Day 3 - API Integration Report - Comforty Chair Marketplace

Introduction

This report outlines the process of integrating APIs, making necessary adjustments to the schemas, and migrating data to the Sanity CMS for **Comforty Chair Marketplace**. The goal was to enable seamless API interaction, integrate external product data, and ensure the CMS is populated and functional.

API Integration Process

1. API Integration

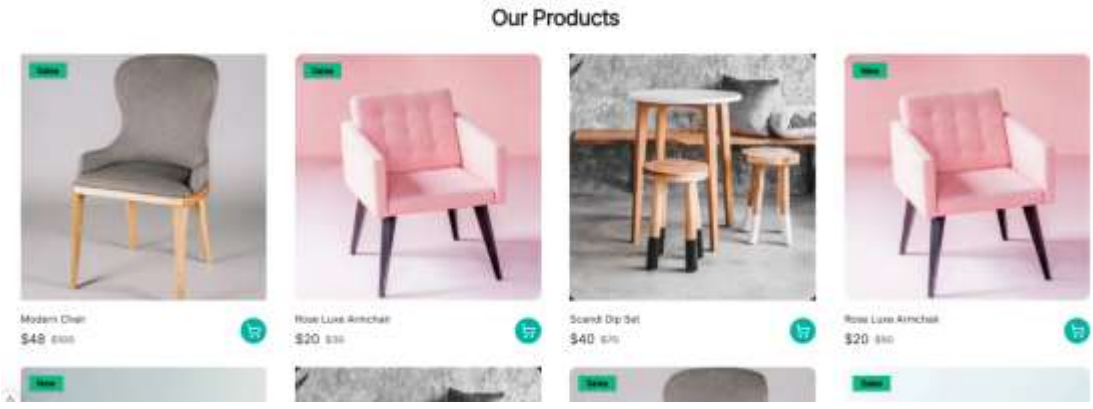
The API integration involved fetching product data from an external source and populating it in Sanity CMS. The following steps were followed:

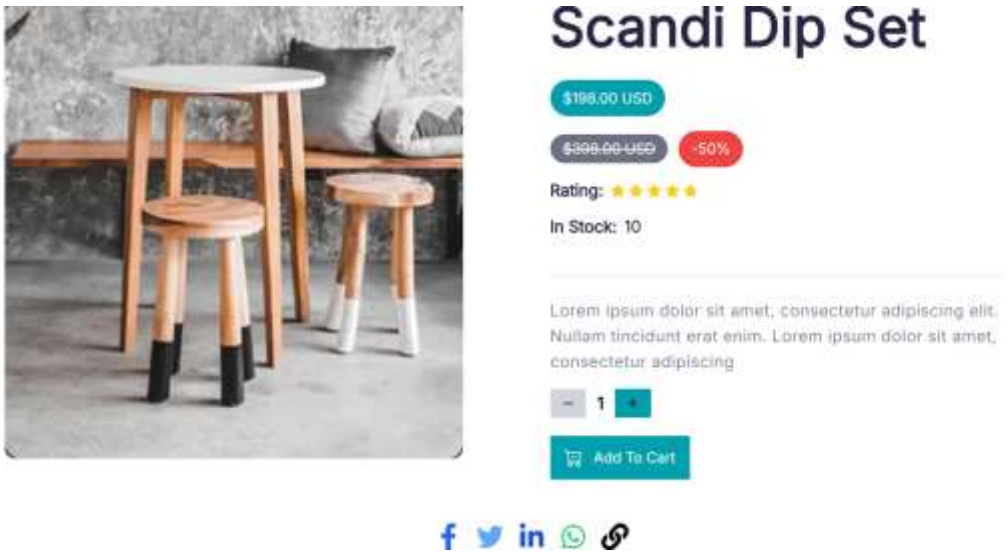
1. **Identifying the API Endpoints**:
    1. The external API that provides product data was identified, including endpoints for product information, images, and categories.
2. **Creating API Fetch Functions**:
    1. We created a fetch function using `fetch()` or an alternative like `axios` to retrieve product data from the external API.
3. **Integrating API Calls in the Project**:
    1. API calls were made during the data fetching process in Next.js. The API was connected to the frontend and used in pages like Product Detail.

```
useEffect(() => {
  const fetchProducts = async () => {
    try {
      const query = `*[_type == "products"] {
      _id,
      title,
      originalPrice,
      discountedPrice,
      "image": image.asset->url,  // Fetch URL from Sanity directly
      isNew,
      isSale
    }`;

      const data: {
        _id: string;
        title: string;
        originalPrice?: number;
        discountedPrice?: number;
        image: string;
        isNew?: boolean;
        isSale?: boolean;
      }[] = await client.fetch(query);
      console.log("Fetched products:", data);

      const formattedProducts = data.map((product) => ({
        id: product._id, // Use _id from Sanity
        title: product.title,
        originalPrice: product.originalPrice,
        discountedPrice: product.discountedPrice,
        image: product.image, // This is already the URL string
        isNew: product.isNew,
        isSale: product.isSale,
      }));

      setProducts(formattedProducts); // Set state to properly typed Product[]
    } catch (err) {
      console.error("Error fetching products:", err);
```

