PREPARED BY UMER AMIN
DAY 3 - API INTEGRATION AND DATA MIGRATION

---

1. Introduction

This report outlines the steps taken to integrate Sanity CMS with my marketplace project, focusing on improving product data management using Sanity's CMS. 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 efficiently structure the data, two schema files—`product.ts` and `category.ts`—were created. These schemas define the structure for 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 (`category.ts`)

Fields: title, image, and products.
This schema manages product categories and their respective images and product counts.

---

3. Data Migration and Environment Setup

3.1 Environment Variables

The necessary environment variables were set in the `.env.local` file to securely store the Sanity project ID, dataset, and authentication token. These variables enable frontend connectivity to the Sanity API.

Example `.env.local` file:

```
makefile
CopyEdit
NEXT_PUBLIC_SANITY_PROJECT_ID=
```

```
NEXT_PUBLIC_SANITY_DATASET=
NEXT_PUBLIC_SANITY_AUTH_TOKEN=
```

3.2 Migration Script

A migration script was created to automate data transfer from external APIs to Sanity. The script, named `migrate.ts`, links API data to the appropriate product and category entries in Sanity.

---

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)

For fetching products tagged as "Featured", 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 dynamically displayed on the frontend.

```
const productData = await GetProductData();
{productData.map((item) => (
  <div key={item._id}>{item.title}</div>
))}
```

5.2 Displaying Featured Products on Frontend:

```
const featuredData = await GetFeaturedProducts();
{featuredData.map((item) => (
  <div key={item._id}>{item.title}</div>
))}
```

5.3 Displaying Categories Data on Frontend:

```
const categoriesData = await GetCategoriesData();
{categoriesData.map((category) => (
  <div key={category._id}>{category.title}</div>
))}
```

---

6. Conclusion

The integration of Sanity CMS with the marketplace was successfully completed. The product and category schemas were structured to handle the data efficiently. The migration process allowed for the seamless transfer of data from the external API to Sanity. On the frontend, data was dynamically rendered, ensuring up-to-date product information is displayed.

Future optimizations can be made by refining queries for better performance and enhancing the frontend layout for a more polished user experience.