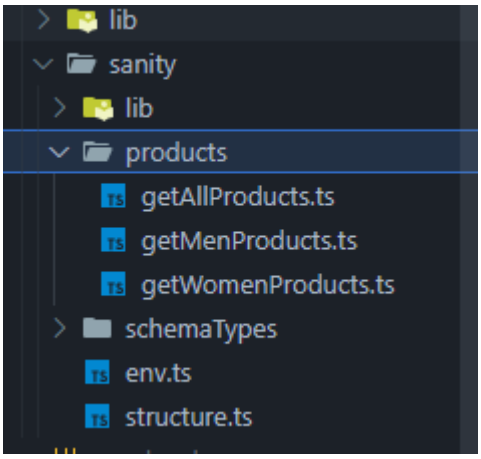
Day 3 - API Integration Report - VaultSneaks

Objective

For integrating existing APIs and migrating data to Sanity CMS to ensure the favorable running of the new dynamic shoe marketplace. This consists of using, collaborating with Typegen for obtaining data, making a helper function in `sanity/products`, and integrating products on the frontend.



Key Activities

1. **API Integration**
    - Integrated API to fetch product data
    - Utilized Typegen to streamline the process of fetching data from Sanity CMS.

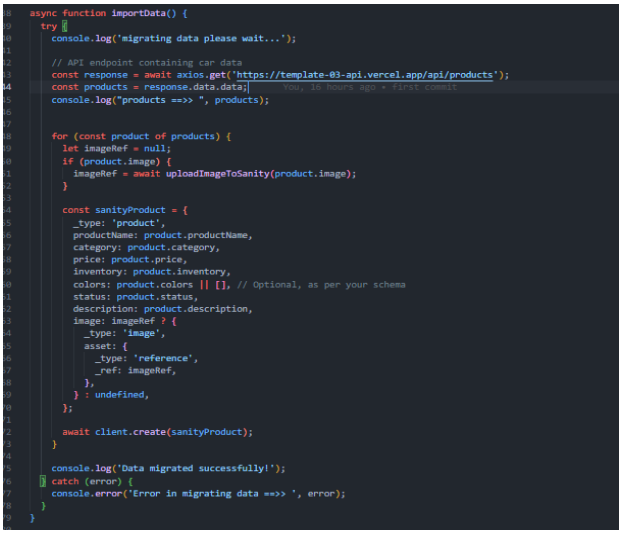
2. **Data Migration**
    - Migrated product data into Sanity CMS using a migration script.
    - Validated data integrity and alignment with the defined schema.

```
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

// Load environment variables from .env.local
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: path.resolve(__dirname, '../.env.local') });

// Create Sanity client
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: false,
  token: process.env.NEXT_PUBLIC_SANITY_TOKEN,
  apiVersion: '2025-01-17'
});


async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop()
    });
    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}
```



```
async function importData() {
  try {
    console.log('migrating data please wait...');

    // API endpoint containing car data
    const response = await axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    console.log("products ==>> ", products);


    for (const product of products) {
      let imageRef = null;
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image);
      }

      const sanityProduct = {
        _type: 'product',
        productName: product.productName,
        category: product.category,
        price: product.price,
        inventory: product.inventory,
        colors: product.colors || [], // Optional, as per your schema
        status: product.status,
        description: product.description,
        image: imageRef ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        } : undefined,
      };

      await client.create(sanityProduct);
    }

    console.log('Data migrated successfully!');
  } catch (error) {
    console.error('Error in migrating data ==>> ', error);
  }
}
```

3. Helper Function Creation

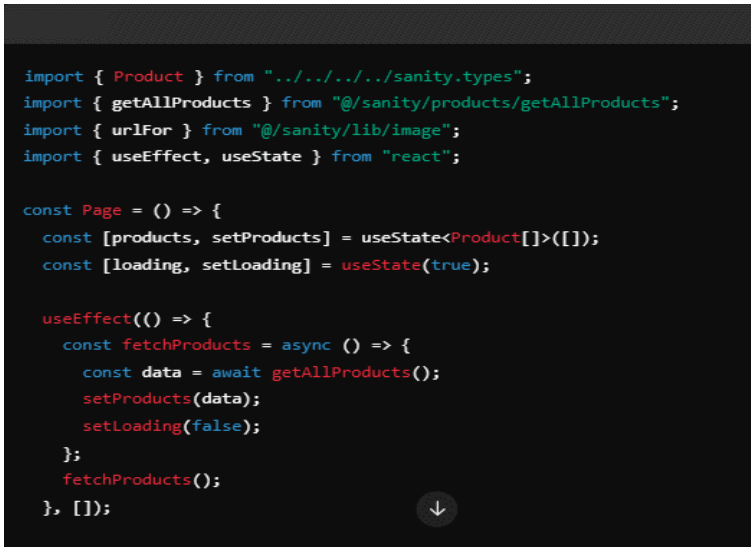
- Developed a helper function in `sanity/products` to efficiently fetch products from Sanity CMS.
- Ensured the function could be reused across different parts of the application to maintain consistency.

```
You, 9 hours ago | 1 author (You)
import { defineQuery } from "next-sanity";
import { client } from "../lib/client";

export const getAllProducts = async () => {
  const All_PRODUCTS_QUERY = defineQuery(
    `*[_type=="product"] | order(name asc)`
  );
  try{
        const products = await client.fetch(
    All_PRODUCTS_QUERY,
        )
        return products || [];
  }catch(error){
    console.error("Error fetching products", error);
    return [];
  }
};      You, 9 hours ago • Add typegen, display products in UI from sanity…
```

