API INTERGRATION REPORT

Comforty

Alishbah M. Kamran

API INTEGRATION

This documentation outlines the comprehensive workflow of integrating an external API into our system, adapting it for seamless compatibility, and fetching data to display on the frontend.

In This Process

Api Integration:

We integrated an external API into our application to leverage its robust features and data. The API provided a structured dataset that required transformation to align with our application's requirements.

Sanity Schema Adjustments:

The external API's structure was migrated into our Sanity CMS, where schema adjustments were made to ensure proper mapping of data. These adjustments involved creating new fields, modifying existing ones, and defining relationships to store and manage data efficiently.

Fetching Data to Frontend:

Once the data was successfully imported into Sanity, it was fetched on the frontend using optimized queries. This ensured that the displayed data was dynamic, accurate, and aligned with the user's experience.

API MIGRATION

MIGRATION SCRIPT

Following migrate categories and products data from a REST API to Sanity CMS while preserving data relationships, uploading images to Sanity, and ensuring schema adjustments.

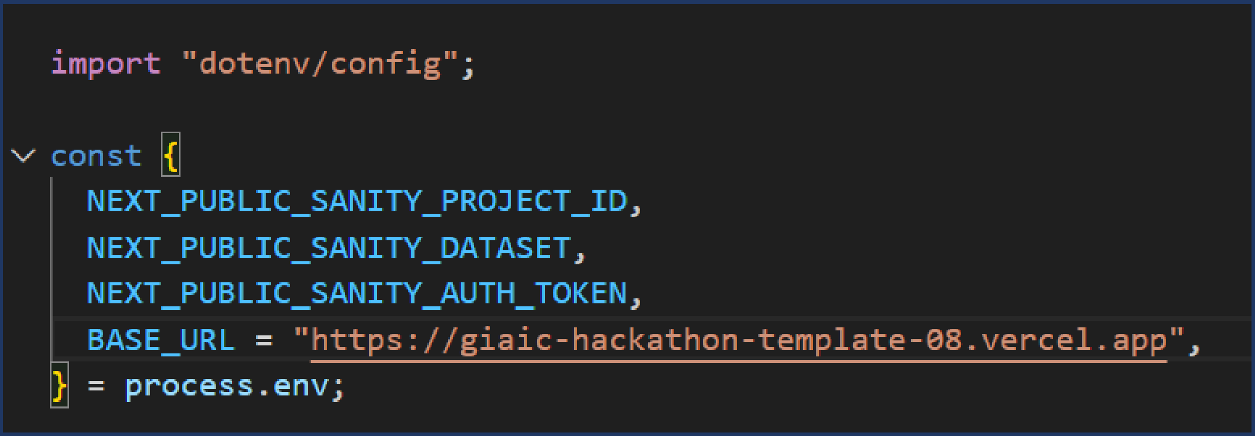
Setup and Configuration:

Environment Variables:

1) *The script loads essential variables from a .env file using dotenv/config.*

2) *Required variables:*

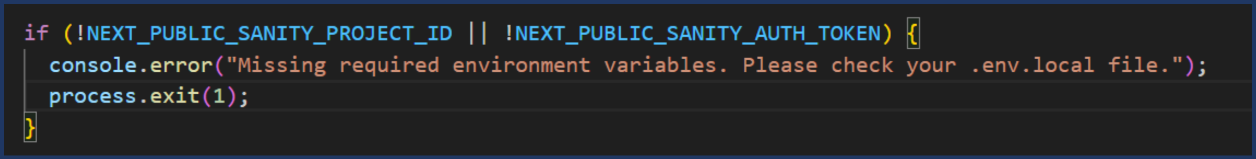
- NEXT_PUBLIC_SANITY_PROJECT_ID (Sanity project ID)
- NEXT_PUBLIC_SANITY_DATASET (Sanity dataset, e.g., "production")
- NEXT_PUBLIC_SANITY_AUTH_TOKEN (Sanity API token)
- BASE_URL (REST API base URL)



```
import "dotenv/config";

const {
  NEXT_PUBLIC_SANITY_PROJECT_ID,
  NEXT_PUBLIC_SANITY_DATASET,
  NEXT_PUBLIC_SANITY_AUTH_TOKEN,
  BASE_URL = "https://giaic-hackathon-template-08.vercel.app",
} = process.env;
```

Validation:

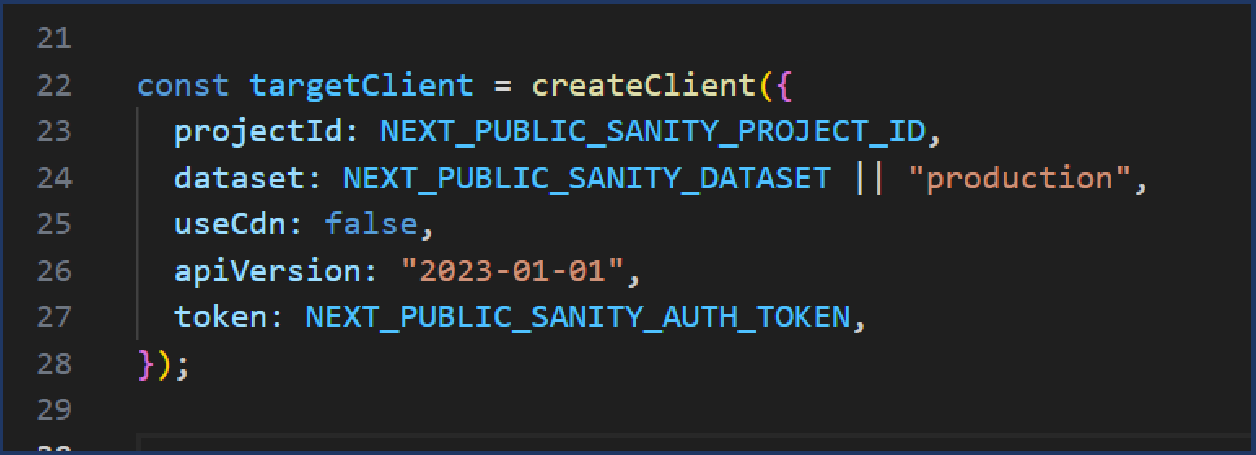
Ensures all required variables are set, terminating the script if not.



```
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);
}
```

