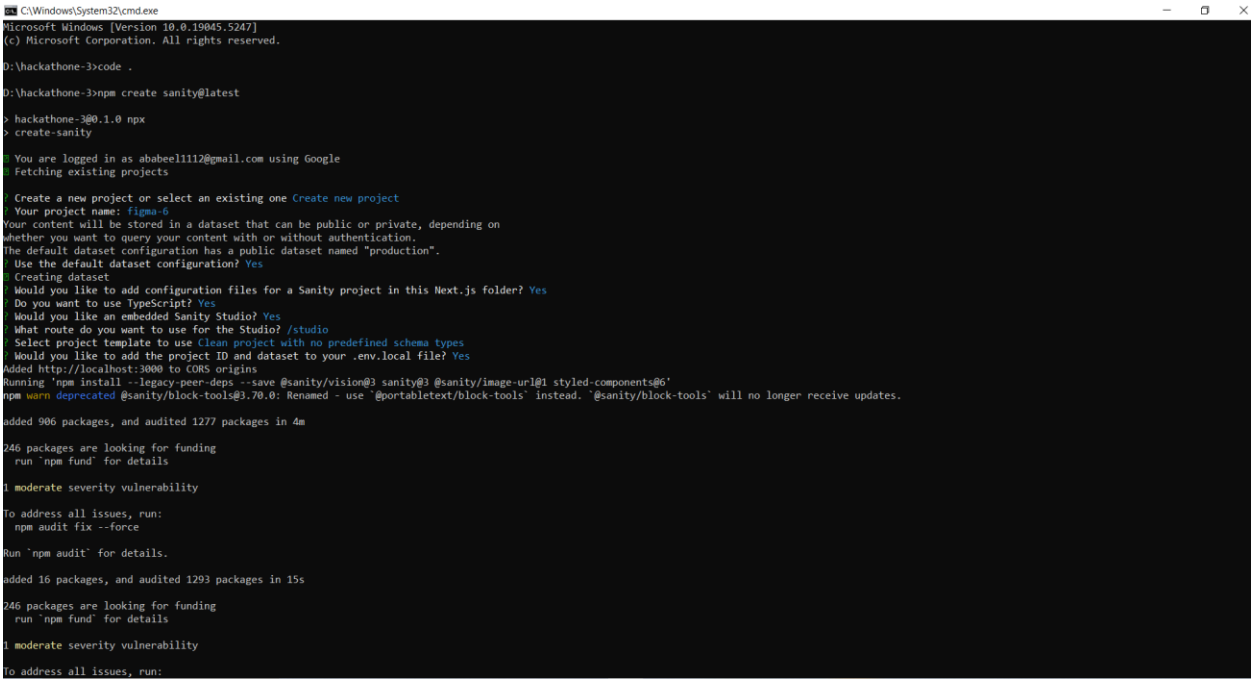
Day 3 - API Integration Report - Furniro

Introduction:

The objective of this report is to strengthen our expertise in API integration and data migration by developing a fully operational marketplace backend. This task involved importing data from a provided API into Sanity CMS and seamlessly integrating it with a Next.js frontend. By undertaking this project, we aimed to simulate real-world development scenarios and equip ourselves to address a variety of client needs effectively.

API Integration Process:



```
Run `npm audit` for details.

Success! Your Sanity configuration files has been added to this project

D:\hackathone-3>
```

1. Overview:

We utilized the designated API for Template 6:
API URL: https://template6-six.vercel.app/api/products
This API supplied comprehensive product information, including titles, images, pricing, and descriptions. The data was seamlessly migrated to Sanity CMS and subsequently retrieved to dynamically render and display on the frontend.

```
}

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();
```

2. Steps Taken:

Environment Variables:

- Sensitive data was securely stored in the .env.local file to prevent hardcoding and maintain security.
- The following environment variables were utilized:
    - **Sanity Project ID**
    - **Sanity Dataset**

Sanity Client Configuration:

- The Sanity client was set up within the Next.js project using the project ID and dataset.
- Sensitive information was securely managed using .env files to ensure data protection.

Data Fetching:

- GROQ queries were employed to retrieve product data from Sanity CMS.
- Key fields such as _id, title, productImage, price, originalPrice, discountPercentage, and description were queried for frontend display.

Data Processing:

- The fetched data was processed to meet frontend requirements.
- The urlFor function was used to dynamically generate image URLs for seamless rendering.