4. Frontend Implementation

- Displayed migrated products on the frontend using Next.js.
- Ensured that data fetched from Sanity CMS was correctly rendered in the user interface.



```
import { Product } from "../../../../sanity.types";
import { getAllProducts } from "@/sanity/products/getAllProducts";
import { urlFor } from "@/sanity/lib/image";
import { useEffect, useState } from "react";

const Page = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchProducts = async () => {
      const data = await getAllProducts();
      setProducts(data);
      setLoading(false);
    };
    fetchProducts();
  }, []);
```

```
  if (loading) {
    return <div>Loading...</div>;
  }

  return (
    <div className="container">
      {products.map((product) => (
        <div key={product._id}>
          <h3>{product.name}</h3>
          <p>Price: ${product.price}</p>
          <img src={urlFor(product.image).url()} alt={product.name} />
        </div>
      ))}
    </div>
  );
};

export default Page;
```

Steps Taken

1. Understanding the Provided API

- Reviewed API documentation to comprehend available endpoints and data structures.
- Key API Endpoint Used:
  - `/products` (GET): Fetched product listings for migration into Sanity CMS.

2. Data Migration Process

- Used the migration script to transfer data from the API to Sanity CMS.
- **Steps**:
  - Fetched data from the API.
  - Transformed data to match Sanity schema requirements.
  - Imported data into Sanity CMS.

3. Helper Function in `sanity/products`

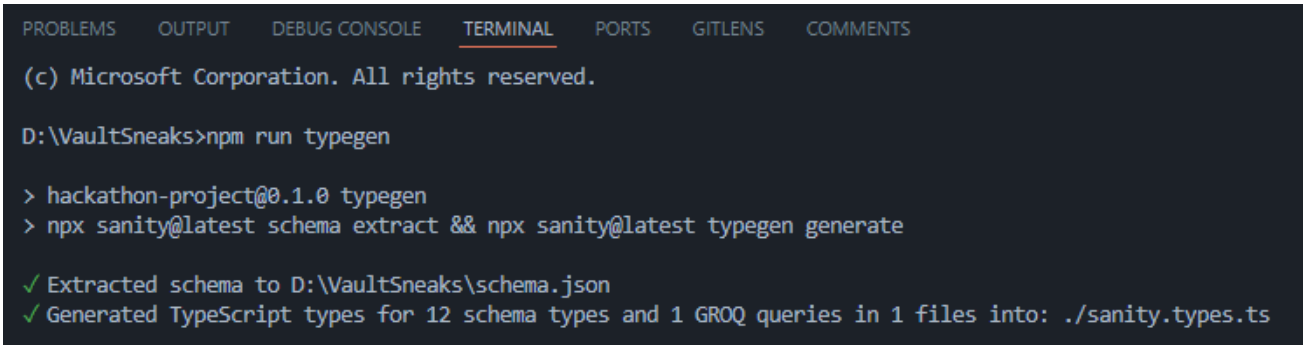
- Created a helper function for fetching product data from Sanity CMS, ensuring that all requests for product data follow a standardized approach.
- **Function**: `getAllProducts()`:
  - **Purpose**: To fetch and return product data from Sanity CMS.
  - **Usage**: Utilized across various components to ensure consistent data fetching and minimize redundant code.

4. API Integration in Next.js

- Created utility functions to fetch and handle API data.
- Rendered fetched data in Next.js components to display product listings on the frontend.
- Tested integration using tools like Postman, thunderclient and browser developer tools.

5. Understanding Typegen

- **Typegen**: A tool that automatically generates TypeScript types based on your GraphQL schema or API data structure. It ensures type safety by providing autocompletion and type-checking in your IDE.