Frontend Display of API Data:

- Once the data is fetched, it is displayed in the frontend. For example, on the Product Detail page, we display the product details like title, price, description, and inventory status fetched from the API.

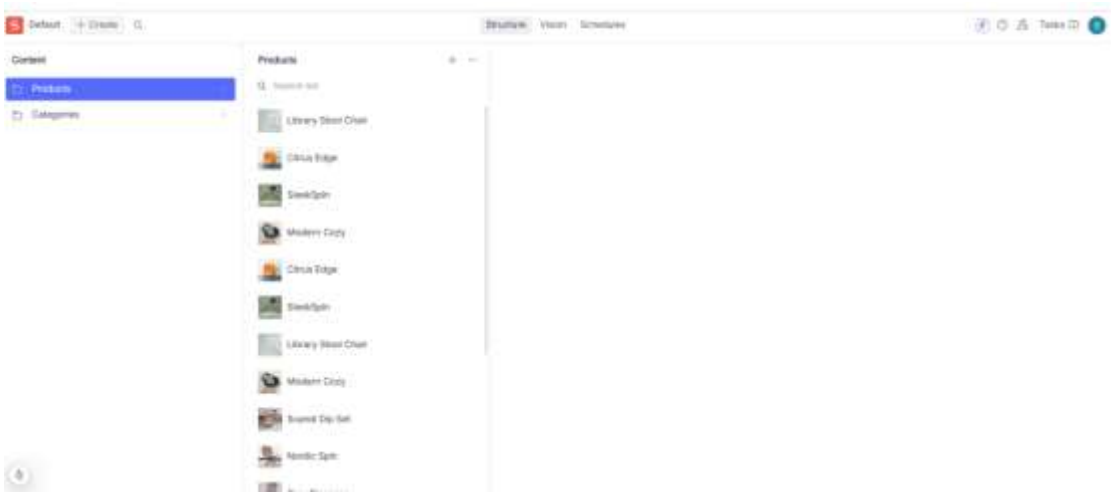




- Handling Data:
    - Upon successful API calls, product data was parsed and displayed in the frontend. The product details were rendered on the Product Detail page.
- Error Handling:
    - A fallback mechanism was implemented to handle any errors during the API fetch process (e.g., when the product isn't found).

```
    if (!response.ok) {
      throw new Error("Failed to fetch products from Sanity");
    }

    const { result } = await response.json();

    setProducts(result);
  } catch (err) {
    console.error("Error fetching products:", err);
    setError((err as Error).message);
  } finally {
    setLoading(false);
  }
};
```

Adjustments Made to Schemas



2. Schema Adjustments

To integrate and store data effectively, the following adjustments were made to the Sanity CMS schemas:

1. Product Schema:

New fields were added to the `product` schema to handle API data, such as `priceWithoutDiscount`, `tags`, and `rating`.

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

export const productSchema = defineType({
  name: "products",
  title: "Products",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Product Title",
      type: "string",
    },
    {
      title: "Price without Discount",
      name: "priceWithoutDiscount",
      type: "number",
    },
    {
      title: "Discounted Price",
      name: "discountedPrice",
      type: "number",
      description: "The price after applying discounts (if any).",
    },
    {
      title: "Discount Percentage",
      name: "discountPercentage",
      type: "number",
      description: "Automatically calculated based on the price difference.",
      readOnly: true, // Prevent manual entry
      options: {
        computed: {
          function: (document: any) => {
            if (document.priceWithoutDiscount && document.discountedPrice) {
              return Math.round(
                ((document.priceWithoutDiscount - document.discountedPrice) /
                  document.priceWithoutDiscount) *
                  100
              );
            }
```