Sanity Schema Development:

- A schema was created in Sanity CMS to mirror the structure of the API data.
- Fields included title, price, originalPrice, discountPercentage, isNew, tags, and description to ensure consistency.

Error Handling:

- Robust error handling was implemented during API calls and data fetching.
- Errors were logged for debugging purposes, and user-friendly messages were displayed on the frontend to enhance the user experience.

Migration Steps and Tools:

Migration Script:

- A custom script was developed to fetch data from the API and programmatically populate Sanity CMS.
- Data validation was performed during migration to ensure accuracy and integrity.

Sanity Schema Adjustment:

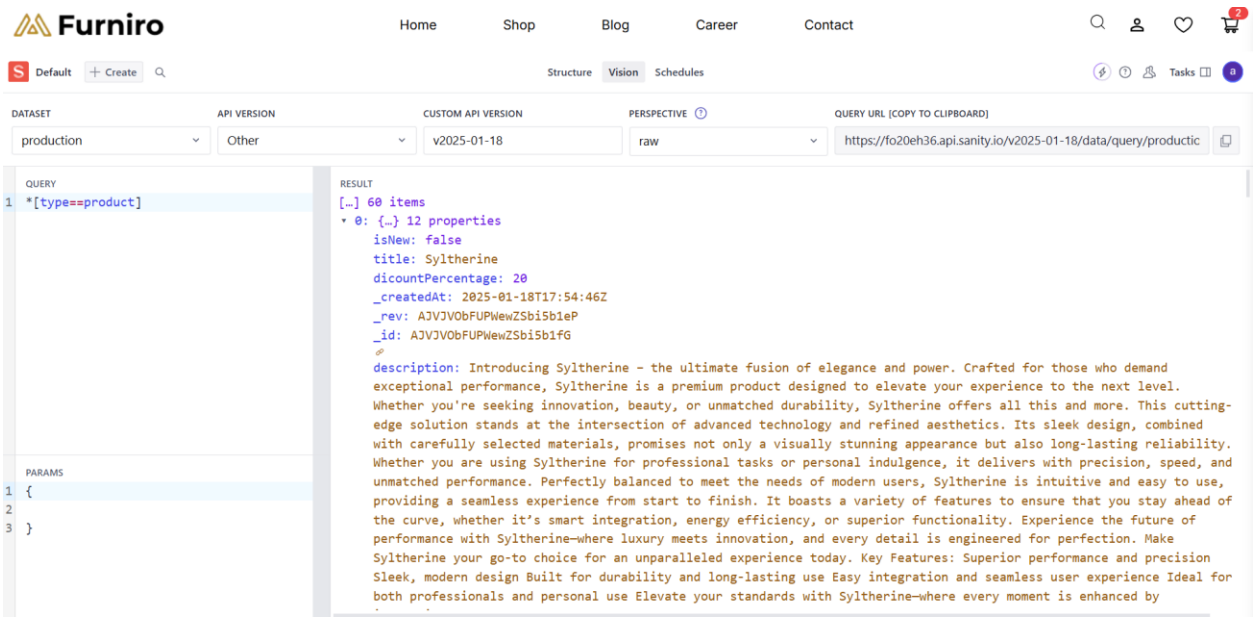
- The schema was fine-tuned to align with the API fields, ensuring a smooth migration and integration process.

Frontend Integration:

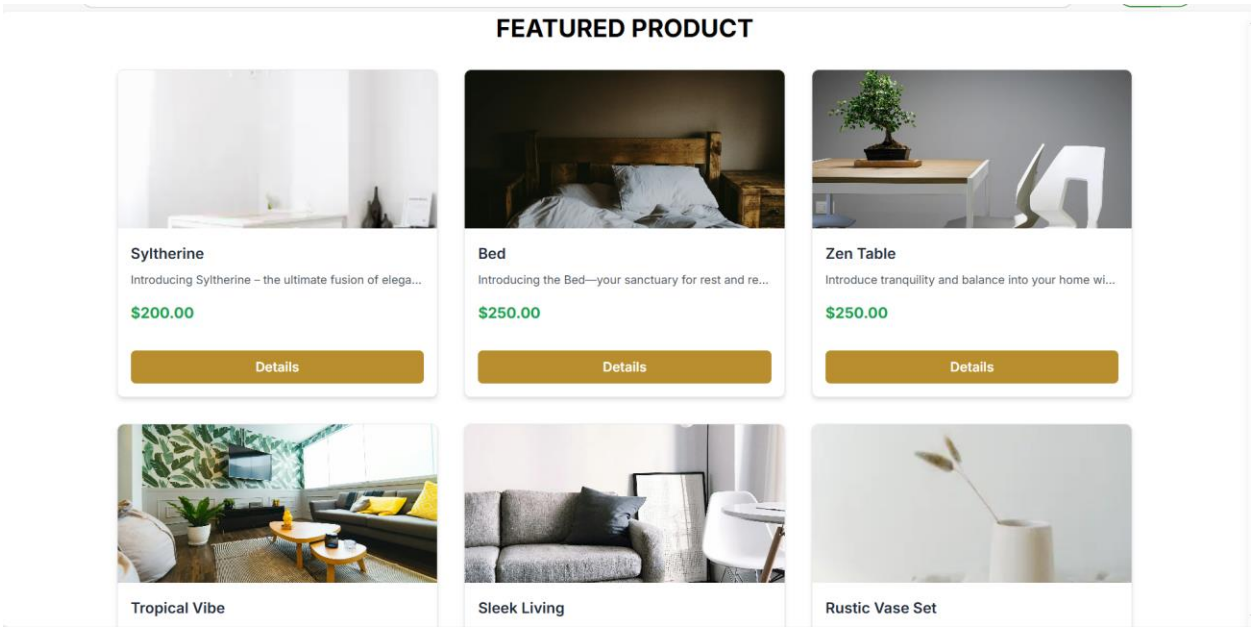
- The migrated data was dynamically integrated into the Next.js frontend.
- A loading state was implemented to improve user experience during data fetching.

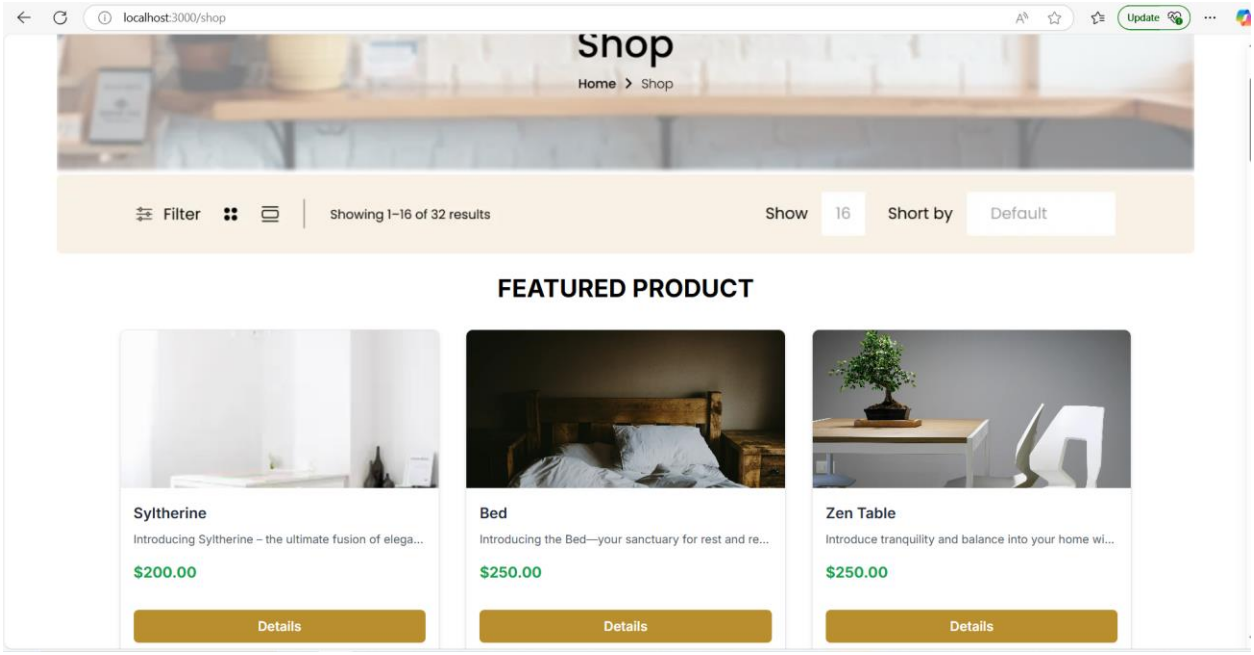
SCREEN SHOTS

API Calls:

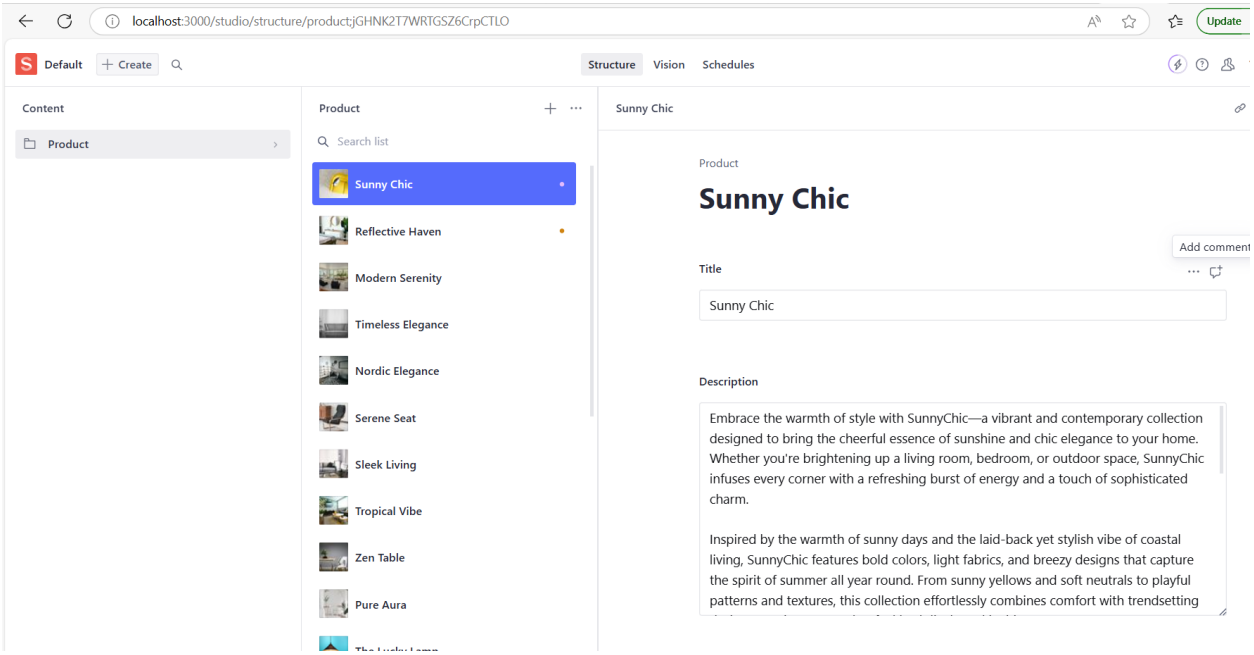


Frontend Display:





SANITY FIELD:



Sanity Fields:Code Snippets:

1. Sanity Schema

The schema used in Sanity CMS for storing product data:

src > sanity > schemaTypes > product.ts > product > fields

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

export const product = defineType({
    name: "product",
    title: "Product",
    type: "document",
    fields: [
        {
            name: "title",
            title: "Title",
            validation: (rule) => rule.required(),
            type: "string"
        },
        {
            name:"description",
            type:"text",
            validation: (rule) => rule.required(),
            title:"Description",
        },
        {
            name: "productImage",
            type: "image",
            validation: (rule) => rule.required(),
            title: "Product Image"
        },
        {
            name: "price",
            type: "number",
            validation: (rule) => rule.required(),
            title: "Price",
        },
        {
            name: "tags",
            type: "array",
            title: "Tags",
            of: [{ type: "string" }]
        },
```

2. GROQ Query for Frontend Integration:

The query used to fetch products from Sanity CMS for rendering in the frontend:

```
interface Product {
  _id: string;
  title: string;
  price: number;
  description: string;
  discountPercentage: number;
  imageUrl: string;
  productImage: {
    asset: {
      _ref: string;
    };
  };
  tags: string[];
}

const ProductCards: React.FC = () => {
  const [products, setProducts] = useState<Product[]>([]);

  const fetchProducts = async () => {
    try {
      const query = `*[_type == "product"] {
        _id,
        title,
        price,
        description,
        discountPercentage,
        "imageUrl": productImage.asset->url,
        tags
      }`;
      const data = await sanity.fetch(query);
      setProducts(data);
    } catch (error) {
      console.error("Error Fetching Products:", error);
    }
  };
```

3. Migration Script

The script used to fetch product data from the API and populate Sanity CMS:

```
importData.js > client > projectId
});

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',
```

Final Checklist:

API Understanding

✓

Schema Validation

✓

Data Migration

✓

API