- **Benefits**:
    - Reduces the risk of runtime errors by catching type-related issues during development
    - Enhances developer productivity through autocompletion and better documentation.

Best Practices Followed

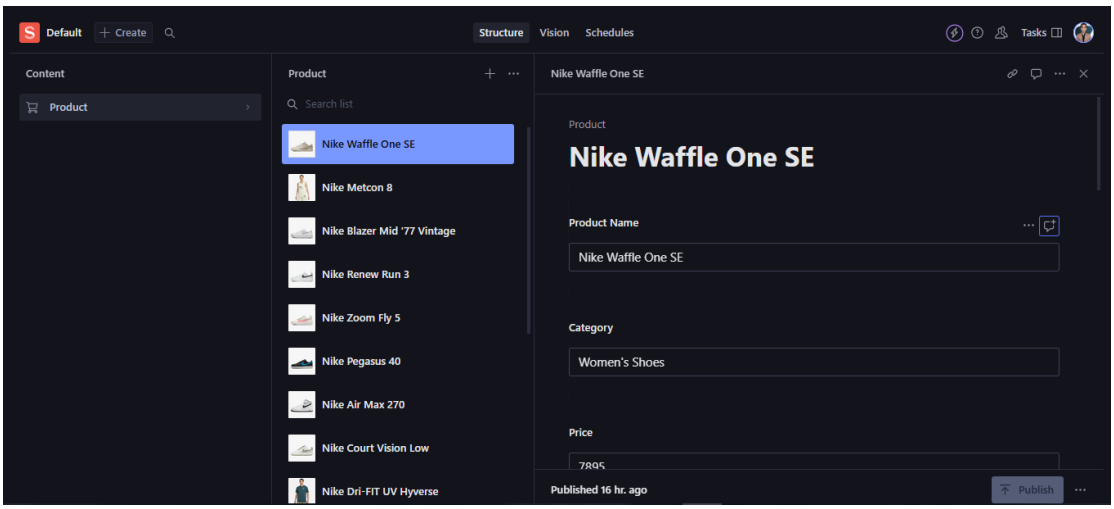
- Used `.env` files for secure API key management.
- Modularized code for better reusability and maintainability.
- Validated data during migration to ensure schema alignment.

Error Handling

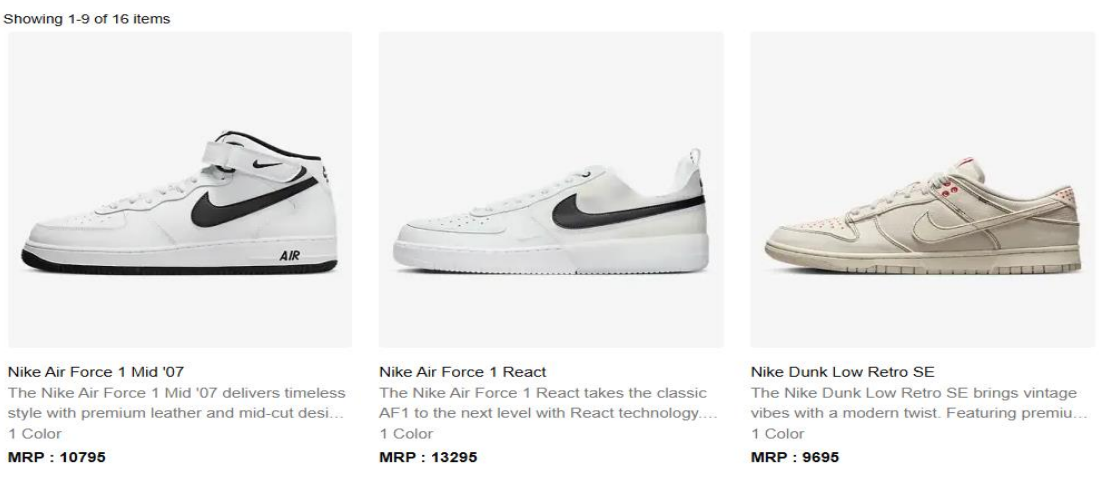
- Implemented error logging for debugging.
- Displayed user-friendly error messages on the frontend.
- Used fallback data to enhance the user experience in case of API failure.

Expected Output

- **Sanity CMS**: Populated with imported product data from API.



- **Frontend**: Functional product listings displayed using data fetched from Sanity CMS.



Conclusion

This documentation encapsulates the efforts on Day 3, highlighting the integration of API, successful data migration into Sanity CMS, the creation of a reusable helper function, and frontend display of products. The use of Typegen ensures type safety and enhances the development process, contributing to a robust and scalable shoe marketplace.