Sanity Client Configuration:

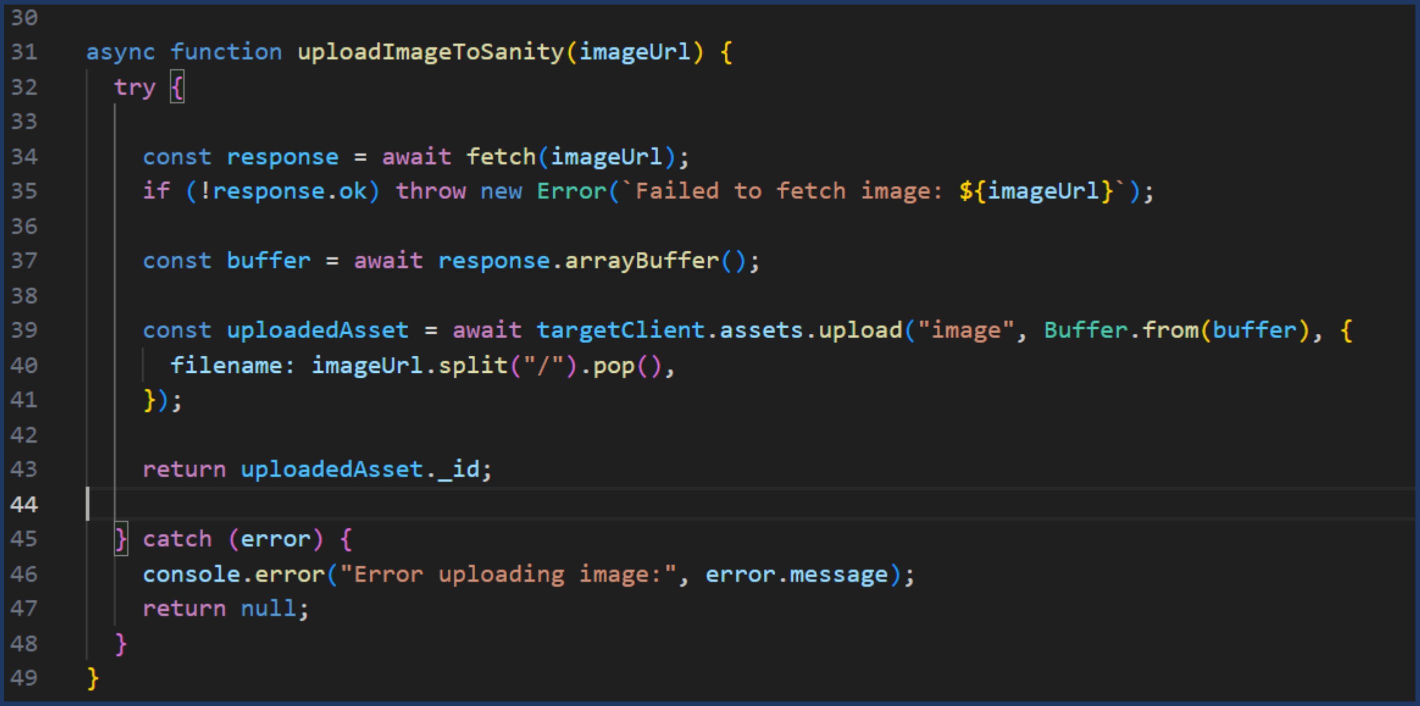
A Sanity client is created using @sanity/client to interact with the target dataset.



```
const targetClient = createClient({
  projectId: NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: NEXT_PUBLIC_SANITY_DATASET || "production",
  useCdn: false,
  apiVersion: "2023-01-01",
  token: NEXT_PUBLIC_SANITY_AUTH_TOKEN,
});
```

Image Upload to Sanity:

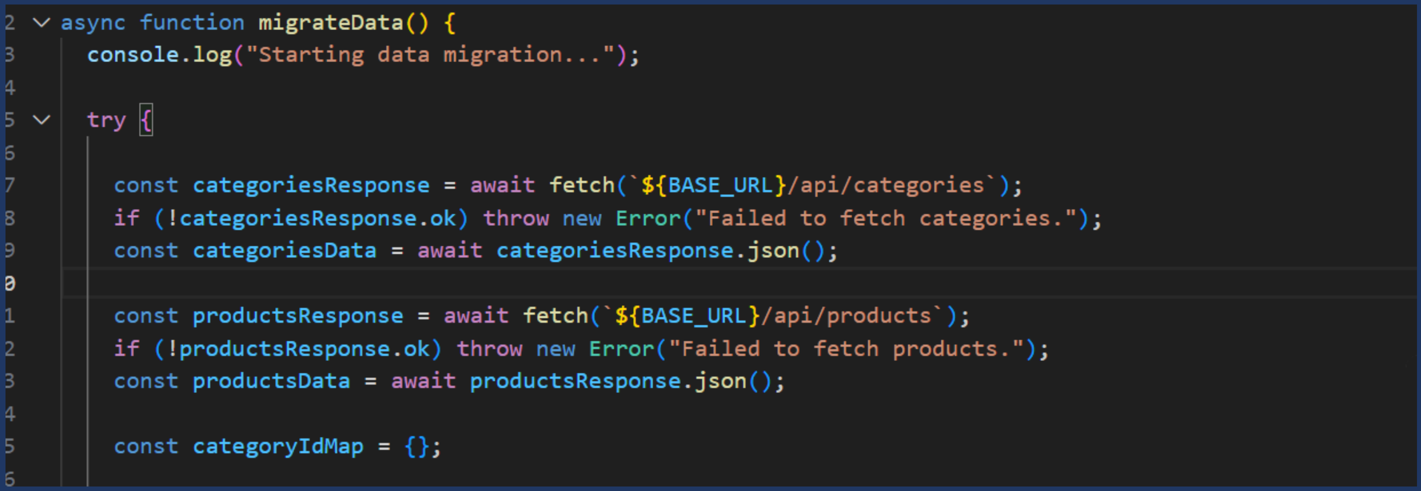
- Downloads an image from a URL and uploads it to Sanity's asset library.
- Returns the asset ID upon successful upload.



```javascript

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;
  }
}
```

Fetch Data from REST API:

Categories and products are fetched from the REST API using the provided BASE_URL.

