Day 3 - API Integration Report - Comforty

Name: **Muskaan Fayyaz**

Roll No: **00216348**

API Integration and Sanity CMS Migration - Comforty

Objective:

The goal was to import third-party API data into Sanity CMS and ensure smooth data synchronization. By doing so, I gained full control over the data, allowing for operations like adding, updating, or deleting products, instead of relying solely on third-party libraries. This would enable dynamic content management within the "Comforty" e-commerce platform.

Steps Taken:

Part I: Set Up Environment

1. **Cloned the Repository**:
    - I cloned the project repository from GitHub using the following command:

      ```
      git clone <repository_url>
      ```

   Install Dependencies:

    - I navigated to the project folder and installed the necessary dependencies:

      ```
      npm install
      ```

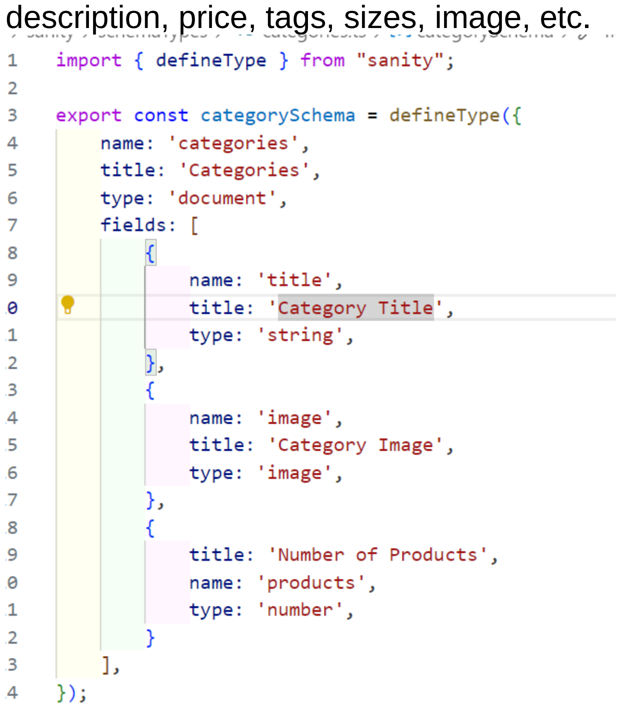
2. **Configured Sanity Environment Variables**:

    - Created a .env file and added necessary environment variables (Sanity project token, dataset, etc.).
    - Retrieved the API token from the Sanity dashboard by navigating to **API > Token**, adding a new token with Editor/Developer role, and saved it in the .env file.

3. **Compiled the Project**:
   - I compiled the project and ran the migration script:

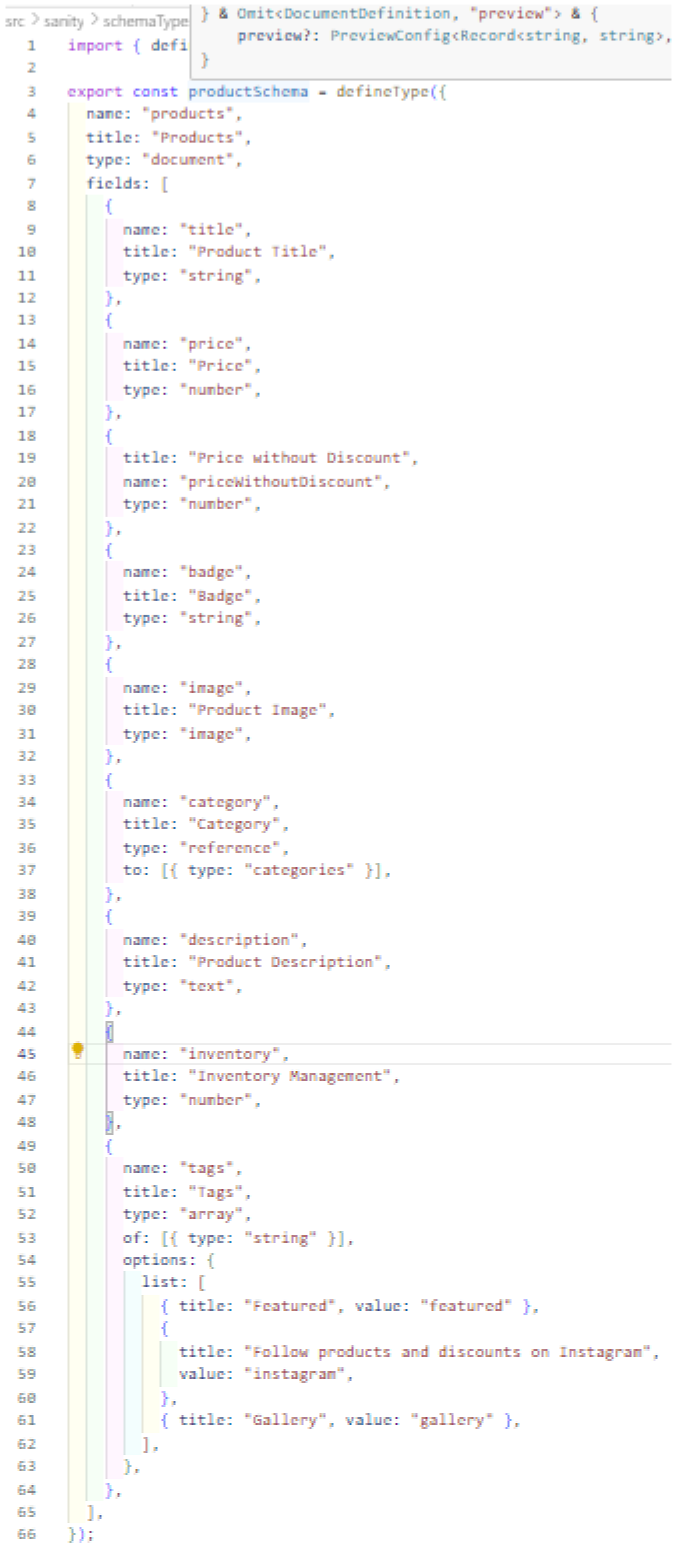
Part II: Defined Schema in Sanity

1. **Product Schema**:
   - Created a new schema file (product.ts) in the sanity/schemaTypes folder and defined the structure for the product data, including fields like name, description, price, tags, sizes, image, etc.



```
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',
        }
    ],
});
```

2. **Categories Schema**:



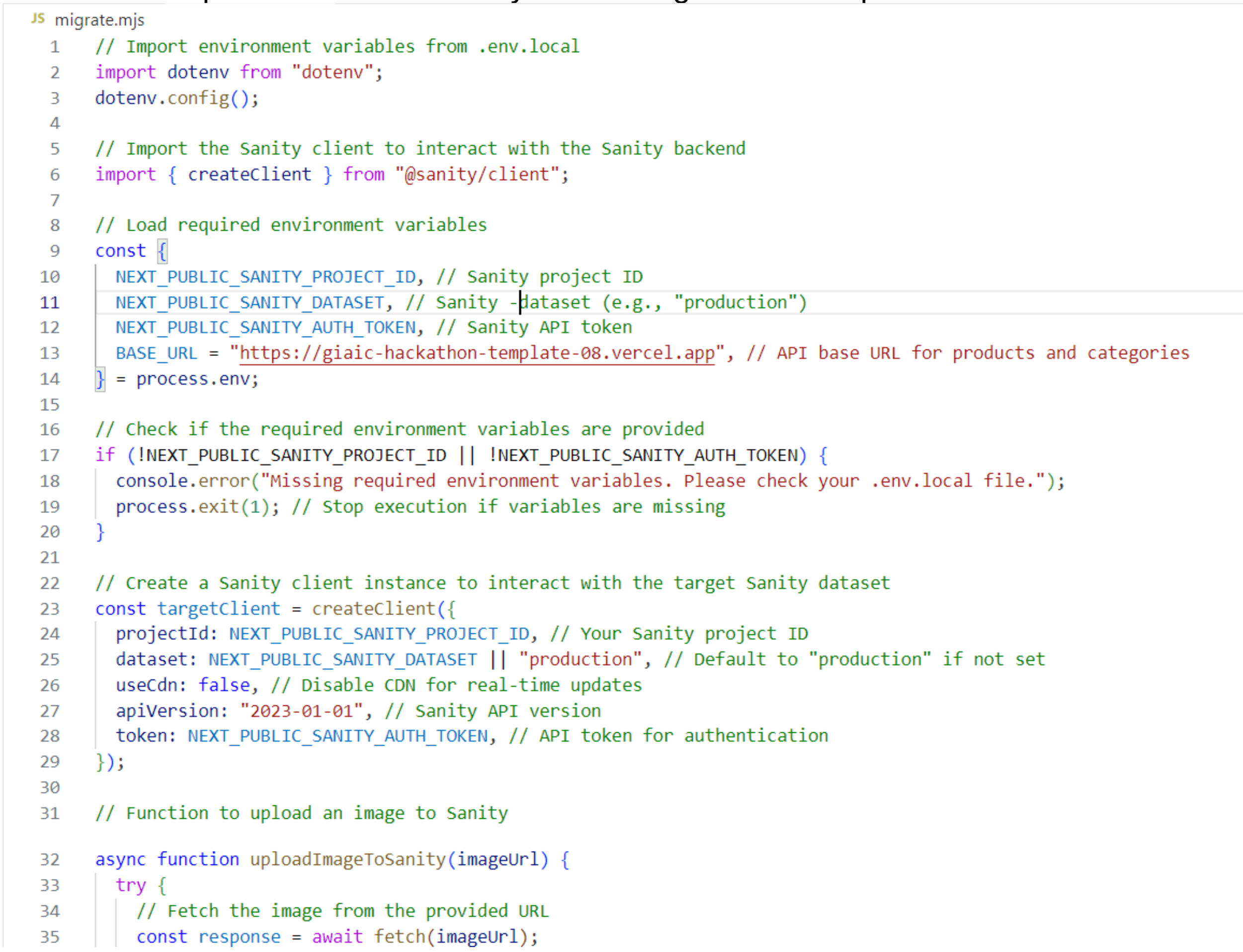
src > sanity > schemaType

```
} & Omit<DocumentDefinition, "preview"> & {
    preview?: PreviewConfig<Record<string, string>,
}
```

```
import { defi

export const productSchema = defineType({
  name: "products",
  title: "Products",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Product Title",
      type: "string",
    },
    {
      name: "price",
      title: "Price",
      type: "number",
    },
    {
      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" }],
    },
    {
      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" },
        ],
      },
    },
  ],
});
```

