API Integration and Data Migration

Prepared by Khalid Hussain

RN:00372206

Slot:Sunday 9-12

Day 3 - API Integration and Data Migration

1. Understanding the Provided API

Review the API Documentation

Before initiating the integration process, I thoroughly reviewed the API documentation for the assigned template. The key endpoints I focused on included:

- **Product Listings** (e.g., `/products`): For retrieving the list of available products.

2. Validating and Adjusting the Schema

Comparing the Existing Sanity CMS Schema with API Data Structure

I carefully cross-checked the field names and types in the Sanity CMS schema with the API data structure. Adjustments were made to align the schema fields for compatibility.

Example:

- **API Field:** `product_title`
- **Schema Field:** `name`
- **Action:** Updated the schema to match or map the fields during integration.

3. Data Migration Process

Methods Used for Data Migration

To streamline the migration process, I applied the following method:

Using the Provided API:

1. I wrote the `importData.js` script to fetch and transform data from the API.
2. Ran the script using Node.js to populate Sanity CMS with the fetched data.

Example Script:

```
const fetch = require('node-fetch');
```

```
const sanityClient = require('@sanity/client');

const client = sanityClient({
  projectId: 'myId',
  dataset: 'production',
  useCdn: false,
});

const importData = async () => {
  const response = await fetch('https://api.template-6/products');
  const products = await response.json();

  for (const product of products) {
    await client.createOrReplace({
      _type: 'product',
      _id: product.id,
      name: product.title,
      description: product.description,
      mainImage: {
        asset: { _ref: product.imageRef },
      },
    });
  }
};

importData().then(() => console.log('Data Imported Successfully')).catch(console.error);
```

Best Practices Applied:

- Backed up the Sanity project before importing large datasets.
- Tested the imported data for accuracy and schema alignment.
- Included validation checks in the script to handle errors gracefully.

4. API Integration in Next.js

Step-by-Step Integration Process

Step 1: Creating Utility Functions

Utility functions were written to fetch data from Sanity CMS.

Example:

```
import { client } from '../lib/sanityClient';

export const fetchProducts = async () => {
  const query = `*[_type == "product"]{_id, title, description, mainImage}`;
  return await client.fetch(query);
```

```
};
```

Step 2: Rendering Data in Components

The fetched data was rendered into a responsive frontend using Next.js and Tailwind CSS.

Example:

```
const ProductsPage = async () => {
  const products = await fetchProducts();
  return (
    <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
      {products.map((product) => (
        <div key={product._id} className="border rounded shadow">
          <img src={product.mainImage.asset.url} alt={product.title} />
          <h2>{product.title}</h2>
          <p>{product.description}</p>
        </div>
      ))}
    </div>
  );
};
export default ProductsPage;
```

Step 3: Testing API Integration

- Used Postman and browser developer tools to test API endpoints.
- Logged responses to ensure consistency and debug any issues.

Error Handling Techniques

- Centralized error logging for debugging purposes.
- Displayed user-friendly error messages in the UI.
- Implemented fallback data and skeleton loaders for better user experience.

Final Outcome

Sanity CMS

- Successfully populated with data imported from the provided API using the `importData.js` script.