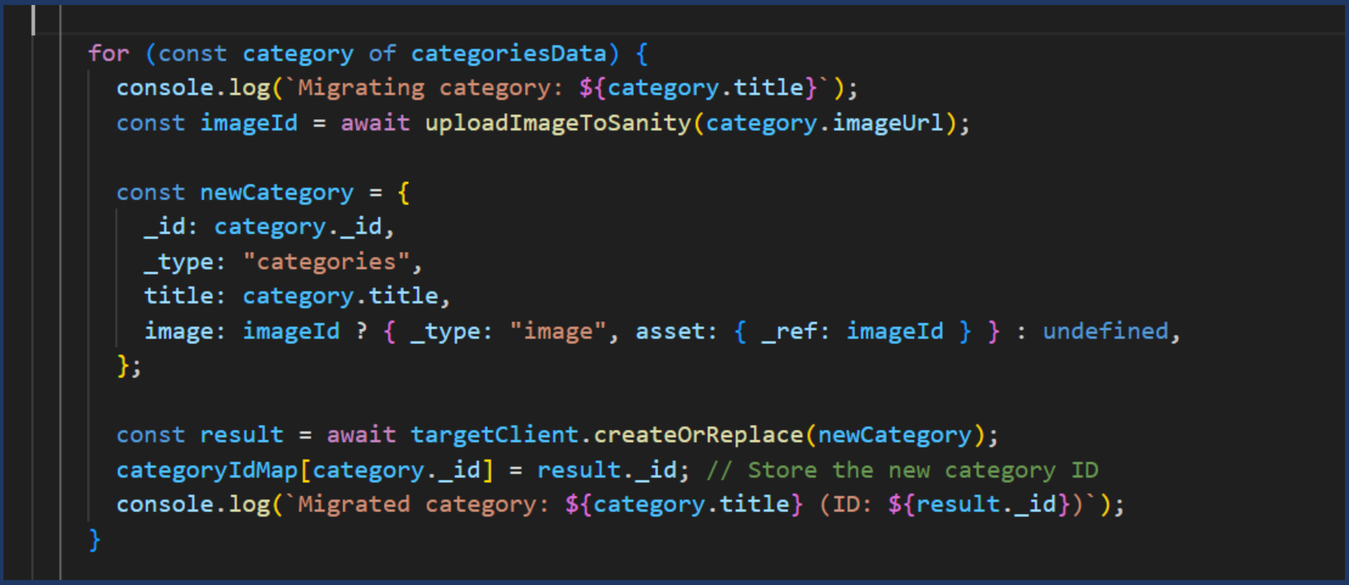


```javascript
async function migrateData() {
  console.log("Starting data migration...");

  try {

    const categoriesResponse = await fetch(`${BASE_URL}/api/categories`);
    if (!categoriesResponse.ok) throw new Error("Failed to fetch categories.");
    const categoriesData = await categoriesResponse.json();

    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 = {};

```

Migrate Categories:

Each category is processed to:

- Upload its image to Sanity.
- Create or replace the category in Sanity with adjusted schema.



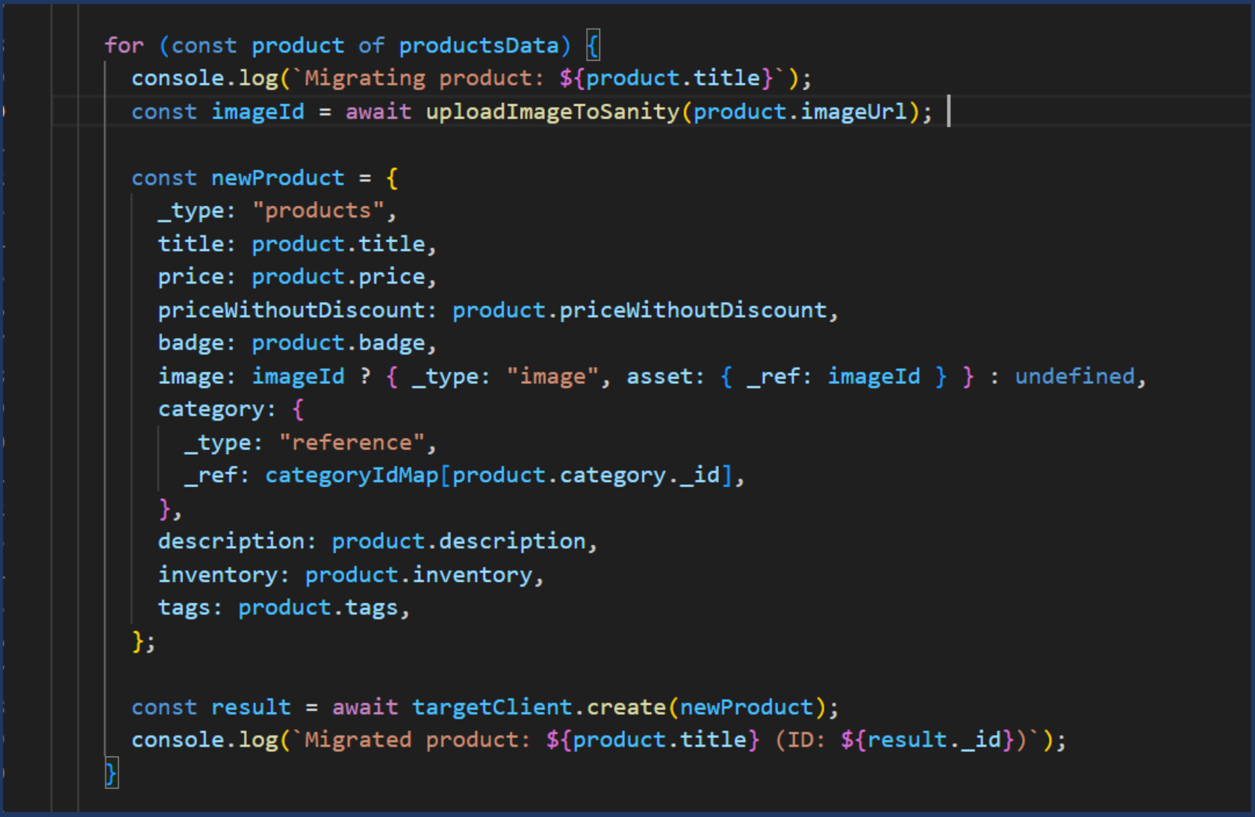
```
for (const category of categoriesData) {
  console.log(`Migrating category: ${category.title}`);
  const imageId = await uploadImageToSanity(category.imageUrl);

  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; // Store the new category ID
  console.log(`Migrated category: ${category.title} (ID: ${result._id})`);
}
```

Products Categories:

Each product is processed to:

- Upload its image to Sanity.