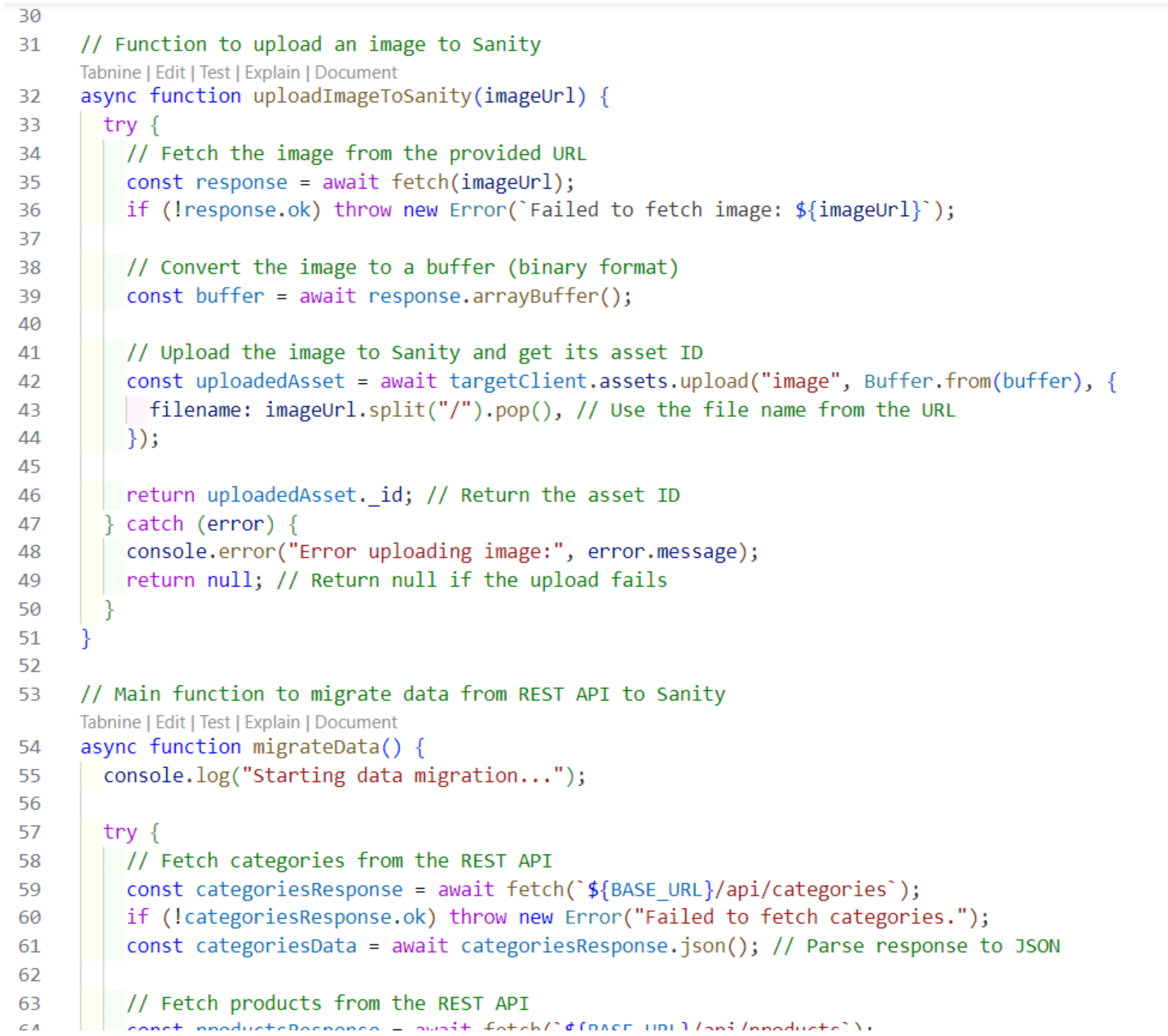
Part III: Fetch and Migrate Data

1. **Fetched Data from MockAPI.io**:
   - Used the provided MockAPI endpoints to fetch product data (name, description, price, images, etc.).
2. **Migrated Data to Sanity**:
   - Wrote a migration script to map the fetched data to the new Sanity schema and uploaded it to the Sanity CMS using a POST request

