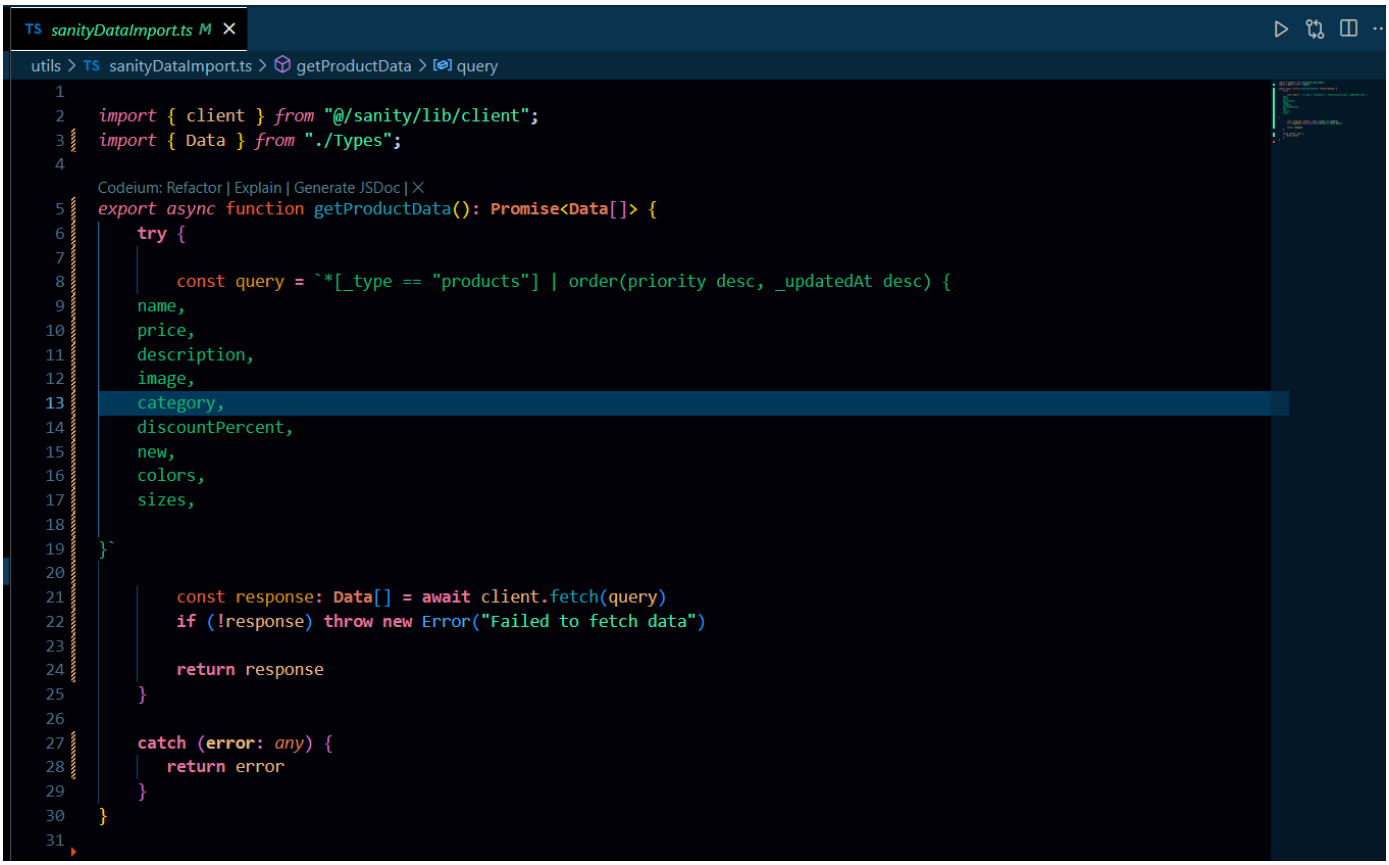
DAY-3 API INTEGRATOIN PROCESS

STEPS INVOLVED:

- Created a Token in the sanity project at sanity.io, with developer access. Copied the script provided to import the data from the api. Created a file with name “importData.mjs”, pasted the code in it and run it with command “node importData.mjs” which successfully imported product data from the api to sanity’s content lake, in the schemas structure I’ve made. (speaking of running the “copied” code, I also got errors which I resolved).
- Haven’t done any adjustment in schema as of now, but will be adding these, “reviews, Stock, Tags, Returnable, return window, on arrival check guidelines” fields right after submitting the day-3,
- Have used the built in fetch method to fetch data from api and used the sanity’s client.fetch to try fetching from sanity i.e. validating that connection with sanity is working, used sanity’s client.create method to create product doc on each iteration.

