

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:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

D:\hackathone-3>code .

D:\hackathone-3>npm create sanity@latest

> hackathone-3@0.1.0 npx
> create-sanity

  You are logged in as ababeel1112@gmail.com using Google
  Fetching existing projects

  Create a new project or select an existing one Create new project
  Your project name: figma-6
  Your content will be stored in a dataset that can be public or private, depending on
  whether you want to query your content with or without authentication.
  The default dataset configuration has a public dataset named "production".
  Use the default dataset configuration? Yes
  Creating dataset
  Would you like to add configuration files for a Sanity project in this Next.js folder? Yes
  Do you want to use TypeScript? Yes
  Would you like an embedded Sanity Studio? Yes
  What route do you want to use for the Studio? /studio
  Select project template to use Clean project with no predefined schema types
  Would you like to add the project ID and dataset to your .env.local file? Yes
Added http://localhost:3000 to CORS origins
Running 'npm install --legacy-peer-deps --save @sanity/vision@3 sanity@3 @sanity/image-url@1 styled-components@6'
npm warn deprecated @sanity/block-tools@3.70.0: Renamed - use '@portabletext/block-tools' instead. '@sanity/block-tools' will no longer receive updates.

added 906 packages, and audited 1277 packages in 4m

246 packages are looking for funding
  run 'npm fund' for details

1 moderate severity vulnerability

To address all issues, run:
  npm audit fix --force

Run 'npm audit' for details.

added 16 packages, and audited 1293 packages in 15s

246 packages are looking for funding
  run 'npm fund' for details

1 moderate severity vulnerability

To address all issues, run:
```

```
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.

```
63 }  
64 }  
65  
66 async function importProducts() {  
67   try {  
68     const response = await fetch('https://template6-six.vercel.app/api/products');  
69  
70     if (!response.ok) {  
71       throw new Error(`HTTP error! Status: ${response.status}`);  
72     }  
73  
74     const products = await response.json();  
75  
76     for (const product of products) {  
77       await uploadProduct(product);  
78     }  
79   } catch (error) {  
80     console.error('Error fetching products:', error);  
81   }  
82 }  
83  
84 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:

The screenshot displays the Furniro API interface. At the top, there's a navigation bar with links for Home, Shop, Blog, Career, and Contact. Below this is a search bar and a 'Create' button. The main interface is divided into several sections: DATASET (production), API VERSION (Other), CUSTOM API VERSION (v2025-01-18), PERSPECTIVE (raw), and QUERY URL (COPY TO CLIPBOARD) (https://fo20eh36.api.sanity.io/v2025-01-18/data/query/productic). The QUERY section shows a query: `*[type==product]`. The RESULT section shows the response: `[...] 60 items` and a detailed description of the product, Syltherine.

Furniro

Home Shop Blog Career Contact

Default + Create

Structure Vision Schedules

DATASET: production API VERSION: Other CUSTOM API VERSION: v2025-01-18 PERSPECTIVE: raw QUERY URL (COPY TO CLIPBOARD): https://fo20eh36.api.sanity.io/v2025-01-18/data/query/productic

QUERY