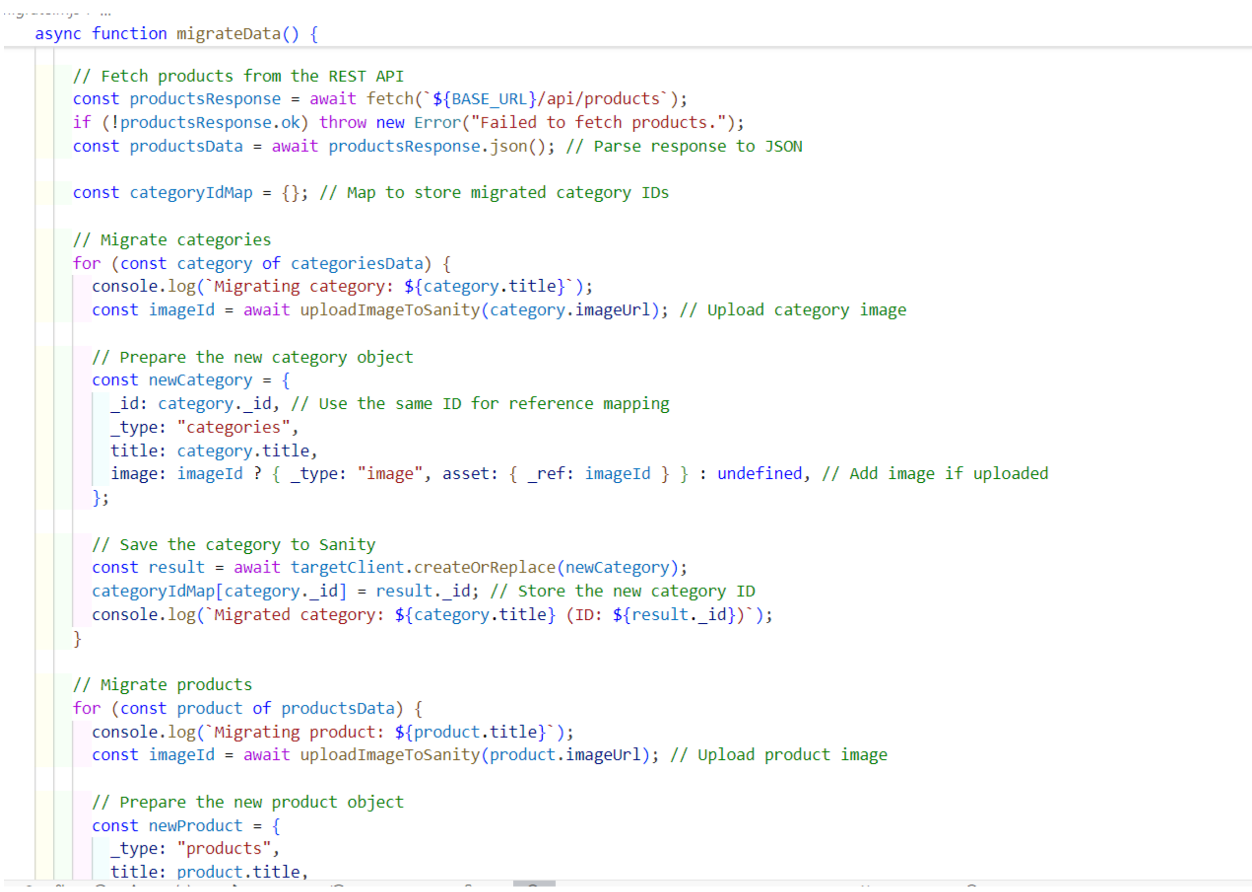


migrate.mjs

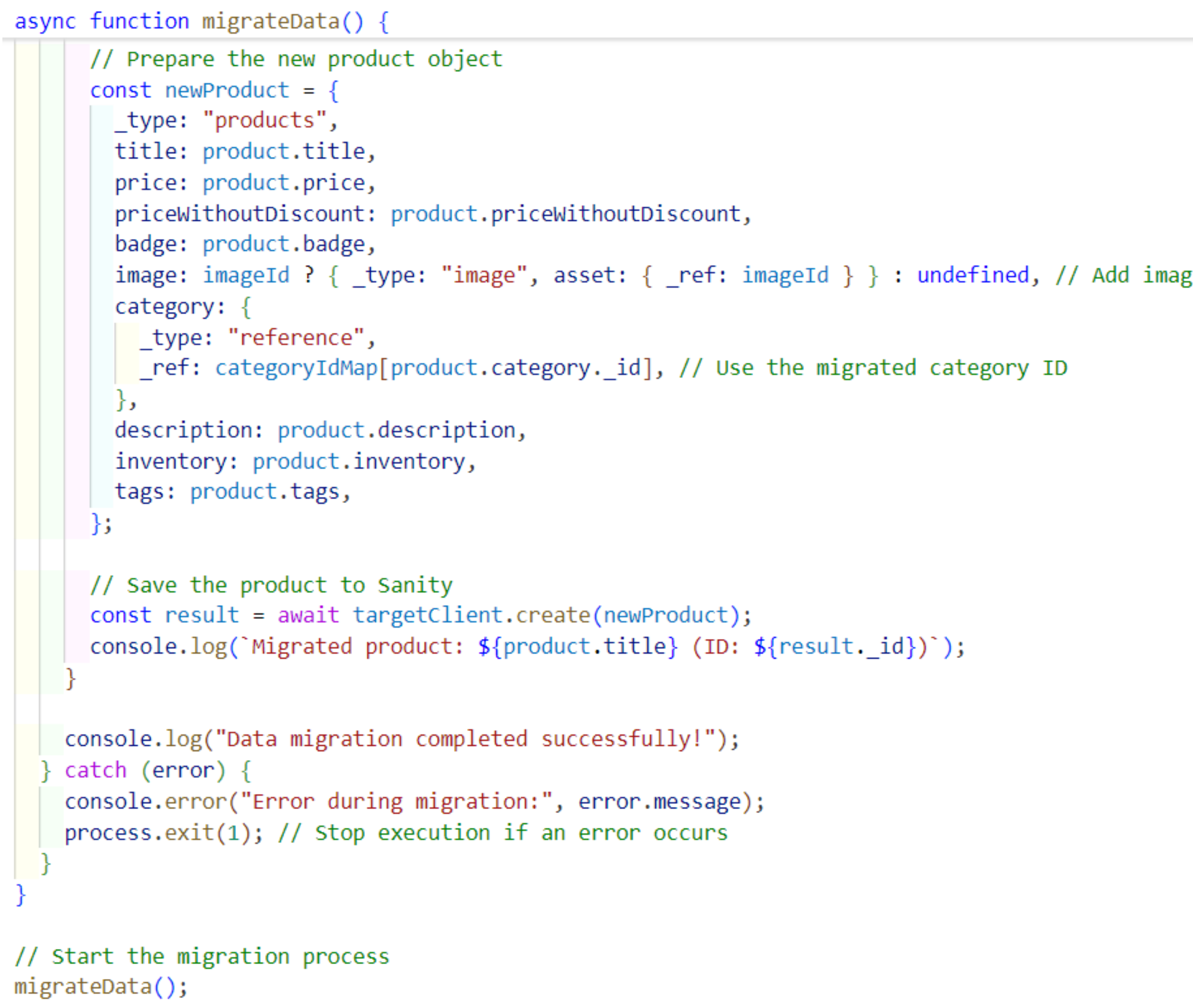
```javascript
// Import environment variables from .env.local
import dotenv from "dotenv";
dotenv.config();

// 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://giaic-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);
```



```
// Function to upload an image to Sanity
Tabnine | Edit | Test | Explain | Document
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
Tabnine | Edit | Test | Explain | Document
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`);
```



```javascript
async function migrateData() {

  // 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",
      title: category.title,
      image: imageId ? { _type: "image", asset: { _ref: imageId } } : undefined, // Add image if uploaded
    };

    // Save the category to Sanity
    const result = await targetClient.createOrReplace(newCategory);
    categoryIdMap[category._id] = result._id; // Store the new category ID
    console.log(`Migrated category: ${category.title} (ID: ${result._id})`);
  }

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

    // Prepare the new product object
    const newProduct = {
      _type: "products",
