API Integration Report

Step 1: Fetch Data from API and Store in Sanity

1. Fetch data from the external API using tools like @sanity/client, axios or dotenv.
2. Process and format the API response to match the Sanity schema.
3. Use Sanity's client library to insert data into Sanity CMS using its GROQ query language.
4. Validate successful data insertion by checking the Sanity Studio dashboard.

```
scripts > JS importSanityData.mjs > ...
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.SANITY_API_TOKEN,
  apiVersion: '2021-08-31'
})
Tabnine | Edit | Test | Explain | Document
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
```

Step 2: Fetch Data from Sanity

1.Use Sanity's client library to query the data using GROQ.

2.Write queries to retrieve specific fields or filter data (e.g., fetch all products or specific categories).

3.Handle API responses, including errors or empty datasets.

4.Test the queries in Sanity's Vision tool to ensure accuracy.

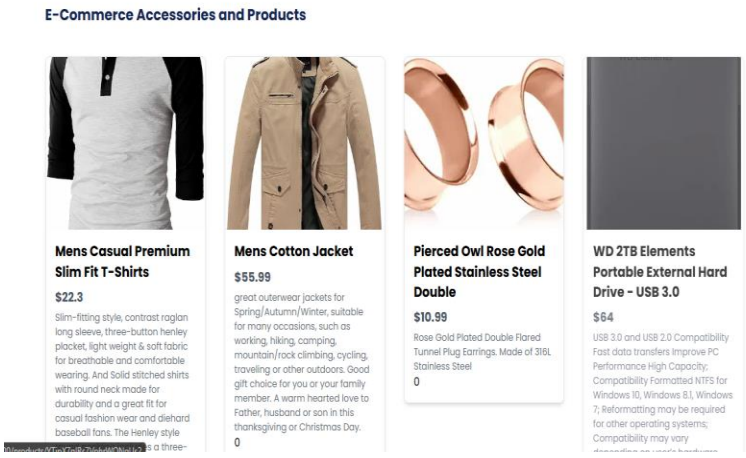
```
"use client"

import React, { useEffect, useState } from "react";
import Image from "next/image";
import Link from "next/link";
import { client } from "@/sanity/lib/client";

Tabnine | Edit | Explain
const Shop = () => {
  const [products, setProducts] = useState<any[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const query = `*[_type == "product"]{
          _id,
          name,
          description,
          price,
          quantity,
          discountPercentage,
          rating,
          tags,
          "image": image.asset->url
        }`;
        const result = await client.fetch(query);
        setProducts(result);
      } catch (error) {
        console.error("Error fetching products:", error);
      } finally {
        setLoading(false);
```

Step 3: Display Data in Frontend (Shop.tsx)

1.Create a React component (Shop.tsx) to fetch data from Sanity at runtime.

2.Use the useEffect hook to fetch data when the component mounts (if using client-side rendering).

3.Pass data as props if using server-side rendering (getServerSideProps or getStaticProps in Next.js).



4.Map over the data and display it in the desired UI format, including images, descriptions, and prices.

Step 4: Flow of Data

- **Data Flow Diagram**:
    1. **External API**: Data is fetched using an HTTP request (GET /endpoint).
    2. **Sanity CMS**: Processed data is inserted into Sanity using GROQ queries.
    3. **Frontend (Shop.tsx)**: Data is retrieved from Sanity and displayed to users.
    4. **Error Handling**: Any errors (e.g., API failure, data mismatch) are logged and handled gracefully at each stage.

Example Flow:

1. **Fetch Data from API**
    - External API endpoint: https://api.example.com/products
    - Data format: JSON (e.g., { id, name, price, image }).
2. **Store in Sanity**
    - Sanity schema: { _type: "product", name, price, description, image }.
3. **Fetch Data in Frontend**
    - GROQ query: *[_type == "product"]{ _id, name, price, image }.
4. **Display in Shop.tsx**
    - Data mapped into cards with Image, name, and price fields.

Conclusion:
This report outlines a structured API integration process to fetch data, store it in Sanity CMS, and display it in the frontend. The flow ensures that data is processed efficiently and rendered accurately, providing a seamless user experience.

Prepared by Kashaf Tariq