Day 3 – Hackathon Documentation

Kashaf Zeeshan (00461288) | Friday Morning Slot (Sir Hamzah Syed) | Template 08 (Comforty)

API Integration & Data Migration for Comforty

1.Sanity Schema: Created `productSchema` & `categorySchema`, added slug for dynamic routing.

2.API Migration: Imported & structured data to match schemas.

3.Frontend: Integrated API data, ensured dynamic rendering.

Data Migration to Sanity

For the **Comforty Marketplace Plan**, I implemented API integration and **data migration** from a REST API to **Sanity CMS**. This process ensured that products and categories were correctly structured and stored in the database for dynamic frontend rendering.

Overview of the Migration Process

The migration script:

1. **Fetched categories and products** from the provided API.

2. **Uploaded images** to Sanity's asset library.

3. **Mapped category IDs** to maintain relationships between products and their categories.

4. **Stored structured data** into Sanity, ensuring compatibility with the existing frontend.

```
// Import environment variables from .env.local
import "dotenv/config";

// Import the Sanity client to interact with the Sanity backend
import { createClient } from "@sanity/client";

// Load required environment variables
const {
  NEXT_PUBLIC_SANITY_PROJECT_ID, // Sanity project ID
  NEXT_PUBLIC_SANITY_DATASET, // Sanity dataset (e.g., "production")
  NEXT_PUBLIC_SANITY_AUTH_TOKEN, // Sanity API token
  BASE_URL = "https://gialc-hackathon-template-08.vercel.app", // API base URL for products and categories
} = process.env;

// Check if the required environment variables are provided
if (!NEXT_PUBLIC_SANITY_PROJECT_ID || !NEXT_PUBLIC_SANITY_AUTH_TOKEN) {
  console.error("Missing required environment variables. Please check your .env.local file.");
  process.exit(1); // Stop execution if variables are missing
}

// Create a Sanity client instance to interact with the target Sanity dataset
const targetClient = createClient({
  projectId: NEXT_PUBLIC_SANITY_PROJECT_ID, // Your Sanity project ID
  dataset: NEXT_PUBLIC_SANITY_DATASET || "production", // Default to "production" if not set
  useCdn: false, // Disable CDN for real-time updates
  apiVersion: "2023-01-01", // Sanity API version
  token: NEXT_PUBLIC_SANITY_AUTH_TOKEN, // API token for authentication
});

// Function to upload an image to Sanity
async function uploadImageToSanity(imageUrl) {
  try {
    // Fetch the image from the provided URL
    const response = await fetch(imageUrl);
    if (!response.ok) throw new Error(`Failed to fetch image: ${imageUrl}`);

    // Convert the image to a buffer (binary format)
    const buffer = await response.arrayBuffer();

    // Upload the image to Sanity and get its asset ID
    const uploadedAsset = await targetClient.assets.upload("image", Buffer.from(buffer), {
      filename: imageUrl.split("/").pop(), // Use the file name from the URL
    });

    return uploadedAsset._id; // Return the asset ID
  } catch (error) {
    console.error("Error uploading image:", error.message);
    return null; // Return null if the upload fails
 }
}

// Main function to migrate data from REST API to Sanity
async function migrateData() {
  console.log("Starting data migration...");

  try {
    // Fetch categories from the REST API
    const categoriesResponse = await fetch(`${BASE_URL}/api/categories`);
    if (!categoriesResponse.ok) throw new Error("Failed to fetch categories.");
    const categoriesData = await categoriesResponse.json(); // Parse response to JSON

    // Fetch products from the REST API
    const productsResponse = await fetch(`${BASE_URL}/api/products`);
    if (!productsResponse.ok) throw new Error("Failed to fetch products.");
    const productsData = await productsResponse.json(); // Parse response to JSON

    const categoryIdMap = {}; // Map to store migrated category IDs

    // Migrate categories
    for (const category of categoriesData) {
      console.log(`Migrating category: ${category.title}`);
      const imageId = await uploadImageToSanity(category.imageUrl); // Upload category image

      // Prepare the new category object
      const newCategory = {
        _id: category._id, // Use the same ID for reference mapping
        type: "categories",
```