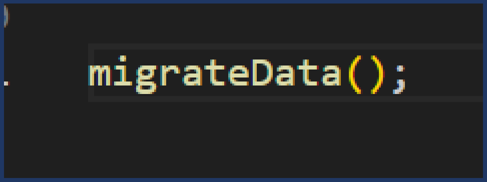
- Create a product in Sanity with a reference to its associated category.



```
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})`);
}
```

Execution:

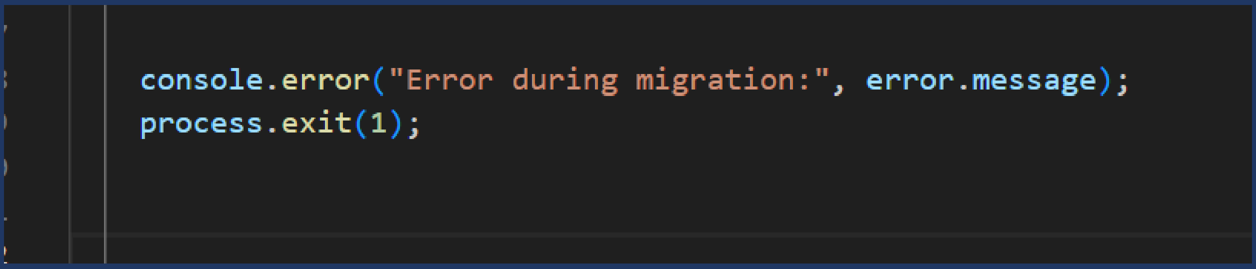
he migration process is initiated by calling the migrateData function.



```
migrateData();
```

Error Handling:

- Catches and logs errors during the migration process.
- Stops execution if a critical failure occurs.



```
console.error("Error during migration:", error.message);
process.exit(1);
```

Conclusion:

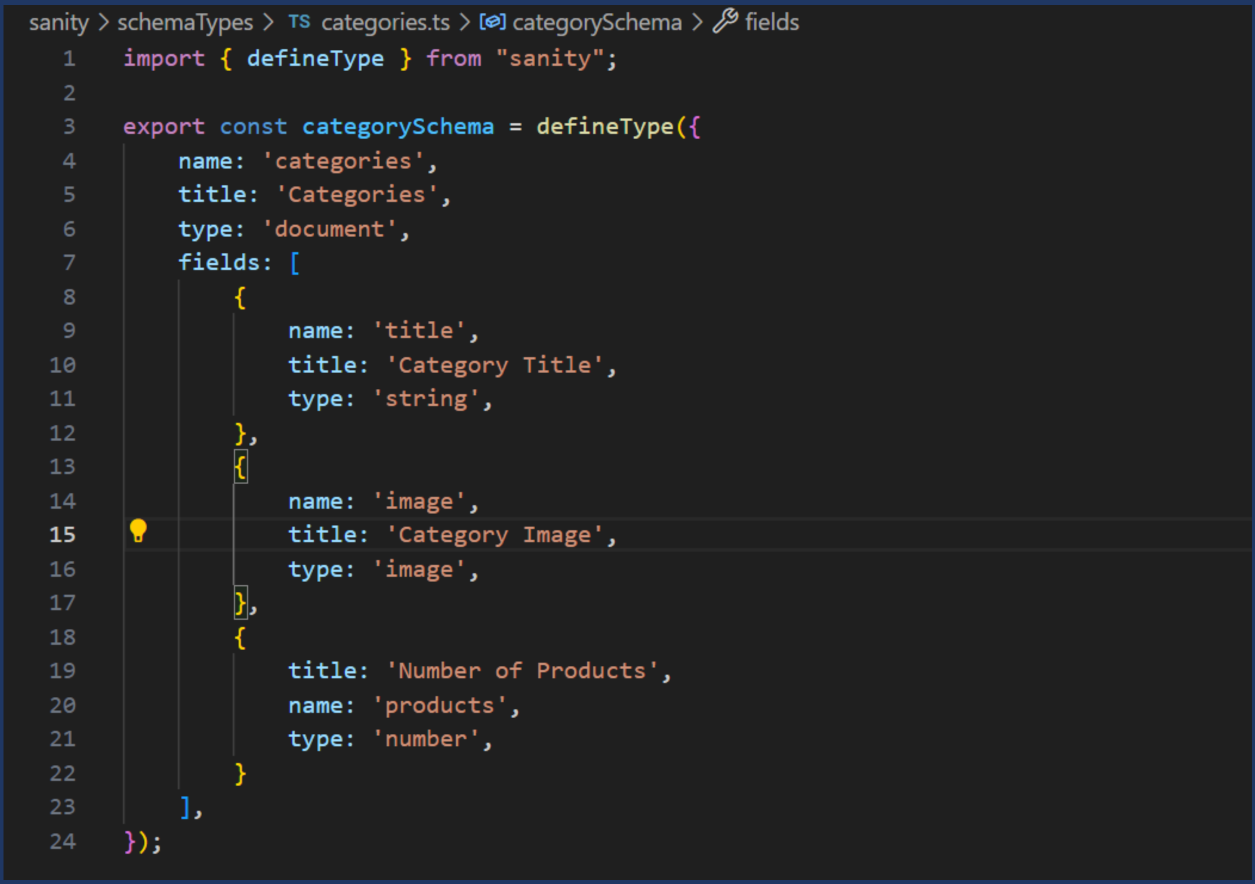
This script automates the data migration process by:

- Fetching categories and products from a REST API.
- Uploading images to Sanity.
- Saving data in Sanity with schema adjustments.

SANITY SCHEMA & DATA FETCHING ON FRONTEND

Categories Schema

The categories schema is designed to manage and store category information. Each category includes a title, an image, and the number of products it contains.