```
fields: [
    options: {
      computed: {
        function: (document: any) => {
          },
        },
      },
    },
  },
  {
    title: "Ratings",
    name: "ratings",
    type: "number",
    description: "Product rating on a scale of 1 to 5.",
    validation: (Rule) =>
      Rule.min(1)
        .max(5)
        .precision(1)
        .error("Rating must be between 1 and 5."),
  },
  {
    name: "badge",
    title: "Badge",
    type: "string",
  },
  {
    name: "image",
    title: "Product Image",
    type: "image",
  },
  {
    name: "category",
    title: "Category",
    type: "reference",
    to: [{ type: "categories" }],
  },
  {
    name: "description",
    title: "Product Description",
    type: "text",
  },
  {
    name: "inventory",
    title: "Inventory Management",
    type: "number",
  },
  {
    name: "tags",
    title: "Tags",
    type: "array",
    of: [{ type: "string" }],
    options: {
      list: [
        { title: "Featured", value: "featured" },
        {
          title: "Follow products and discounts on Instagram",
          value: "instagram",
        },
        { title: "Gallery", value: "gallery" },
```

2. Category Schema:

A category reference was added in the product schema to link products to specific categories.

```
export default {
  name: "categories",
  type: "document",
  title: "Categories",
  fields: [
    { name: "title", type: "string", title: "Title" },
    { name: "image", type: "image", title: "Image" },
  ],
};
```

3.Image Handling:

- The product images fetched via the API were connected with Sanity's image asset reference field to ensure proper handling of image data.

Migration Steps and Tools Used

3. Data Migration

The product data was migrated from the external API into the Sanity CMS using the following steps:

Preparing the Migration Script:

1. A Node.js script was written to automate the migration process. This script used Sanity's client to create documents in the CMS with data from the external API.

Script to Migrate Product Data:

1. The script iterated over the product data and pushed it to Sanity's CMS using the following code:

```
// Import environment variables from .env.local
import dotenv from "dotenv";
dotenv.config({ path: ".env.local" });

// 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://giaic-hackathon-template-08.vercel.app", // API base
} = process.env;
```

```
// Debugging environment variables
console.log("Loaded environment variables:", {
  NEXT_PUBLIC_SANITY_PROJECT_ID,
  NEXT_PUBLIC_SANITY_DATASET,
  NEXT_PUBLIC_SANITY_AUTH_TOKEN,
  BASE_URL,
});

// 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 f
  );
  process.exit(1); // Stop execution if variables are missing
}
```

```
// Create a Sanity client instance
const targetClient = createClient({
  projectId: NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: NEXT_PUBLIC_SANITY_DATASET || "production",
  useCdn: false, // Disable CDN for real-time updates
  apiVersion: "2023-01-01", // API version
  token: NEXT_PUBLIC_SANITY_AUTH_TOKEN, // API token
});

// Function to upload an image to Sanity
Qodo Gen: Options | Test this function
async function uploadImageToSanity(imageUrl) {
  try {
    const response = await fetch(imageUrl);
    if (!response.ok) throw new Error(`Failed to fetch image: ${imageUrl}`);
    const buffer = await response.arrayBuffer();
    const uploadedAsset = await targetClient.assets.upload(
      "image",
      Buffer.from(buffer),
      {
        filename: imageUrl.split("/").pop(),
      }
    );
    return uploadedAsset._id;
  } catch (error) {
    console.error("Error uploading image:", error.message);
    return null;
  }
}
```

```
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();

    // 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();

    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);
      const newCategory = {
        _id: category._id,
        _type: "categories",
        title: category.title,
        image: imageId
          ? { _type: "image", asset: { _ref: imageId } }
          : undefined,
      };
      const result = await targetClient.createOrReplace(newCategory);
      categoryIdMap[category._id] = result._id;
      console.log(`Migrated category: ${category.title} (ID: ${result._id})`);
    }
```

```
async function migrateData() {
    // Migrate products
    for (const product of productsData) {
      console.log(`Migrating product: ${product.title}`);
      const imageId = await uploadImageToSanity(product.imageUrl);
      const newProduct = {
        _type: "products",
        title: product.title,
        price: product.price,
        priceWithoutDiscount: product.priceWithoutDiscount,
        badge: product.badge,
        image: imageId
          ? { _type: "image", asset: { _ref: imageId } }
          : undefined,
        category: {
          _type: "reference",
          _ref: categoryIdMap[product.category._id],
        },
        description: product.description,
        inventory: product.inventory,
        tags: product.tags,
      };
      const result = await targetClient.create(newProduct);
      console.log(`Migrated product: ${product.title} (ID: ${result._id})`);
    }

    console.log("Data migration completed successfully!");
  } catch (error) {
    console.error("Error during migration:", error.message);
    process.exit(1);
  }
```