Sanity Schema for Products

To structure and store **product data**, I created a **Sanity schema** named `productSchema`. This schema defines the fields required for each product, ensuring consistency in the database

```
import { defineType } from "sanity";

export const productSchema = defineType({
  name: "products",
  title: "Products",
  type: "document",
  fields: [
   {
      name: "title",
      title: "Product Title",
      type: "string",
    },
   {
      name: "price",
      title: "Price",
      type: "number",
    },
   {
        name: "slug",
        title: "Slug",
        type: "slug",
        options: {
          source: "title",
          maxLength: 200,
        },
      },
   {
      title: "Price without Discount",
      name: "priceWithoutDiscount",
      type: "number",
    },
   {
      name: "badge",
      title: "Badge",
      type: "string",
    },
   {
      name: "image",
      title: "Product Image",
      type: "image",
    },
   {
      name: "category",
      title: "Category",
      type: "reference",
      to: [{ type: "categories" }],
    },
   {
```

Product Schema Fields (Sanity):

1. **title** – Stores the product name.
2. **price** – Stores the current price of the product.
3. **slug** – Generates an SEO-friendly URL.
4. **priceWithoutDiscount** – Stores the original price before any discount.
5. **badge** – Stores a special label for the product (e.g., "New").
6. **image** – Stores the product image.
7. **category** – Links the product to a specific category.
8. **description** – Stores a detailed description of the product.
9. **inventory** – Defines the available stock for the product.

10. **tags** – Stores tags for product categorization and filtering.

Category Schema (Sanity)

The categorySchema defines the structure for storing category-related data in Sanity. It includes fields for the category title, an image to represent the category, and a numeric field to track the number of products within each category. This schema ensures organized and easily accessible category information.


import { defineType } from "sanity";
export const categorySchema = defineType
({ name: 'categories',
title: 'Categories',
type: 'document',
fields: [
{
name: 'title',
title: 'Category Title',
type: 'string',
},
{
name: 'image',
title: 'Category Image',
type: 'image',
},
{
title: 'Number of Products',
name: 'products',
type: 'number',
}
],
});

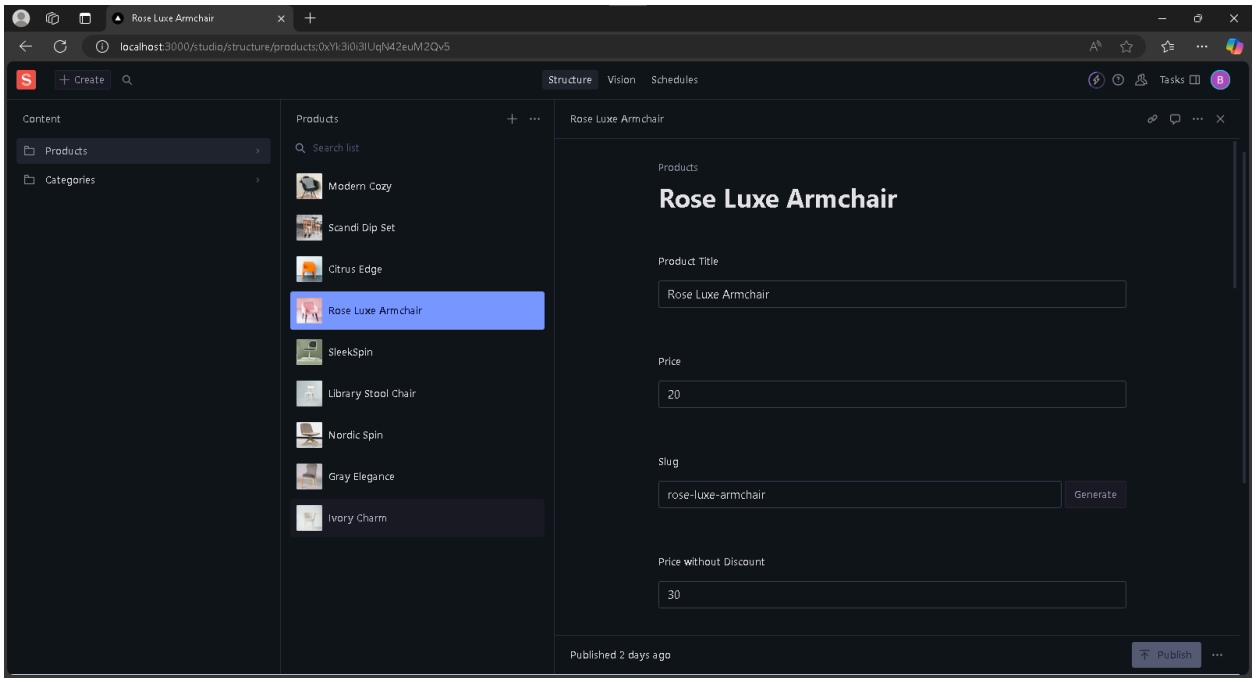
Fields:

1. **title** – Stores the category name.
2. **image** – Stores the category image.
3. **products** – Stores the number of products in the category.

Sanity Studio - Imported Data Preview:

Below is a screenshot of the Sanity Studio interface, displaying the successfully imported categories and products. This confirms that the API data migration was

executed properly, and the structured schemas are now populated with relevant information.

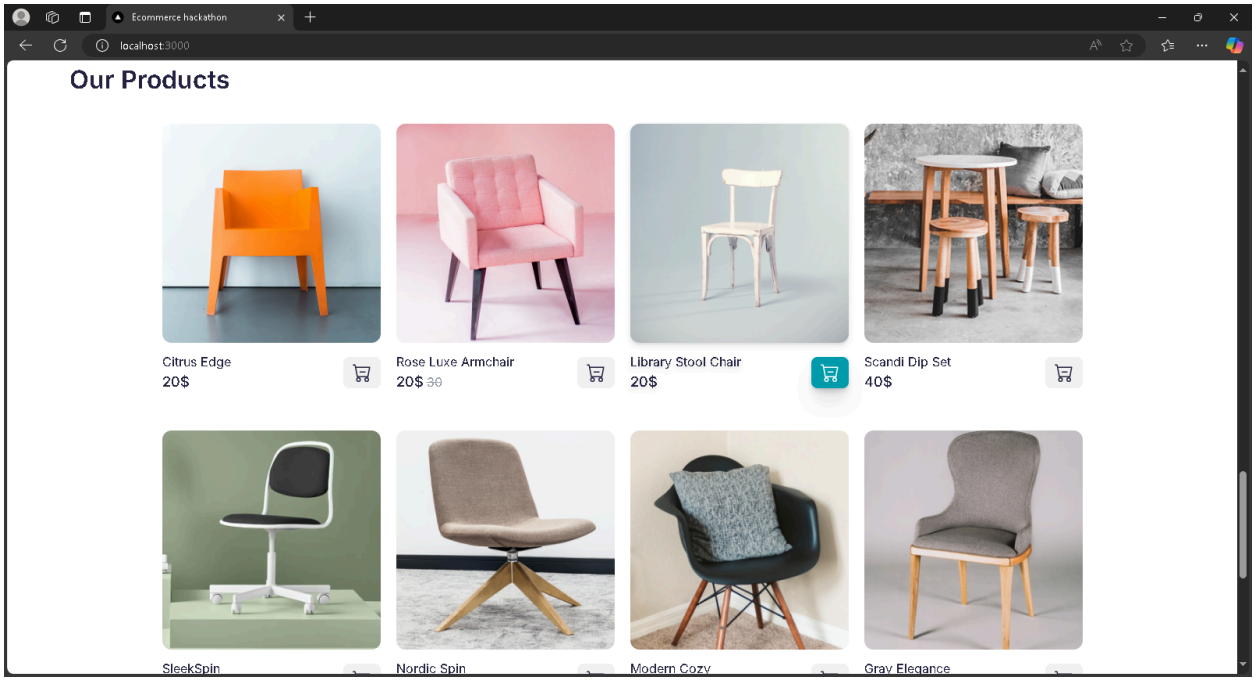


Fetching Data from GROQ in Sanity

```tsx

"use client";

import { useState, useEffect } from "react";
import { client } from "@/sanity/lib/client";
import Link from "next/link";
import { BsCartDash } from "react-icons/bs";

const OurProduct = () => {
  const [products, setProducts] = useState<any[]>([]);

  useEffect(() => {
    const fetchProducts = async () => {
      const data = await client.fetch(`*[_type == "products"][0...8]{
        "slug":slug.current,
        title,
        price,
        priceWithoutDiscount,
        badge,
        color,
        "imageUrl": image.asset->url,
      }`);
      setProducts(data);
    };

    fetchProducts();
  }, []);

  return (
    <div className="lg:mx-20 sm:mx-10 mx-3 mb-20 lg:mb-40">
      <div>
        <h2 className="text-[#272343] lg:text-[32px] text-2xl md:text-left text-center tracking-normal font-semibold">
          Our Products
        </h2>
        <div className="products flex flex-wrap justify-center lg:mt-10 mt-3 gap-5 gap-y-20 lg:gap-y-28">
          {products.map((prod) => (
            <div key={prod.title} className="w-[280px] h-[280px] cursor-pointer hover:drop-shadow-md">
              <div className="img relative">
                <Link href={`/product/${prod.slug}`}>
                  <img
                    src={prod.imageUrl}
                    alt={prod.title}
                    className="w-full h-full object-cover rounded-xl"
                  />
                </Link>
                {prod.color && (
                  <span
                    className="absolute top-4 left-4 rounded-lg text-sm px-3 py-1"
                    style={{ backgroundColor: prod.color, color: "white" }}