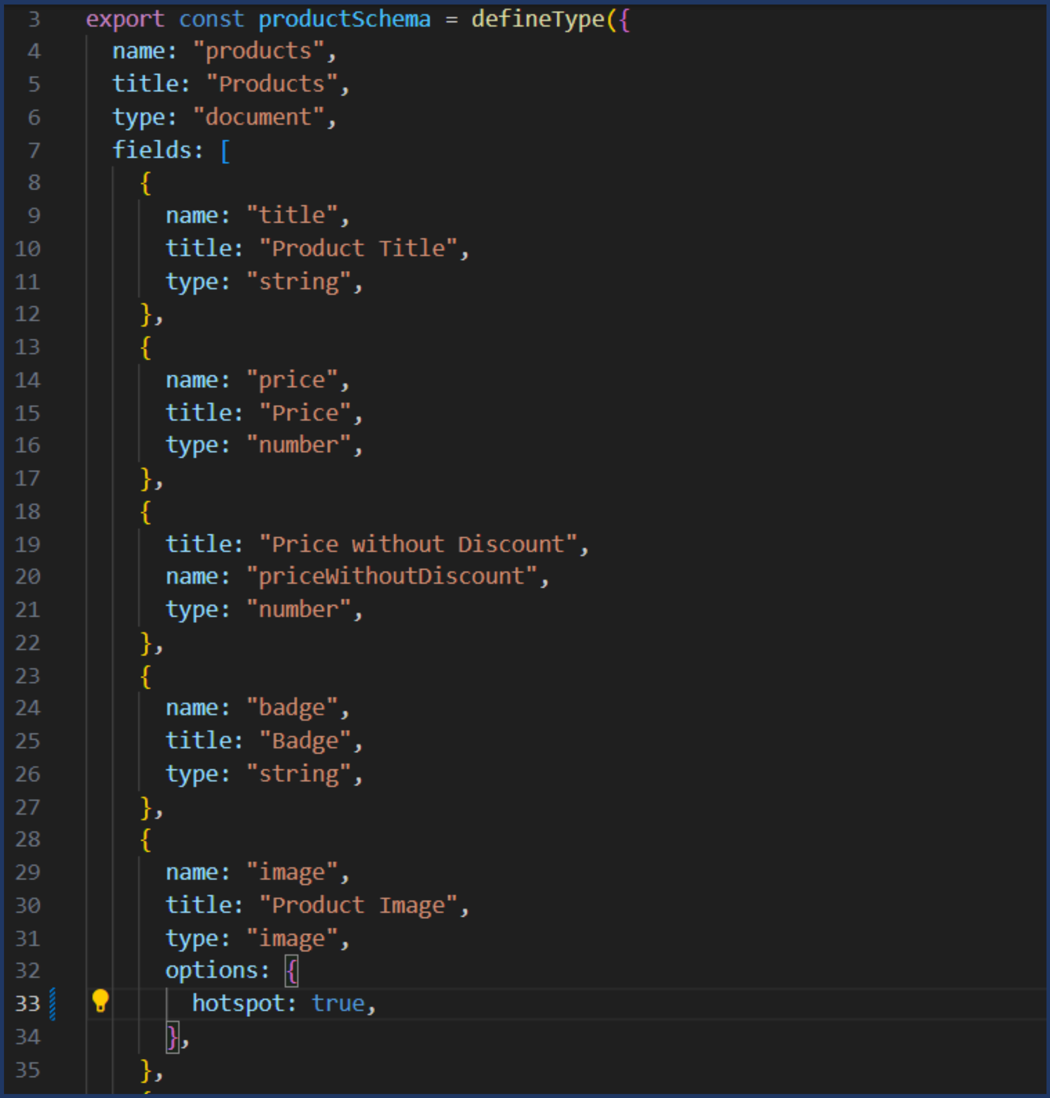


sanity > schemaTypes > categories.ts > categorySchema > fields

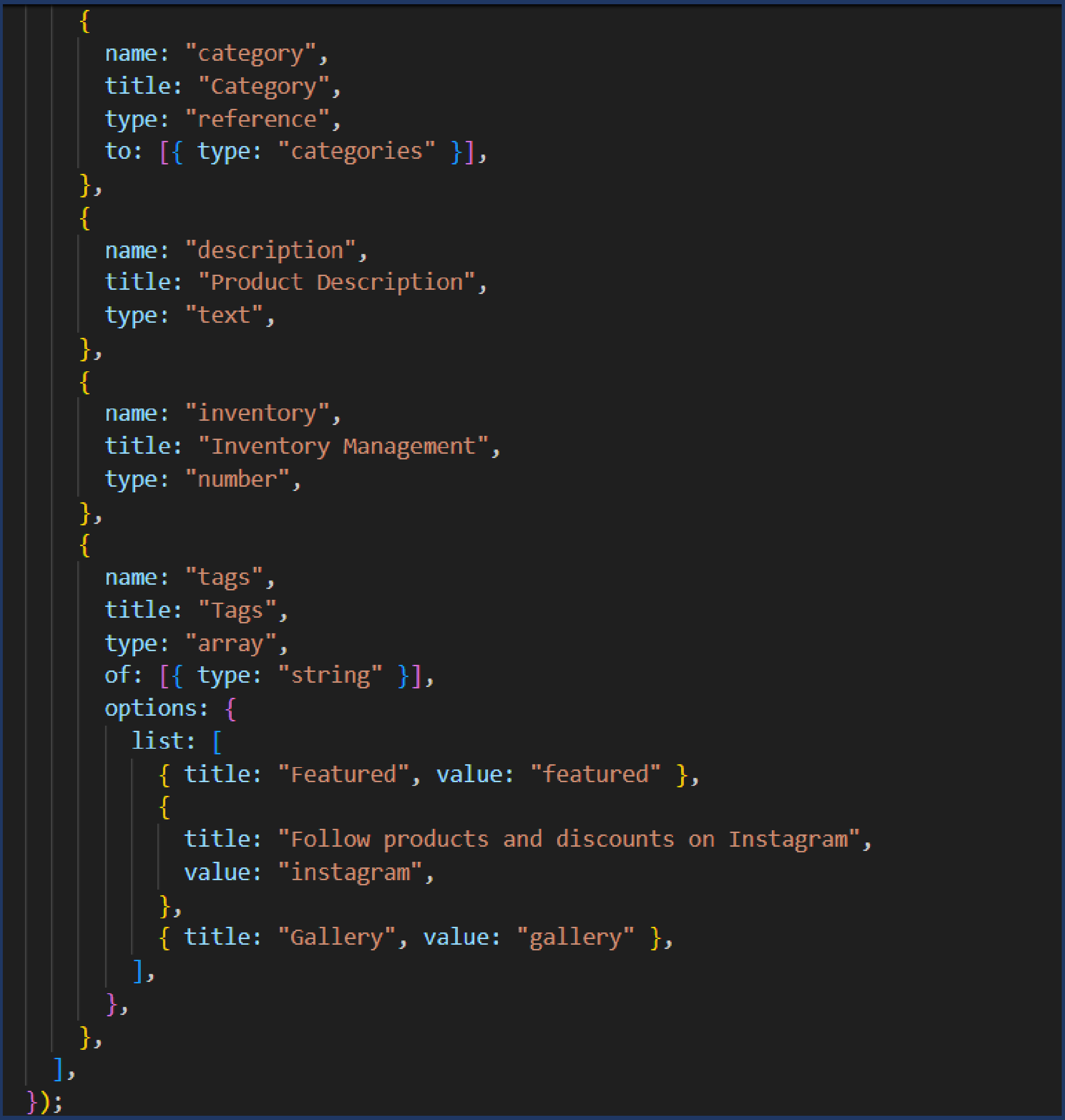
```ts
import { defineType } from "sanity";

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',
        }
    ],
});
```