```
1 *[type==product]
```

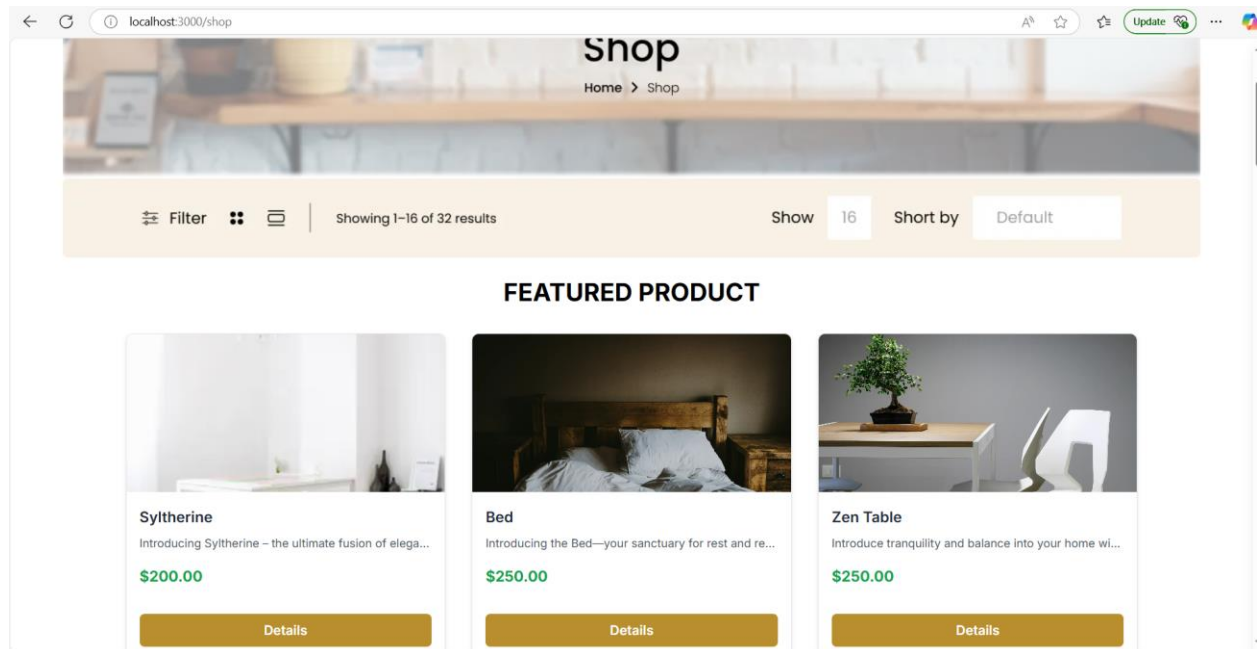
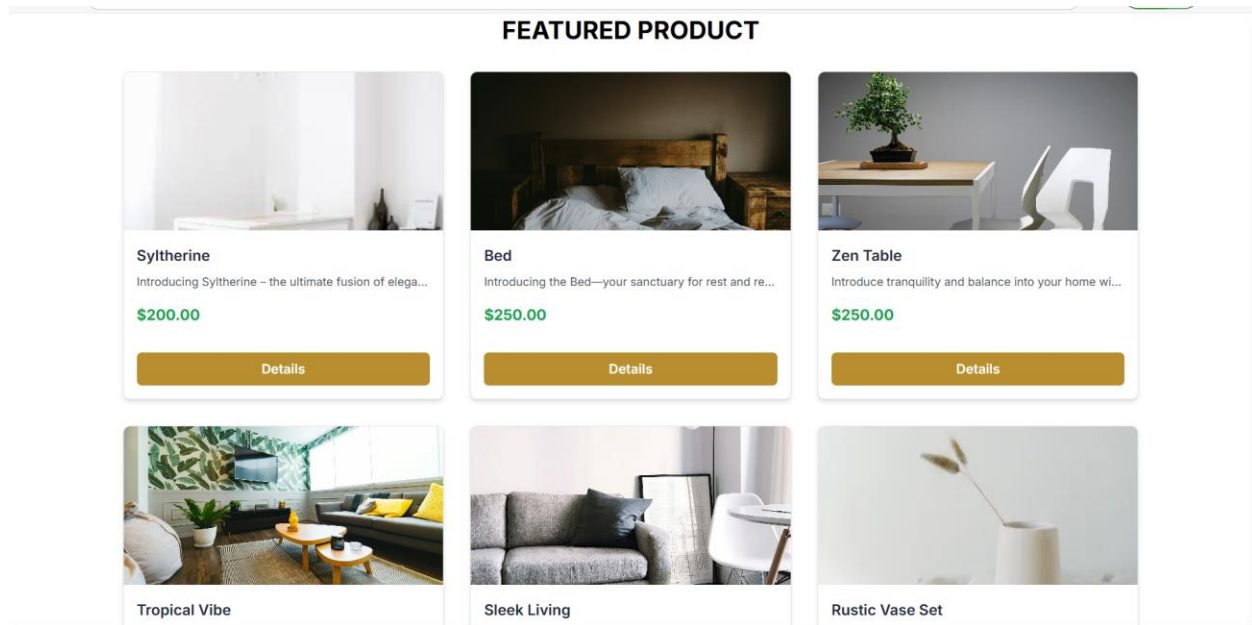
PARAMS