>
                    {prod.badge}
                  </span>
                )}
              </div>
              <div className="flex items-center justify-between mt-3">
                <div className="text gap-[10px]">
                  <h4 className="text-[#272343]">{prod.title}</h4>
                  <div>
                    <span className="text-[#272343] text-[18px] font-medium">{prod.price}$</span>
                    <del className="ml-1 text-[#9A9CAA]">{prod.priceWithoutDiscount}</del>
                  </div>
                </div>
                <div className="cart px-3 py-2 bg-[#F0F2F3] hover:shadow-xl hover:bg-[#029FAE] text-[#272343] hover:text-white rou
nded-lg">
                  <Link href="/carts">
                    <BsCartDash className="size-6" />
                  </Link>
                </div>
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
};

export default OurProduct;

```

In this component, we are using **GROQ queries** to fetch product data from Sanity. The fetched data includes the **slug, title, price, discounted price, badge, color, and image URL** of each product. The products are then displayed in a responsive grid layout with individual cards, each containing an image, title, price details, and an add-to-cart button. The `useEffect` hook ensures that the data is fetched when the component mounts.

Frontend View



Data from Sanity is fetched and displayed on the frontend. Each product's title, price, image, and badge are dynamically rendered in a responsive grid layout, ensuring a smooth user experience.

Conclusion:

We started by migrating our product data to **Sanity CMS** using schemas for structured storage. The `productSchema` and `categorySchema` defined how data is stored, including fields like **title, price, images, slug, and categories**.

After migration, the data was accessible in **Sanity Studio**, where we verified the imported content.

Next, we used **GROQ queries** in our **Next.js frontend** to fetch the data from Sanity. Using `client.fetch()`, we retrieved **products with their images, prices, and badges**, dynamically displaying them on the page.