Products Schema

The products schema is designed to manage product information. Each product includes details such as its title, price, category, description, and tags, among others.



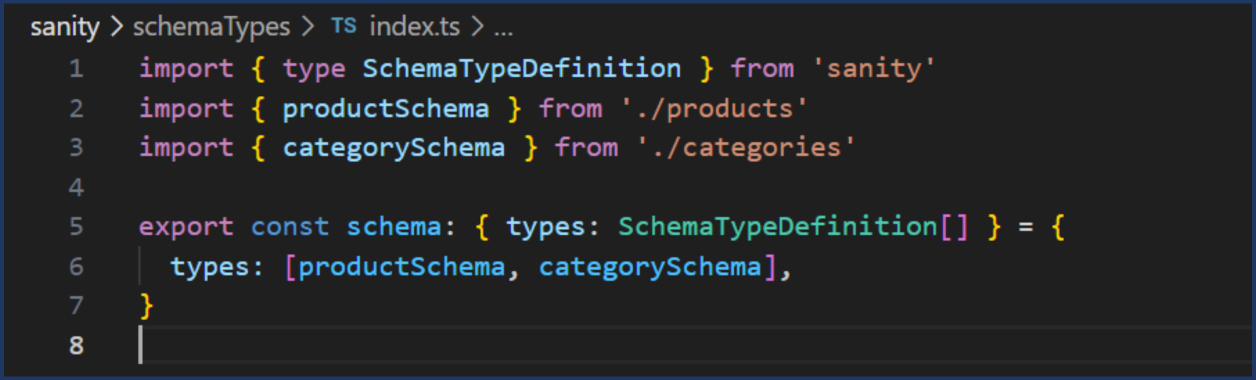
```
export const productSchema = defineType({
  name: "products",
  title: "Products",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Product Title",
      type: "string",
    },
    {
      name: "price",
      title: "Price",
      type: "number",
    },
    {
      title: "Price without Discount",
      name: "priceWithoutDiscount",
      type: "number",
    },
    {
      name: "badge",
      title: "Badge",
      type: "string",
    },
    {
      name: "image",
      title: "Product Image",
      type: "image",
      options: {
        hotspot: true,
      },
    },
```



```
      {
        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" },
          ],
        },
      },
    ],
});
```

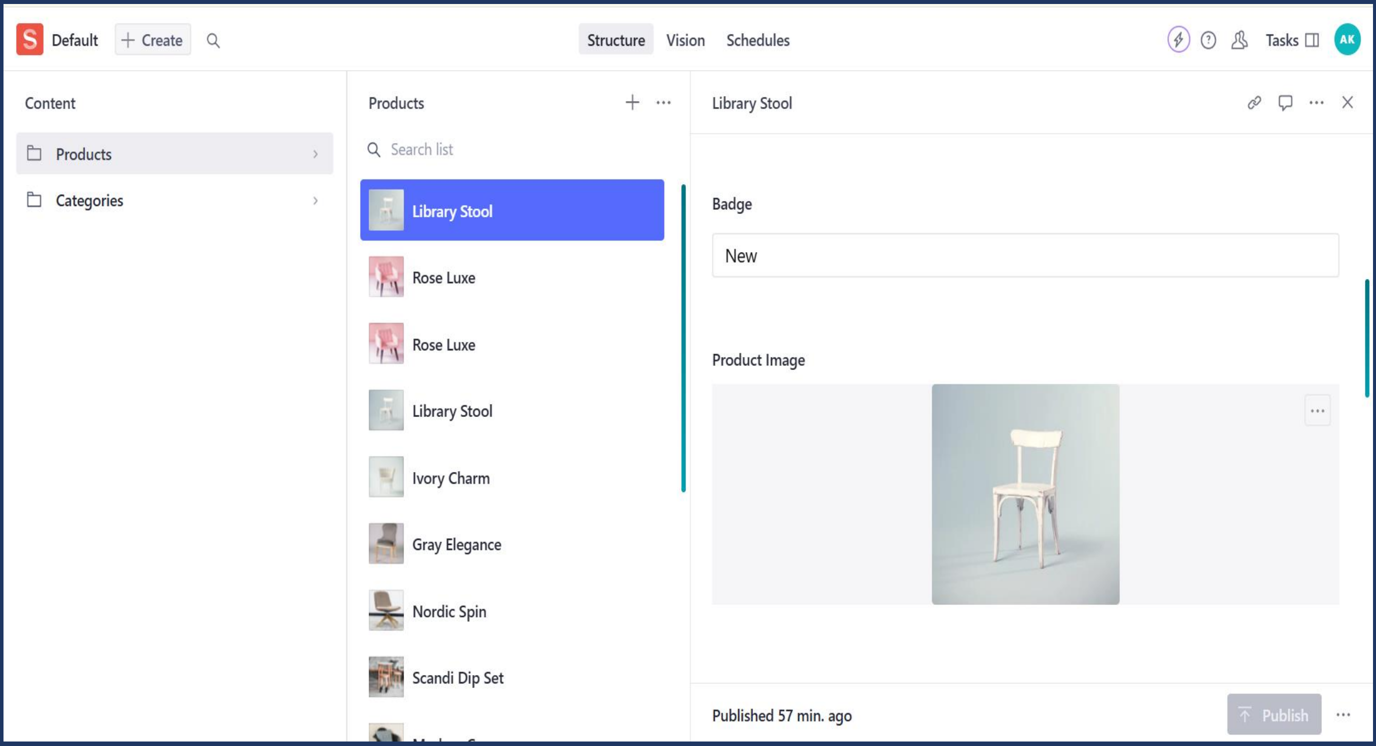
Schema Registration

Both schemas (categories and products) are registered in the index.ts file under schemaTypes folder to be used in Sanity project.