Here are the Screen Shots of:

1. **Api Call:**

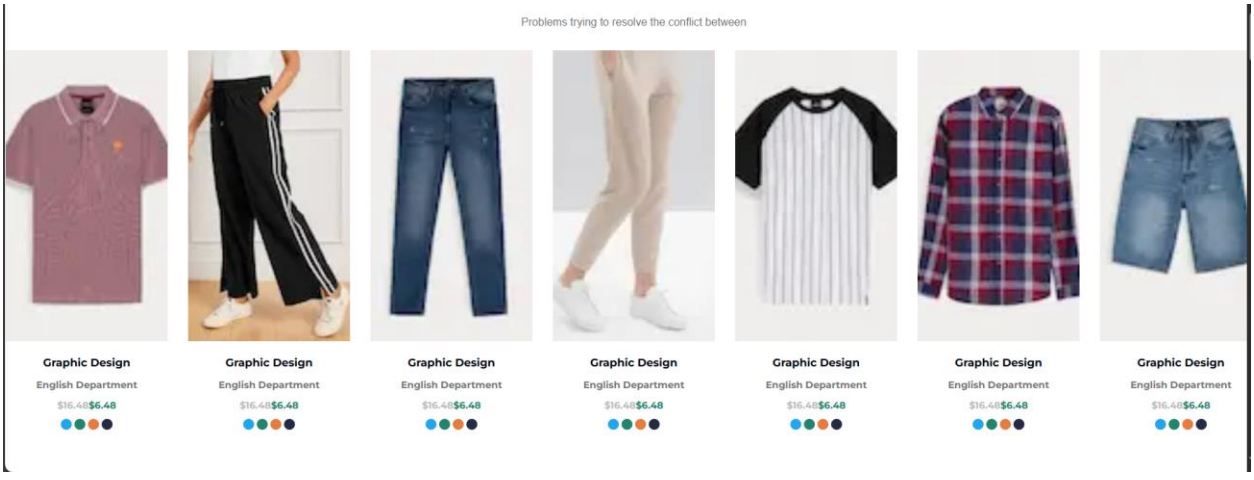


BestSellerProducts.tsx M

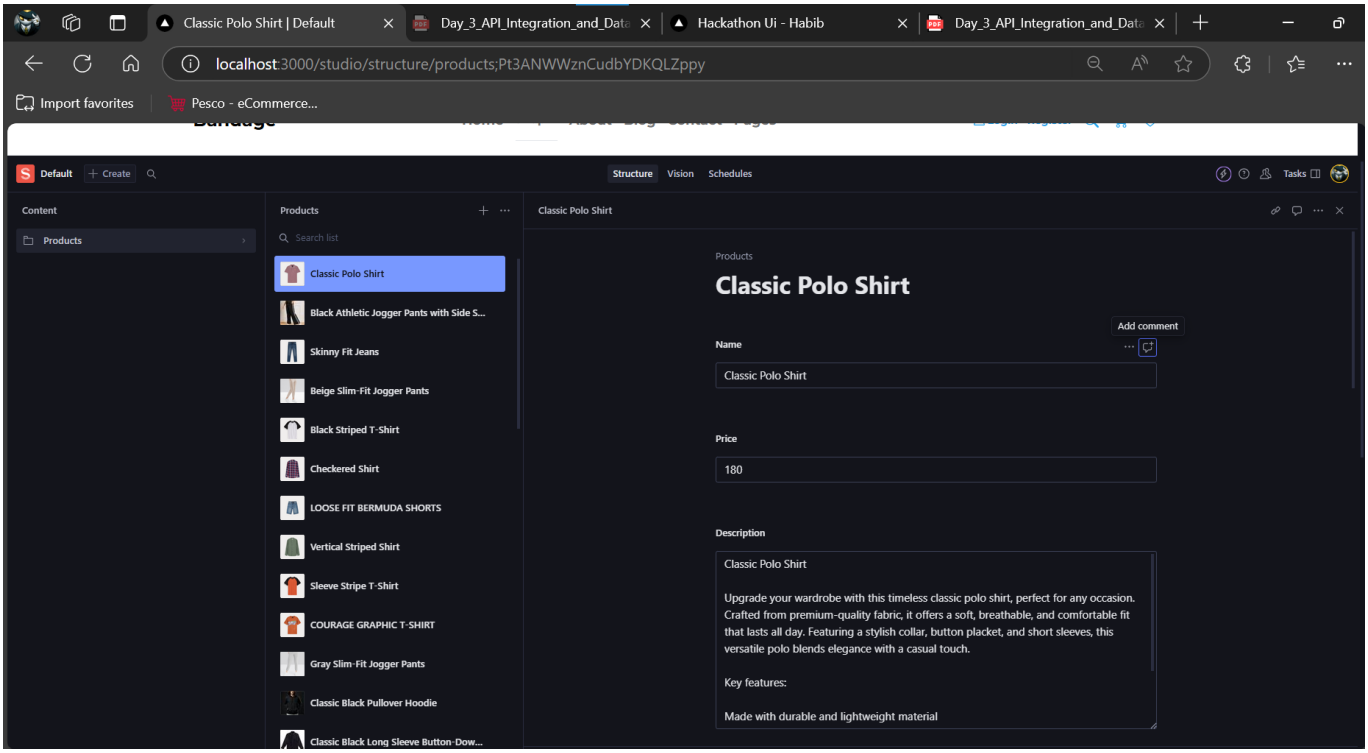
src > components > BestSellerProducts.tsx > BestSellerProducts > data.map() callback

```tsx
import Link from "next/link";
import ProductCard from "./ProductCard";
import { getProductData } from "../../utils/sanityDataImport";
import { Data } from "../../utils/Types";
import { urlFor } from "@/sanity/lib/image";

Codeium: Refactor | Explain | Generate JSDoc |
export default async function BestSellerProducts() {


    const data: Data[] = await getProductData()

    return (
        <div className="w-screen mb-20">
            <div className="w-full flex flex-col justify-center items-center">
                <p className="text-lg font-montserrat text-[#737373] font-medium mb-3"
                >Featured Products</p>

                <h1 className="text-2xl font-montserrat font-bold text-black
                mb-2">BESTSELLER PRODUCTS</h1>

                <p className="text-base text-[#737373]">Problems trying to resolve the
                conflict between  </p>
            </div>


            <div className="w-full flex justify-center items-center flex-wrap gap-[30px]
            ">
                {
                    data.map((item: Data) => (<ProductCard key={item.name} image={item?.
```