      title: product.title,
```



```
async function migrateData() {
      // Prepare the new product object
      const newProduct = {
        _type: "products",
        title: product.title,
        price: product.price,
        priceWithoutDiscount: product.priceWithoutDiscount,
        badge: product.badge,
        image: imageId ? { _type: "image", asset: { _ref: imageId } } : undefined, // Add imag
        category: {
          _type: "reference",
          _ref: categoryIdMap[product.category._id], // Use the migrated category ID
        },
        description: product.description,
        inventory: product.inventory,
        tags: product.tags,
      };

      // Save the product to Sanity
      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); // Stop execution if an error occurs
  }
}

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

.

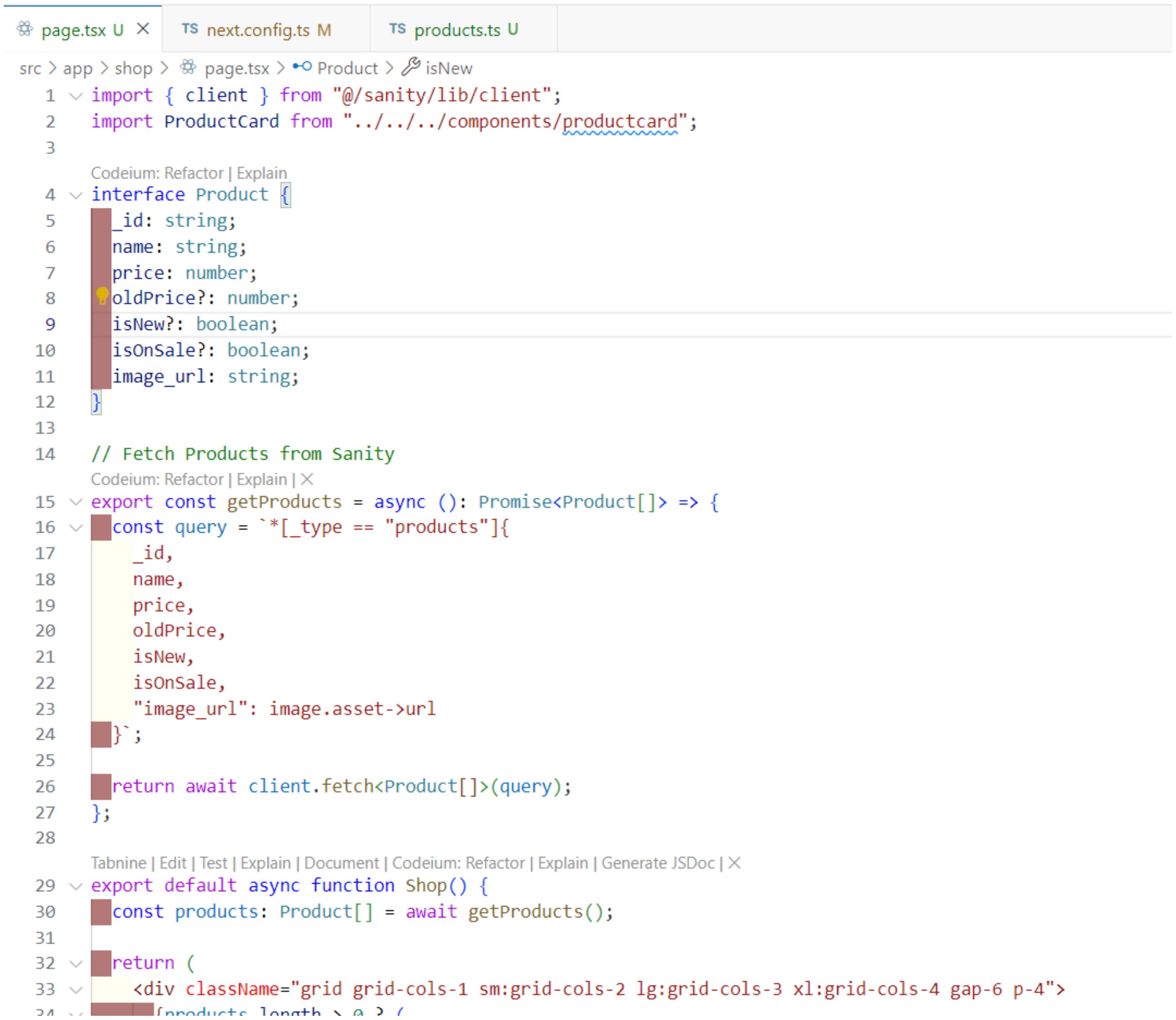
Part IV: Dynamic Content Fetching from Sanity

1. **Fetch Data Dynamically**:
   - I wrote a query to fetch dynamic product data based on tags such as "Featured" and "Best Sellers" using Sanity’s GROQ queries. This ensures that data can be filtered based on specific attributes, enabling the frontend to display relevant content.

