Day 3 - API Integration Report - General Ecommerce

Prepared By: Muhammad Rehan

API Integration Process

Overview:

This document outlines the API integration process, focusing on fetching and migrating data from an external API into Sanity CMS for the marketplace backend. The external API used is available at: `https://hackathon-apis.vercel.app/api/products`.

Steps Followed:

1. **Understanding API Documentation:**
    - Reviewed the provided external API endpoints.
    - Identified key endpoints such as product listings and categories.
2. **Authentication Setup:**
    - The API did not require authentication, simplifying the integration process.
3. **Endpoint Testing:**
    - Tested API calls using Postman.
    - Verified the structure and accuracy of the data returned.
4. **Backend Integration:**
    - Implemented functions in the backend to fetch and process API data.
    - Established reusable utility functions for querying Sanity CMS.

Adjustments Made to Schemas

Schema Changes:

1. **Sanity Schema Updates:**
    - Adjusted `product` schema to include fields such as `tags`, `dimensions`, and `category`.

    - Added a `category` schema to manage product categories.
2. **Field Mappings:**
    - API Field: `title` → Schema Field: `name`
    - API Field: `price` (number) → Schema Field: `price` (float)
    - API Field: `image` → Schema Field: `image.asset` (reference)
3. **Relationships:**
    - Linked `category` in the `product` schema as a reference field to the `category` schema.

Migration Steps and Tools Used

Tools:

- **Axios:** For making API calls.
- **Sanity Client:** For interacting with the Sanity CMS.
- **Slugify:** For generating slugs from product names.

Steps:

1. **Fetch Data from API:**
    - Used Axios to retrieve product data from the external API.
2. **Data Transformation:**
    - Mapped API data fields to match the Sanity schema.
    - Created a utility function to upload images to Sanity and return their references.
3. **Sanity Data Upload:**
    - Iterated over the transformed data to create or update records in Sanity.
4. **Verification:**
    - Verified imported data by querying it via the Sanity CMS Studio.

Challenges Faced and Solutions

Challenges:

1. **Data Format Mismatch:**
    - The API returned inconsistent field names compared to the Sanity schema.

    - **Solution:** Used a mapping layer in the migration script to standardize field names.
2. **Image Upload Errors:**
    - Timeout issues occurred while uploading large images.
    - **Solution:** Implemented error handling and retries for image uploads.
3. **Data Duplication:**
    - Duplicate entries in the API response.
    - **Solution:** Added logic to check for existing records in Sanity before creating new ones.

Screenshots

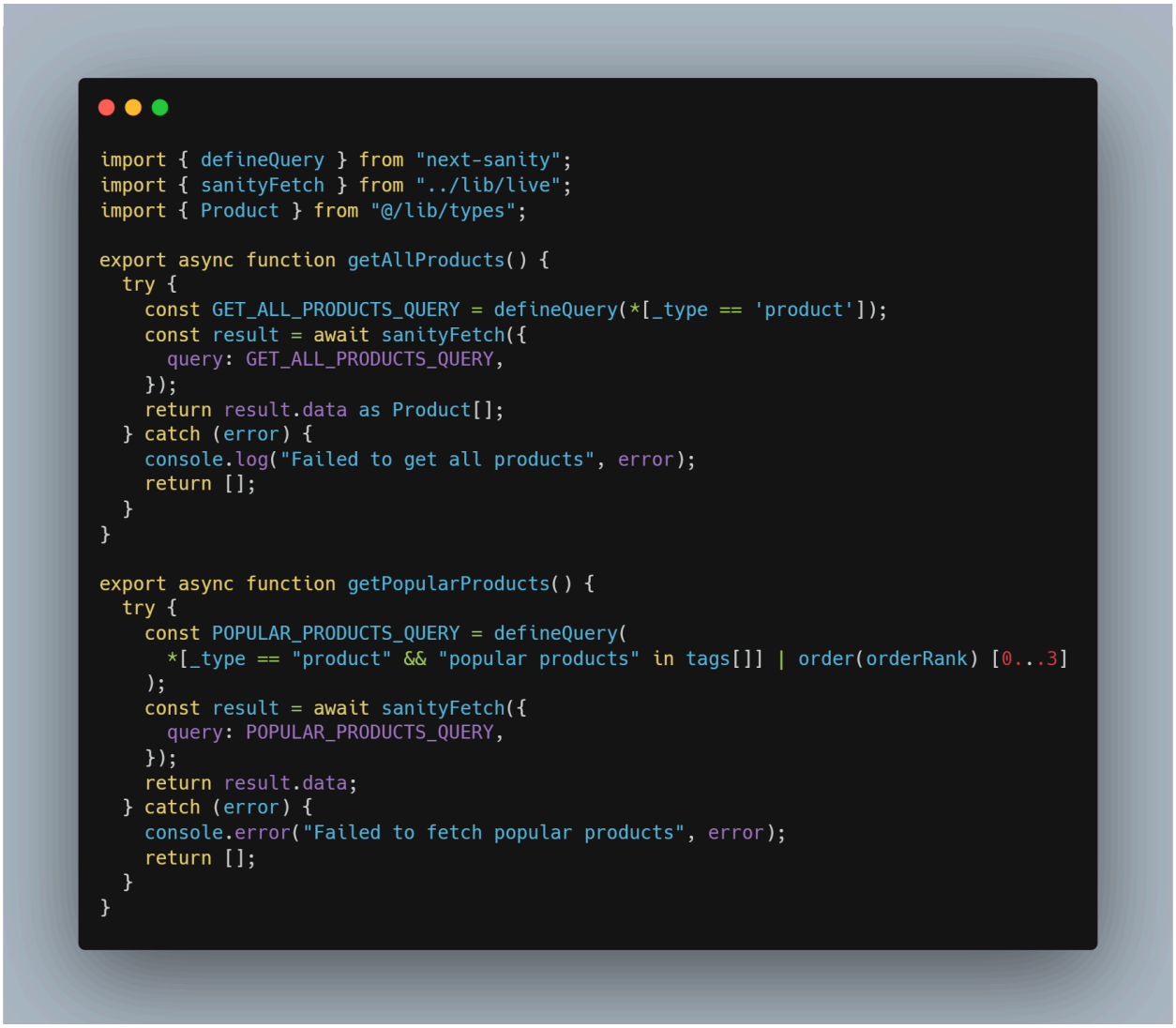
1. **API Calls:**
    - Screenshot of successful GET request to the external API.

