



Day 3: API Integration and Data Migration

Introduction

In Day 3, we focus on **API integration** and **data migration** into **Sanity CMS**. This process allows us to dynamically pull and display product data from an external API and subsequently populate our **Sanity CMS** with that data. The objective is to connect an external product data source (API) with a headless CMS (Sanity) and display it in a **Next.js** frontend.

Key Learning Outcomes

- ❖ **API Integration:** Understand how to fetch data from external APIs and use that data within a **Next.js** app.
- ❖ **Schema Modifications:** Learn how to modify and design schemas in **Sanity CMS** to fit your data structure.
- ❖ **Data Migration:** Migrate product data to **Sanity CMS** using API calls and images from external URLs.
- ❖ **Frontend Display:** Learn to display migrated data, including images, in a visually appealing manner on the frontend.

API Integration Process

1. API Integration

The first step in this project was to **integrate an external API** into our **Next.js** app to fetch product data. We used the following API:

API Endpoint: <https://template-03-api.vercel.app/api/products>

This API returns a list of products, each containing details such as:

- Product name
- Category
- Price
- Inventory count
- Available colors
- Status
- Image URL
- Description

We used **axios** to fetch the data and then set it in the **React state** for rendering.

API Request Setup:

We use Axios for making API requests to fetch the product data from an endpoint
import axios from 'axios';

```
async function fetchData() {  
  
  try {  
  
    const response = await  
    axios.get('https://template-03-api.vercel.app/api/products');  
  
    const products = response.data.data;  
  
    console.log("Products fetched:", products);  
  } catch (error) {  
    console.error('Error fetching product data:', error);  
  }  
}  
fetchData();
```

This function sends a GET request to the API, retrieves product data, and logs it.

This product data is later used to populate the Sanity CMS

Schema Adjustments

For the product data to be correctly migrated and stored in **Sanity CMS**, we adjusted the **Sanity schema** to match the product fields coming from the API. We made sure that the data structure aligns with the expected format in Sanity, such as: For this project, we need to create a schema for storing product data such as productName, category, price, description, and image.

Sanity Schema:

Here is an example of how the schema for products might look in Sanity:

```
export default {  
  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: 'description',
```

```
title: 'Description',
type: 'text',
},
{
name:
'image', title:
'Image', type:
'image',
options: {
hotspot: true,
},
},
],
};
```

- Product Name, Category, Price: These are stored as strings and numbers in the schema.
- Image: The product image is of type image in Sanity, with additional options like hotspot to allow users to crop or focus on certain parts of the image.

We have ensured that the necessary elds are available in the schema for importing data like product names, descriptions, images, etc.

Migration Steps and Tools Used:

1. **Fetching Data:** We used the axioslibrary to fetch data from the external API.
2. **Image Upload:** For product images, we fetched the image as a buffer and uploaded it to Sanity using the Sanity client.
3. **Data Migration:** After fetching the product data, we iterated over the products and created new documents in **Sanity CMS** using the client.create()method.
4. **Sanity Client:** We used **Sanity's client library** (@sanity/client) to interact with the CMS.

Migration Script Overview:

1. Setting Up Sanity Client

To interact with the Sanity API, we need to set up the Sanity client using the provided credentials in .env.local.

```
import { createClient } from '@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',
});
```

- **projectId:** Your Sanity project ID.
- **dataset:** The dataset within the Sanity project.
- **useCdn:** Set to false for real-time data fetching duri

2. Uploading Images to Sanity

The images are fetched from the external URLs and uploaded to Sanity using the Sanity API.

```
import axios from 'axios';
async function uploadImageToSanity(imageUrl) { try
{
  console.log(`Uploading image: ${imageUrl}`);
  const response = await axios.get(imageUrl, { responseType: 'arraybuer' });
  const buer = Buer.from(response.data);
  const asset = await client.assets.upload('image', buer, {
    lename: 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;
}
}
```

- `axios.get(imageUrl)`: Fetches the image as an arraybuer.
- `client.assets.upload`: Uploads the image as a buer to the Sanity Asset Manager.