```
const query = `*[_type == "products"]{
   _id,
   name,
   price,
   oldPrice,
   isNew,
   isOnSale,
   "image_url": image.asset->url
 }`;
```

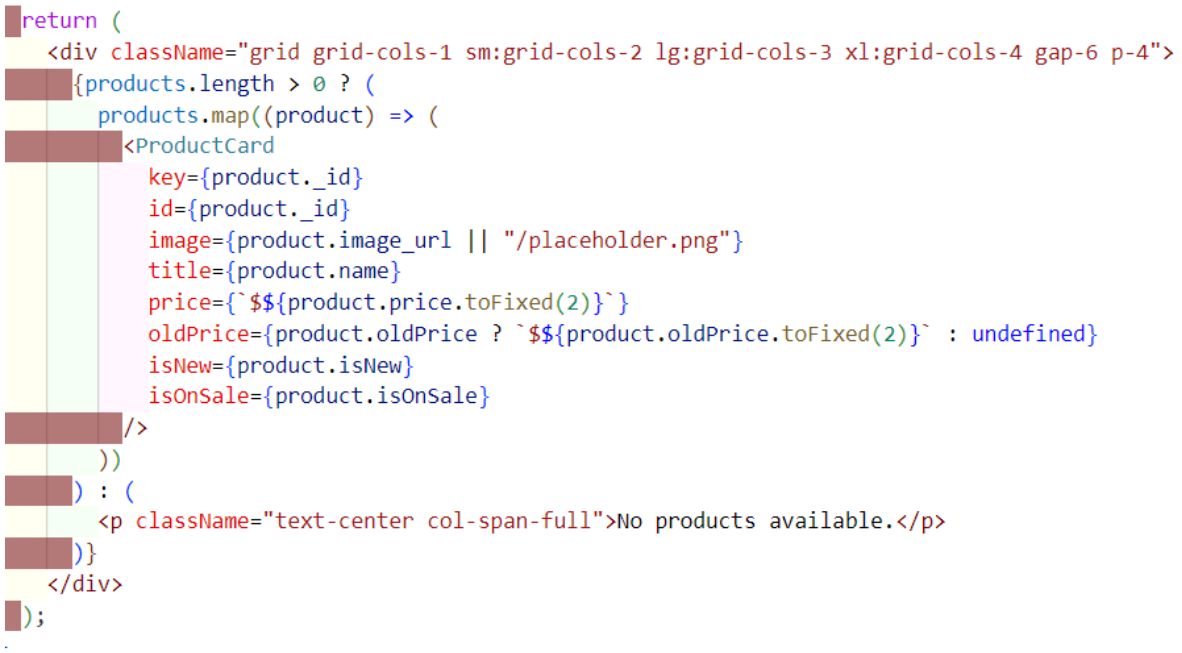
Integrated Data into Next.js:

- Used Next.js to fetch data from Sanity CMS and display it on the frontend.



page.tsx U
next.config.ts M
products.ts U
src > app > shop > page.tsx > Product > isNew
Codeium: Refactor | Explain
Tabnine | Edit | Test | Explain | Document | Codeium: Refactor | Explain | Generate JSDoc