2. Data fetched and displayed on Frontend:



3. Schema populated:

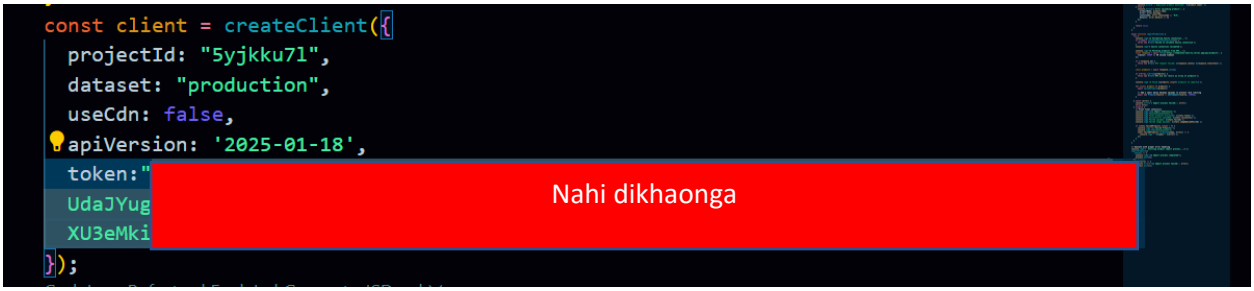


Code Snippets of Data Migration process

1. Importing data from the Api and handling the errors:

```
Codeium: Refactor | Ex
async function importProducts() {
  try {
    console.log('🔄 Validating Sanity connection...');
    if (!(await validateSanityConnection())) {
      throw new Error('Failed to validate Sanity connection');
    }
    console.log('✅ Sanity connection validated');

    console.log('🔄 Fetching products from API...');
    const response = await fetch('https://template1-neon-nu.vercel.app/api/products', {
      timeout: 30000 // 30 second timeout
    });
```

2. **Created sanity client with the sanity's provided "create client" method.**



3. **Function to handle product upload, it initially passed the products' image from the product object assigned in the arguments to "uploadImageToSanity" function which will handle image upload to sanity, and then it creates the required document for the product, which is an object based on the same structure of our Sanity's product Schema, the thing which make it identifiable as a sanity schema document is its "_type".**

   Then is document is created in the products' schema through the sanity's "client.create" method by providing it the document object, which is populated with the product data coming through api.

```
async function uploadProduct(product) {
  try {
    stats.total++;
    console.log(`\n📦 Processing product ${stats.total}/${stats.total}: "${product.name}"`);

    function validateProductData(product: any): boolean
    validateProductData(product);

    const imageId = await uploadImageToSanity(product.imageUrl, product.name);
    if (!imageId) {
      throw new Error('Image upload failed');
    }

    const document = {
      _type: 'products',
      name: product.name,
      description: product.description,
      price: product.price,
      image: {
        _type: 'image',
        asset: {
          _ref: imageId,
        },
      },
      category: product.category,
      discountPercent: product.discountPercent || 0,
      new: Boolean(product.isNew),
      colors: Array.isArray(product.colors) ? product.colors : [],
      sizes: Array.isArray(product.sizes) ? product.sizes : [],
      _createdAt: new Date().toISOString()
    };

    const createdProduct = await client.create(document);
    console.log(`✅ Product "${product.name}" uploaded successfully`);
    stats.successful++;
```

4. **Loop which calls the product upload function for every product.**

```
for (const product of products) {
  await uploadProduct(product);

  // Add a small delay between uploads to prevent rate limiting
  await new Promise(resolve => setTimeout(resolve, 1000));
}
```

5. **Created an async function “getproductData” to import data from sanity, the function does is, creating a query variable which contains the “groq” query to fetch the required product data from sanity. And finally the function return the data.**

```
import { client } from "@/sanity/lib/client";
import { Data } from "./Types";

Codeium: Refactor | Explain | Generate JSDoc | X
export async function getProductData(): Promise<Data[]> {
    try {

        const query = `*[_type == "products"] | order(priority desc, _updatedAt desc) {
    name,
    price,
    description,
    image,
    category,
    discountPercent,
    new,
    colors,
    sizes,

}`

        const response: Data[] = await client.fetch(query)
        if (!response) throw new Error("Failed to fetch data")

        return response
    }

    catch (error: any) {
       return error
    }
}
```

6. **Finally called the “getProductData” in the required frontend component and awaited the result and stored the product in the “data” variable, at last run map function on the data to convert each product stored in the data variable into a product card.**

```
const data: Data[] = await getProductData()

return (
    <div className="w-screen mb-20">
        <div className="w-full flex flex-col justify-center items-center">
            <p className="text-lg font-montserrat text-[#737373] font-medium mb-3"
            >Featured Products</p>

            <h1 className="text-2xl font-montserrat font-bold text-black
            mb-2">BESTSELLER PRODUCTS</h1>

            <p className="text-base text-[#737373]">Problems trying to resolve the
            conflict between  </p>
        </div>


        <div className="w-full flex justify-center items-center flex-wrap gap-[30px]
        ">
            {
                data.map((item: Data) => (<ProductCard key={item.name} image={item?.
                image ? urlFor(item?.image).url() : ""} />))
            }
        </div>
```