3. Creating Product Documents in Sanity

For each product, a new document is created in the product schema.

javascript

CopyEdit

```
async function importData() {

  try {
    const response = await
    axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    for (const product of products) {
      let imageRef = null;
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image);
      }
      const sanityProduct = {
        _type: 'product',
        productName:
        product.productName, category:
        product.category,
        price: product.price,
```

```

inventory:
product.inventory, colors:
product.colors || [], status:
product.status,
description: product.description,
image: imageRef ? {
  _type:
'image', asset:
{
  _type: 'reference',
  _ref: imageRef,
},
}: undened,
};
await client.create(sanityProduct);
console.log(`Product ${product.productName} created successfully.`);
}
console.log('Data migrated successfully!');
} catch (error) {
console.error('Error migrating data:', error);
}
}
importData();

```

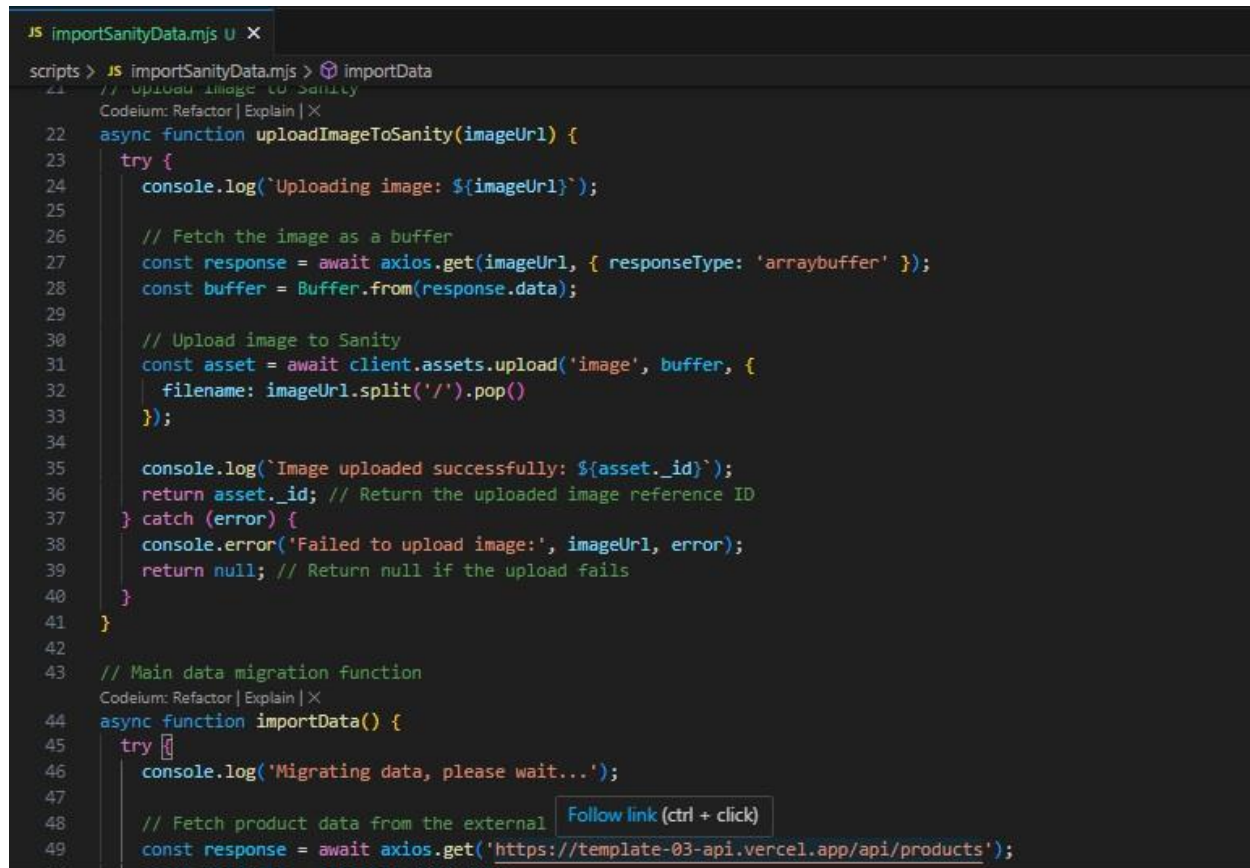
- **uploadImageToSanity:** First, we upload the product image to Sanity and retrieve its reference (`_id`).
- **client.create(sanityProduct):** This creates a new product document in the Sanity dataset.