```
import { client } from "@/sanity/lib/client";
import ProductCard from "../../../components/productcard";

interface Product {
  _id: string;
  name: string;
  price: number;
  oldPrice?: number;
  isNew?: boolean;
  isOnSale?: boolean;
  image_url: string;
}

// Fetch Products from Sanity
export const getProducts = async (): Promise<Product[]> => {
  const query = `*[_type == "products"]{
    _id,
    name,
    price,
    oldPrice,
    isNew,
    isOnSale,
    "image_url": image.asset->url
  }`;

  return await client.fetch<Product[]>(query);
};

export default async function Shop() {
  const products: Product[] = await getProducts();

  return (
    <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6 p-4">
```



```
return (
  <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6 p-4">
    {products.length > 0 ? (
      products.map((product) => (
        <ProductCard
          key={product._id}
          id={product._id}
          image={product.image_url || "/placeholder.png"}
          title={product.name}
          price={`$${product.price.toFixed(2)}`}
          oldPrice={product.oldPrice ? `$${product.oldPrice.toFixed(2)}` : undefined}
          isNew={product.isNew}
          isOnSale={product.isOnSale}
        />
      ))
    ) : (
      <p className="text-center col-span-full">No products available.</p>
    )}
  </div>
);
```

Part V: Testing and Debugging

1. **Tested API Calls**:
   - Used **Postman** to test all API requests and ensure the data is being fetched correctly.
2. **Ensured Data Integrity**:
   - Verified that the data is correctly populated in Sanity and reflected on the website frontend.

Conclusion:

The integration of third-party API data into Sanity CMS for the "Comforty" e-commerce platform was successfully completed. This process involved setting up the environment, defining the schema, migrating data, and fetching dynamic content. The website is now able to display product information fetched from Sanity CMS, allowing for greater control over the content, including the ability to update or delete products as needed. The next step is to enhance the frontend with more features and ensure the platform is fully dynamic.