```
1 {
2
3 }
```

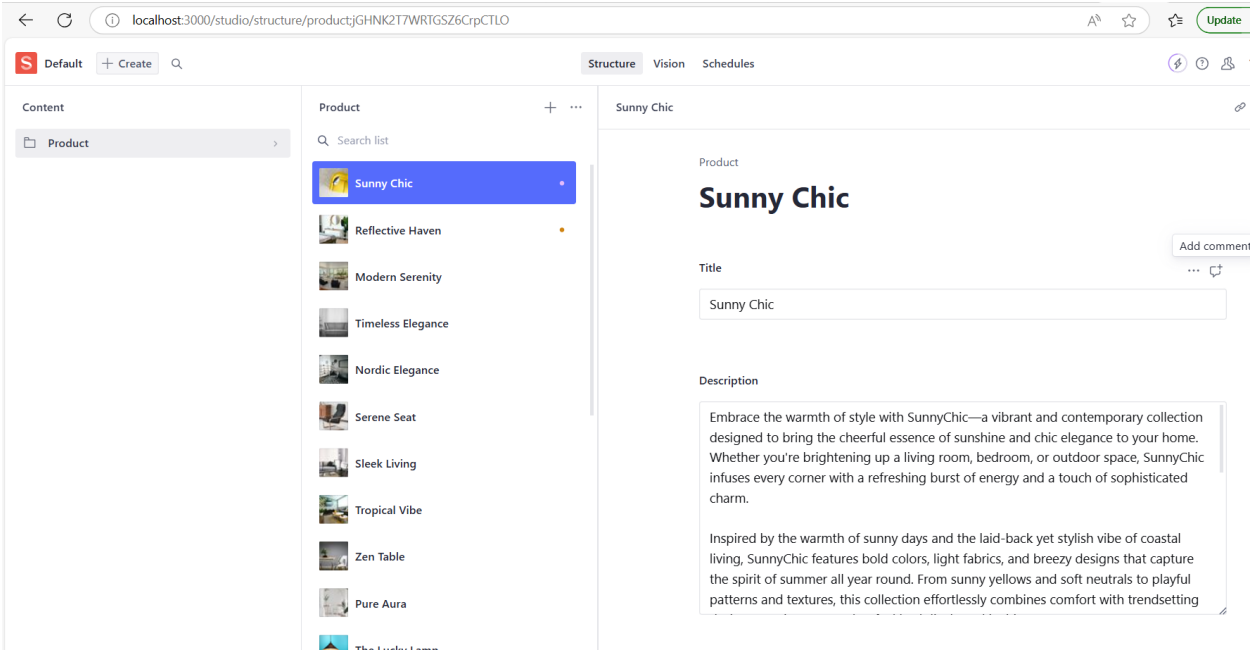
RESULT

```
[...] 60 items
  0: {...} 12 properties
    isNew: false
    title: Syltherine
    dicountPercentage: 20
    _createdAt: 2025-01-18T17:54:46Z
    _rev: AJVJVObFUPWewZSbi5b1eP
    _id: AJVJVObFUPWewZSbi5b1fG
    description: Introducing Syltherine - the ultimate fusion of elegance and power. Crafted for those who demand exceptional performance, Syltherine is a premium product designed to elevate your experience to the next level. Whether you're seeking innovation, beauty, or unmatched durability, Syltherine offers all this and more. This cutting-edge solution stands at the intersection of advanced technology and refined aesthetics. Its sleek design, combined with carefully selected materials, promises not only a visually stunning appearance but also long-lasting reliability. Whether you are using Syltherine for professional tasks or personal indulgence, it delivers with precision, speed, and unmatched performance. Perfectly balanced to meet the needs of modern users, Syltherine is intuitive and easy to use, providing a seamless experience from start to finish. It boasts a variety of features to ensure that you stay ahead of the curve, whether it's smart integration, energy efficiency, or superior functionality. Experience the future of performance with Syltherine—where luxury meets innovation, and every detail is engineered for perfection. Make Syltherine your go-to choice for an unparalleled experience today. Key Features: Superior performance and precision. Sleek, modern design. Built for durability and long-lasting use. Easy integration and seamless user experience. Ideal for both professionals and personal use. Elevate your standards with Syltherine—where every moment is enhanced by
```

Frontend Display:



SANITY FIELD:



Sanity Fields:Code Snippets:

1. Sanity Schema

The schema used in Sanity CMS for storing product data:

```
src > sanity > schemaTypes > TS product.ts > [e] product > fields
1  import { defineType } from "sanity"
2
3  export const product = defineType({
4    name: "product",
5    title: "Product",
6    type: "document",
7    fields: [
8      {
9        name: "title",
10       title: "Title",
11       validation: (rule) => rule.required(),
12       type: "string"
13     },
14     {
15       name: "description",
16       type: "text",
17       validation: (rule) => rule.required(),
18       title: "Description",
19     },
20     {
21       name: "productImage",
22       type: "image",
23       validation: (rule) => rule.required(),
24       title: "Product Image"
25     },
26     {
27       name: "price",
28       type: "number",
29       validation: (rule) => rule.required(),
30       title: "Price",
31     },
32     {
33       name: "tags",
34       type: "array",
35       title: "Tags",
36       of: [{ type: "string" }]
37     },
38   ],
39 })
```


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:

```
> JS 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