Tools and Technologies Used:

- **Sanity CMS:** Headless CMS used for managing and storing product data.
 - **Axios:** For fetching data from the external API.
 - **Sanity API:** For interacting with the Sanity project and uploading assets.
 - **Next.js:** The React-based framework for building the front-end application.
 - **dotenv:** For managing environment variables like API keys securely
-

Visual Documentation

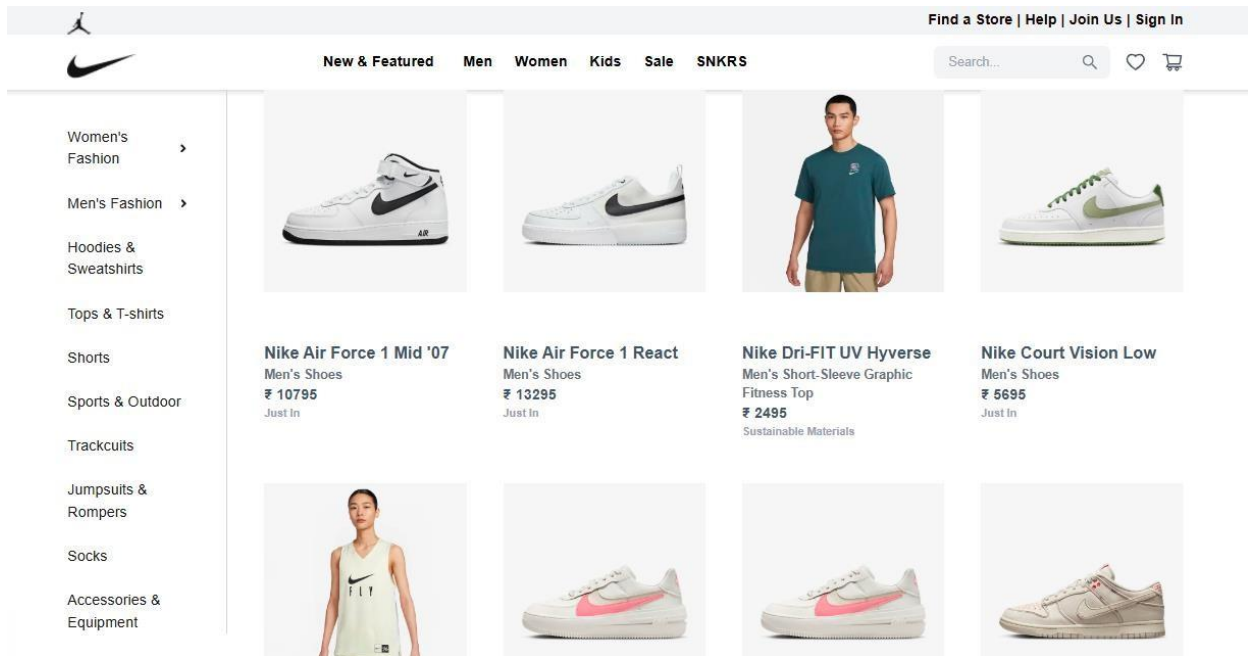
API Calls:



```
JS importSanityData.mjs U X
scripts > JS importSanityData.mjs > importData
21 // Upload image to Sanity
Codeium: Refactor | Explain | X
22 async function uploadImageToSanity(imageUrl) {
23   try {
24     console.log(`Uploading image: ${imageUrl}`);
25
26     // Fetch the image as a buffer
27     const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
28     const buffer = Buffer.from(response.data);
29
30     // Upload image to Sanity
31     const asset = await client.assets.upload('image', buffer, {
32       filename: imageUrl.split('/').pop()
33     });
34
35     console.log(`Image uploaded successfully: ${asset._id}`);
36     return asset._id; // Return the uploaded image reference ID
37   } catch (error) {
38     console.error('Failed to upload image:', imageUrl, error);
39     return null; // Return null if the upload fails
40   }
41 }
42
43 // Main data migration function
Codeium: Refactor | Explain | X
44 async function importData() {
45   try {
46     console.log('Migrating data, please wait...');
47
48     // Fetch product data from the external
49     const response = await axios.get('https://template-03-api.vercel.app/api/products');
```