Next.js Frontend

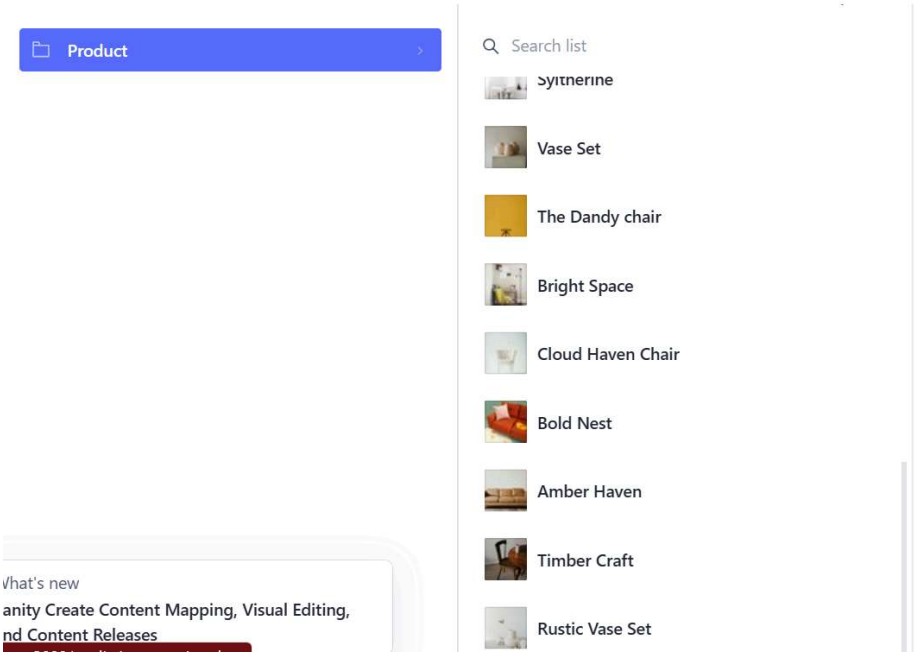
- Fully functional integration displaying:
    - Product listings.
    - Categories.

- Other relevant data.

Screenshots and Code Snippets

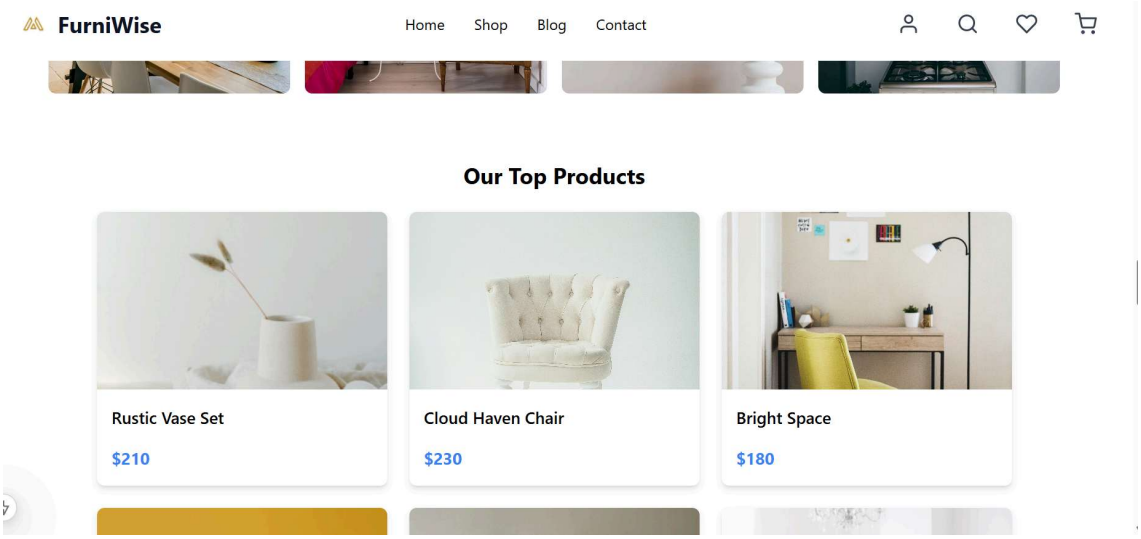
Screenshot 1:

- Sanity CMS Dashboard populated with API data.



Screenshot 2:

- Rendered Products Page in Next.js.



Summary

- Reviewed and understood API documentation.
- Adjusted Sanity CMS schema for compatibility.
- Populated Sanity CMS via scripts.
- Integrated data into the Next.js frontend.
- Successfully displayed API data in a responsive and functional UI.

This document reflects my hands-on experience and serves as a guide for future API integration and data migration tasks.