2. **Frontend Data Display:**
    - Screenshot showing products rendered in the marketplace frontend.

3. **Sanity CMS:**
    - Screenshot of products and categories populated in Sanity CMS.

Code Snippets

API Integration Functions:



```
import { defineQuery } from "next-sanity";
import { sanityFetch } from "../lib/live";
import { Product } from "@/lib/types";

export async function getAllProducts() {
  try {
    const GET_ALL_PRODUCTS_QUERY = defineQuery(*[_type == 'product']);
    const result = await sanityFetch({
      query: GET_ALL_PRODUCTS_QUERY,
    });
    return result.data as Product[];
  } catch (error) {
    console.log("Failed to get all products", error);
    return [];
  }
}

export async function getPopularProducts() {
  try {
    const POPULAR_PRODUCTS_QUERY = defineQuery(
      *[_type == "product" && "popular products" in tags[]] | order(orderRank) [0...3]
    );
    const result = await sanityFetch({
      query: POPULAR_PRODUCTS_QUERY,
    });
    return result.data;
  } catch (error) {
    console.error("Failed to fetch popular products", error);
    return [];
  }
}
```

Migration Script:

```
import axios from 'axios';
import { client } from './sanityClient.js';
import slugify from 'slugify';

async function uploadImageToSanity(imageUrl: string): Promise<string | null> {
  try {
    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(),
    });
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}

async function importData() {
  try {
    const response = await axios.get('https://hackathon-apis.vercel.app/api/products');
    const products = response.data;
    let counter = 1;

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

      const sanityProduct = {
        _id: `product-${counter}`,
        _type: 'product',
        name: product.name,
        slug: {
          _type: 'slug',
          current: slugify(product.name, { lower: true, strict: true }),
        },
        price: product.price,
        tags: product.tags || [],
        image: imageRef ? { _type: 'image', asset: { _type: 'reference', _ref: imageRef } } :
undefined,
        description: product.description,
      };

      await client.createOrReplace(sanityProduct);
      console.log(`✅ Imported product: ${sanityProduct.name}`);
      counter++;
    }

    console.log('✅ Data import completed!');
  } catch (error) {
    console.error('Error importing data:', error);
  }
}

importData();
```

Summary

This report provided a comprehensive overview of the API integration and data migration process for the marketplace backend. By utilizing tools such as Axios, Sanity CMS, and Slugify, we successfully imported data from an external API, transformed it to match the Sanity schema, and verified its integrity. The reusable backend functions and scripts ensure scalability and flexibility for future development. This workflow demonstrates a practical approach to real-world API integration and CMS management challenges, equipping the team with essential skills for dynamic marketplace creation.