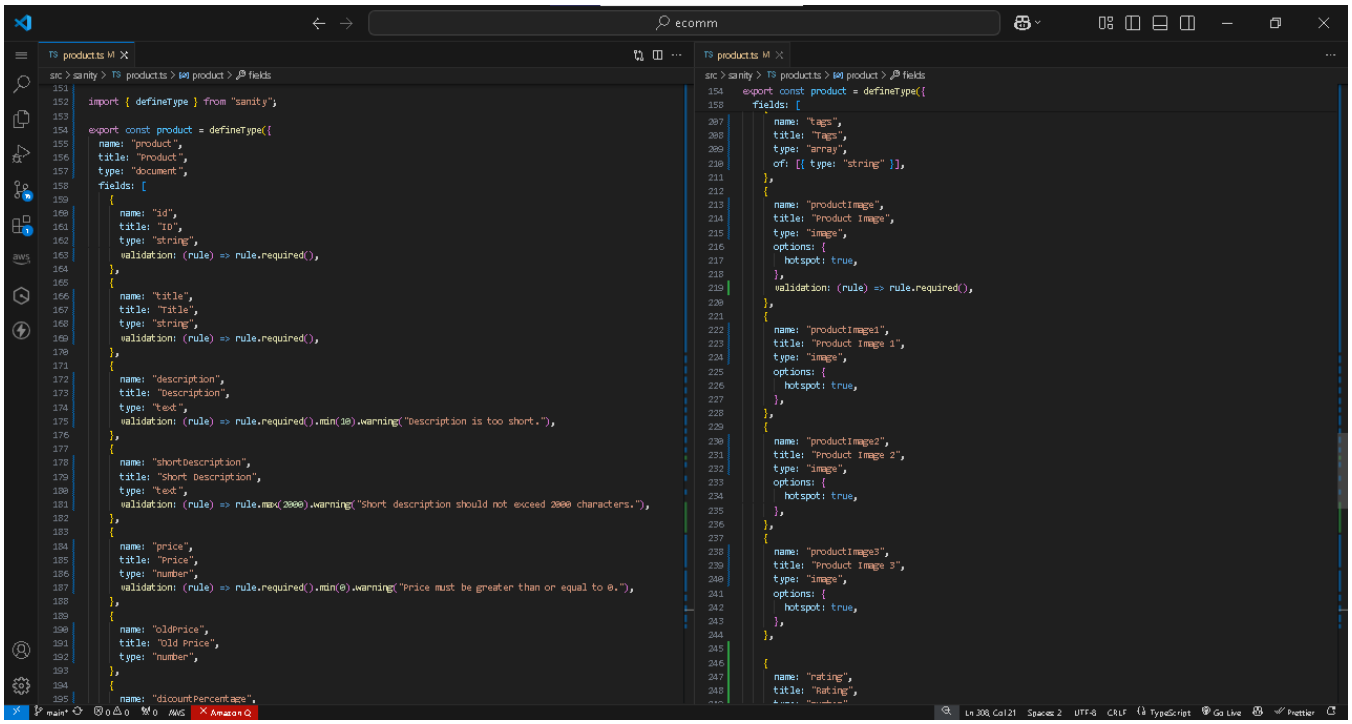


3. Data Successfully Displayed on Frontend:



Code Snippets

1.Sanity Schema.



```
import { defineType } from "sanity";

export const product = defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    {
      name: "id",
      title: "ID",
      type: "string",
      validation: (rule) => rule.required(),
    },
    {
      name: "title",
      title: "Title",
      type: "string",
      validation: (rule) => rule.required(),
    },
    {
      name: "description",
      title: "Description",
      type: "text",
      validation: (rule) => rule.required().min(10).warning("Description is too short."),
    },
    {
      name: "shortDescription",
      title: "Short Description",
      type: "text",
      validation: (rule) => rule.max(2000).warning("Short description should not exceed 2000 characters."),
    },
    {
      name: "price",
      title: "Price",
      type: "number",
      validation: (rule) => rule.required().min(0).warning("Price must be greater than or equal to 0."),
    },
    {
      name: "oldPrice",
      title: "Old Price",
      type: "number",
    },
    {
      name: "discountPercentage",
```

```
export const product = defineType({
  fields: [
    {
      name: "tags",
      title: "Tags",
      type: "array",
      of: [{ type: "string" }],
    },
    {
      name: "productImage",
      title: "Product Image",
      type: "image",
      options: {
        hotspot: true,
      },
      validation: (rule) => rule.required(),
    },
    {
      name: "productImage1",
      title: "Product Image 1",
      type: "image",
      options: {
        hotspot: true,
      },
    },
    {
      name: "productImage2",
      title: "Product Image 2",
      type: "image",
      options: {
        hotspot: true,
      },
    },
    {
      name: "productImage3",
      title: "Product Image 3",
      type: "image",
      options: {
        hotspot: true,
      },
    },

    {
      name: "rating",
      title: "Rating",
```

2.Frontend Integration:

```
useEffect(() => {
  const fetchProducts = async () => {
    try {
      const products: IProduct[] = await client.fetch(
        '*[_type == "product"]{_id, title, shortDescription, discount, price, oldPrice, tag, productImage}'
      );
      setData(products);
      setIsLoading(false);
    } catch (error) {
      console.error("Failed to fetch products:", error);
      setIsLoading(false);
    }
  };

  fetchProducts();
}, []);
```

3.MigrationScript:

importData.js

```
import { createClient } from '@sanity/client';

const client = createClient({
  projectId: "u0▬▬jdo",
  token:
"pp5WmsOLcSgqOv9PZC48k8er49CiKYBwokjwLikm2jfkO4j1M4tT48VyRny5j0bdj7Z6ejGvmHsUx5N7wQid57oaBL7qYY5fFKTS
zFtSzcCaS",
  dataset: "produ55656ction",
  apiVersion:"2024988-12-27",
  useCdn: true,
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);

    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }

    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);

    const asset = await client.assets.upload('image', bufferImage, {
      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 uploadProduct(product) {
  try {
    const imageId = await uploadImageToSanity(product.imageUrl);

    if (imageId) {
      const document = {
        _type: 'product',
        title: product.title,
        price: product.price,
        productImage: {
          _type: 'image',
          asset: {
            _ref: imageId,
          },
        },
        tags: product.tags,
        dicountPercentage: product.dicountPercentage, // Typo in field name: dicountPercentage →
discountPercentage
        description: product.description,
        isNew: product.isNew,
      };

      const createdProduct = await client.create(document);
      console.log(`Product ${product.title} uploaded successfully:`, createdProduct);
    } else {
      console.log(`Product ${product.title} skipped due to image upload failure.`);
    }
  } catch (error) {
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {
  try {
    const response = await fetch('https://template6-six.vercel.app/api/products');

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const products = await response.json();

    for (const product of products) {
      await uploadProduct(product);
    }
  } catch (error) {
    console.error('Error fetching products:', error);
  }
}

importProducts();
```