sanity > schemaTypes > TS index.ts > ...

```ts
import { type SchemaTypeDefinition } from 'sanity'
import { productSchema } from './products'
import { categorySchema } from './categories'

export const schema: { types: SchemaTypeDefinition[] } = {
  types: [productSchema, categorySchema],
}

```

Populated Fields



Badge

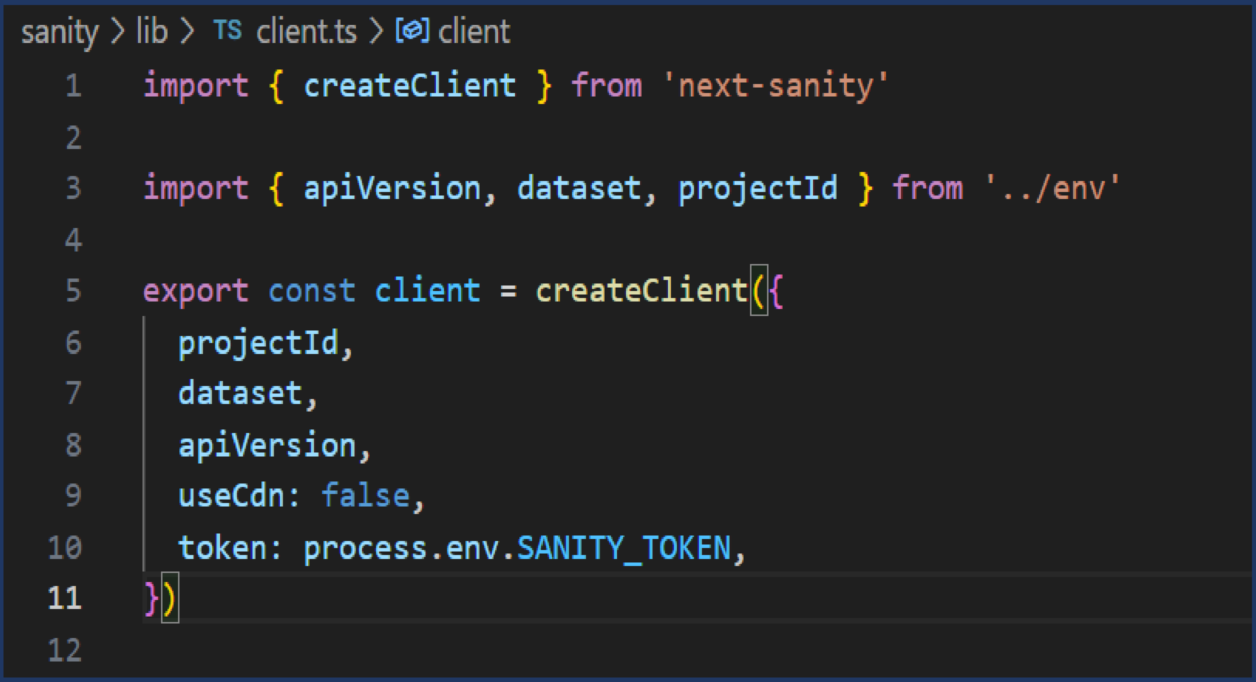
New

Product Image

Published 57 min. ago

Displaying On Frontend

Setting up sanity client:



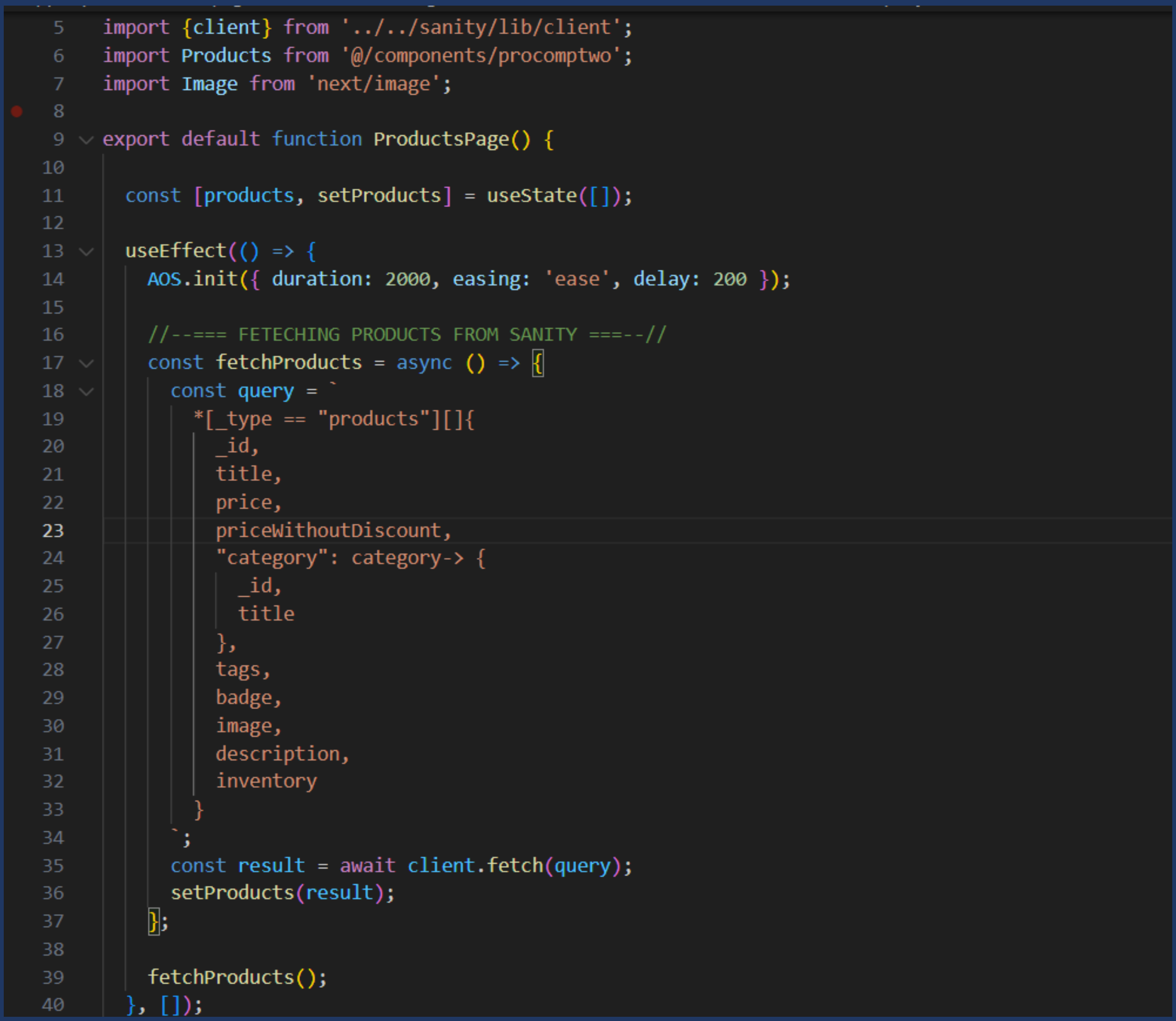
sanity > lib > TS client.ts > client

```ts
import { createClient } from 'next-sanity'

import { apiVersion, dataset, projectId } from '../env'

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  useCdn: false,
  token: process.env.SANITY_TOKEN,
})

