Day 3

API Integration Report - Comforty

1. Introduction

This report outlines the steps taken to integrate Sanity CMS with the Comforty chair marketplace. The goal was to enhance product data management by using Sanity's CMS system.

The process involved setting up schemas, configuring environment variables, linking Sanity with external APIs, and rendering the data on the frontend.

2. API Integration Process

2.1 Sanity Schema Creation

To structure the data efficiently, two schema files—product.ts and categories.ts—were created. These schemas are responsible for defining the structure of product and category documents in Sanity CMS.

2.2 Product Schema (product.ts):

Fields: title, price, priceWithoutDiscount, badge, image, category, description, inventory, and tags.

Tags include options like "Featured", "Instagram", "Gallery", and "Detail" to categorize products.

2.3 Category Schema (categories.ts):

Fields: title, image, and products.

This schema handles product categories and their respective images and product counts.

3. Data Migration and Environment Setup

3.1 Environment Variables

Environment variables were configured in the .env.local file to securely store the Sanity project ID, dataset, and authentication token. These variables are essential for connecting the frontend to the Sanity API.

Example .env.local file:

```
NEXT_PUBLIC_SANITY_PROJECT_ID=""

NEXT_PUBLIC_SANITY_DATASET=""

NEXT_PUBLIC_SANITY_AUTH_TOKEN=""
```

3.2 Migration Script

To automate the migration of data from external APIs to Sanity, a migration script was created in the scripts folder as migrate.ts. The script links the API data to the Sanity CMS and updates the respective product and category entries.

Once the Sanity schema and migration were set up, product data was fetched from the Sanity API and displayed on the frontend. The following function fetches the product details:

4. Fetching and Displaying Data on Frontend

4.1 Product Data Fetching Query (Sanity Script)

To fetch product data from Sanity, the following query was used:

```
export async function GetProductData() {
return sanityClient.fetch(
groq`
*[_type == "products"] {
_id,
title,
price,
priceWithoutDiscount,
badge,
"imageURL": image.asset->url,
category->{ _id, title },
description,
inventory,
tags
```

```
}`
)
}
```

4.2 Featured Products Query (Sanity Script)

To fetch products with the "Featured" tag, the following query was used:

```
export async function GetFeaturedProducts() {
return sanityClient.fetch(
groq`
*[_type == "products" && "featured" in tags] {
_id,
title,
price,
priceWithoutDiscount,
badge,
"imageURL": image.asset->url,
category->{ _id, title },
description,
inventory,
tags
}`
)
```

```
}
```

4.3 Categories Data Fetching Query (Sanity Script)

To fetch category data, the following query was used:

```
export async function GetCategoriesData() {
return sanityClient.fetch(
groq`
*[_type == "categories"] {
title,
products,
"imageURL": image.asset->url,
}`
)
}
```

5. Displaying Data on Frontend

Once the data is fetched from Sanity, it is displayed on the frontend using the following methods:

5.1 Displaying Product Data on Frontend:

```
const productData = await GetProductData();
{productData.map((item) => (
```

```
{item.title}

))}
```

5.2 Displaying Featured Products on Frontend:

```
const featuredData = await GetFeaturedProducts();

{featuredData.map((item) => (

{item.title}

))}
```

5.3 Displaying Categories Data on Frontend:

```
const categoriesData = await GetCategoriesData();

{categoriesData.map((category) => (

{category.title}

))}
```

5. Conclusion

The integration of Sanity CMS with the Comforty Marketplace was successfully completed. The schemas for products and categories were structured to handle data efficiently. The migration process allowed for the seamless transfer of product and category data to Sanity. On the frontend, data was dynamically rendered based on various queries, including those for featured products, Instagram products, and category-based filtering.

This process not only automated the data migration but also ensured that the marketplace was up-to-date with product information. Further optimization and enhancements can be added by refining queries and improving the frontend display.