```
    // Migrate products
    for (const product of productsData) {
      console.log(`Migrating product: ${product.title}`);
      const imageId = await uploadImageToSanity(product.imageUrl);
      const newProduct = {
        _type: "products",
        title: product.title,
        price: product.price,
        priceWithoutDiscount: product.priceWithoutDiscount,
        badge: product.badge,
        image: imageId
          ? { _type: "image", asset: { _ref: imageId } }
          : undefined,
        category: {
          _type: "reference",
          _ref: categoryIdMap[product.category._id],
        },
        description: product.description,
        inventory: product.inventory,
        tags: product.tags,
      };
      const result = await targetClient.create(newProduct);
      console.log(`Migrated product: ${product.title} (ID: ${result._id})`);
    }

    console.log("Data migration completed successfully!");
  } catch (error) {
    console.error("Error during migration:", error.message);
    process.exit(1);
  }
}

// Start the migration process
migrateData();
```

Sanity Studio:

1. Sanity Studio was used to verify that the data was correctly migrated and populated. We could check the `product` documents and their fields to ensure that all relevant information was stored.

Verification:

1. After migration, the data was verified in the Sanity Studio interface, ensuring that all product data, including pricing, description, and categories, were correctly added.

Screenshots

API Calls:

1. Screenshots of successful API calls made to fetch product data.

```
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const query = `*[_type == "products" && _id == $id][0] {
        _id,
        title,
        price,
        priceWithoutDiscount,
        "category": category->{
          _id,
          title
        },
        "imageUrl": image.asset->url,
        description,
        inventory,
        tags,
        badge,
        rating
      }`;

      const response = await client.fetch(query, { id });

      if (!response) {
        setError("Product not found!");
      } else {
        setProduct(response);
      }
    } catch (err) {
      setError("Error fetching product data.");
    } finally {
      setLoading(false);
    }
  };

  fetchProduct();
}, [id]);
```

Frontend Display:

1. A screenshot showing the data displayed on the frontend (e.g., product details page with information from the API).

Scandi Dip Set

$196.00 USD

~~$398.00 USD~~ -50%

Rating: ★★★★★

In Stock: 10

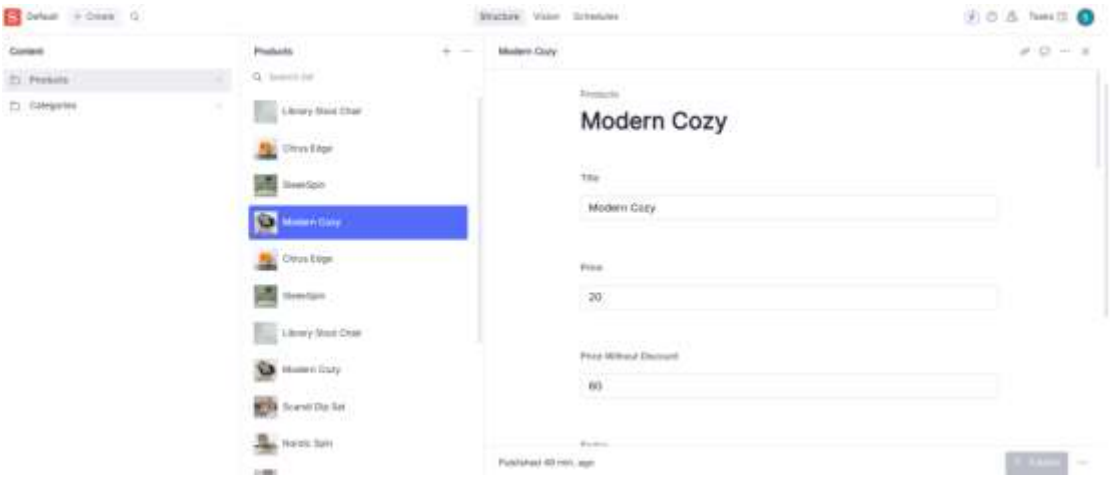
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing

− 1 +

Add To Cart

Populated Sanity CMS Fields:

1. A screenshot of Sanity Studio showing the product document with populated fields, such as price, description, and tags.



Conclusion

The API integration process was successfully completed, with the product data being fetched, stored, and displayed on the frontend. Necessary adjustments were made to the Sanity CMS schema to accommodate the new product information. Data migration was automated using a custom script, ensuring a smooth transition from the external API to the CMS.