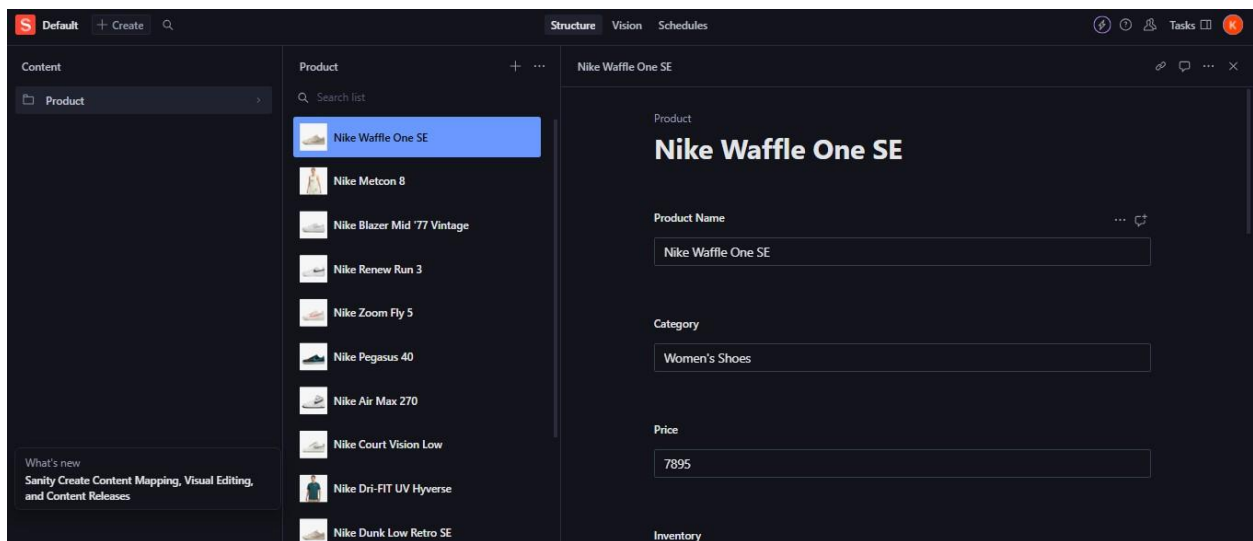
Screenshot displaying the successful API call to fetch product data.

Frontend Display



Screenshot of the products displayed on the frontend after the data migration.

Populated Sanity CMS Fields



Screenshot of the Sanity CMS dashboard showing the populated product fields.

Conclusion

Through the integration of external APIs, image uploading, and schema modifications, we've successfully migrated product data into Sanity CMS and displayed it on the frontend. This approach equips you with the skills necessary for integrating dynamic data sources into your CMS-driven projects and creating a seamless, data-driven marketplace.