```

Fetching Data From Sanity:

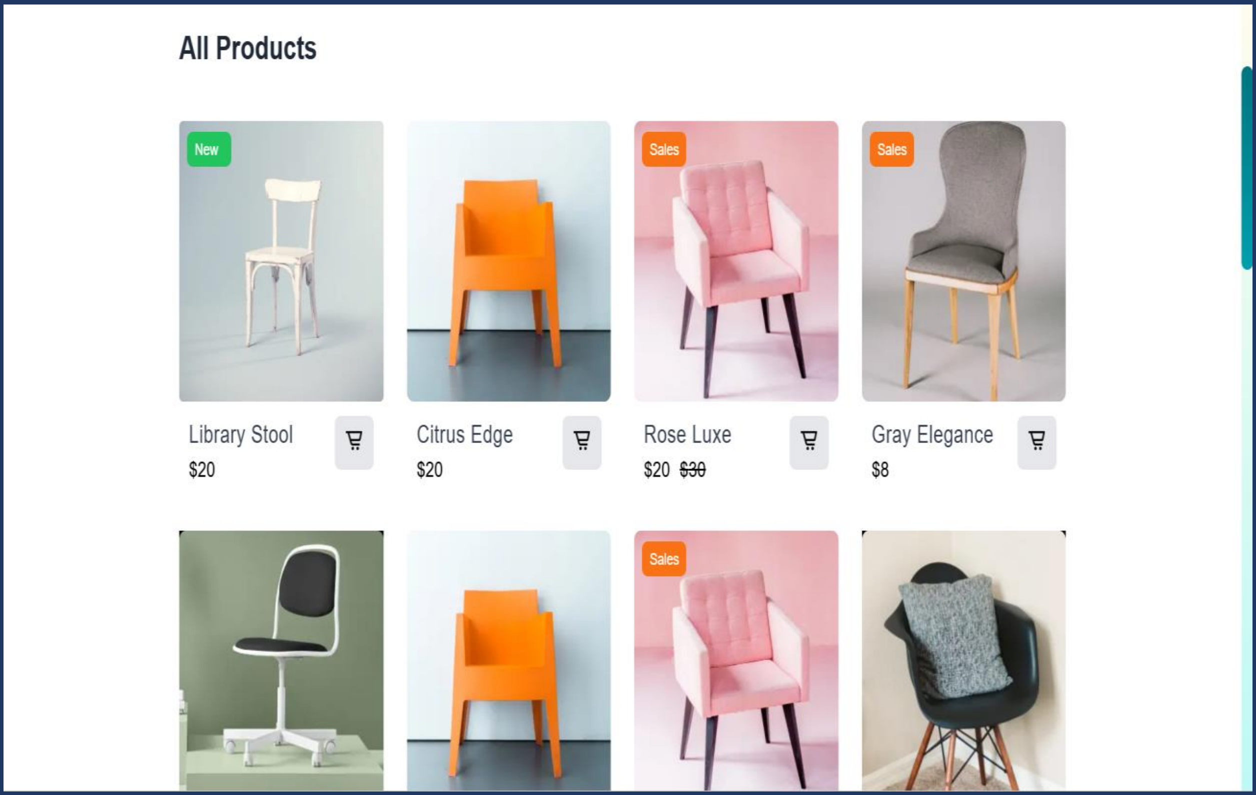
The data fetching logic is implemented in the Products' Page.

- *Sanity Query:* The GROQ query fetches data from the products dataset.



```
import {client} from '../../sanity/lib/client';
import Products from '@/components/procomptwo';
import Image from 'next/image';

export default function ProductsPage() {

  const [products, setProducts] = useState([]);

  useEffect(() => {
    AOS.init({ duration: 2000, easing: 'ease', delay: 200 });

    //--=== FETECHING PRODUCTS FROM SANITY ===--//
    const fetchProducts = async () => {
      const query = `
        *[_type == "products"][]{
          _id,
          title,
          price,
          priceWithoutDiscount,
          "category": category-> {
            _id,
            title
          },
          tags,
          badge,
          image,
          description,
          inventory
        }
      `;
      const result = await client.fetch(query);
      setProducts(result);
    };

    fetchProducts();
  }, []);
```

Displaying On Frontend:


All Products

New

Library Stool
$20

Citrus Edge
$20

Sales

Rose Luxe
$20 ~~$30~~

Sales

Gray Elegance
$8

Sales