Day 3 - API Integration Report General E-Commerce

API Integration and data Migration Process

1. Objectives:

- **Integrate APIs to fetch and dynamically display data on the frontend.**
- **Establish seamless communication between Sanity CMS and the frontend using Next.js.**

2. Steps Followed:

- **Identified the required APIs to retrieve product, category, and user-related data.**
- **Configured API endpoints within the Next.js application.**
- **Utilized the fetch function in useEffect hooks to make API calls.**
- **Validated API responses using TypeScript interfaces for type safety.**
- **Debugged and resolved discrepancies in the API responses to ensure accuracy.**

3. Challenges and Solutions:

- **Challenge: Handling inconsistent API data formats.**
  Solution: Mapped API responses to predefined TypeScript interfaces to standardize and ensure type safety.

- **Challenge: Slow response times during API calls.**
  Solution: Implemented caching mechanisms and optimized API queries for enhanced performance.

Adjustments Made to Schemas

1. Sanity CMS Schemas:
Updated the product schema to include additional fields for API integration:

- **_id**
- **title**
- **price**
- **productImage: productImage.asset->url**
- **description**
- **isNew**
- **tags[]**
- **_type**
- **category**

Migration Steps and Tools Used

1. Migration Process:

- **Configured the .env file with the following values:**

NEXT_PUBLIC_SANITY_PROJECT_ID="abc "

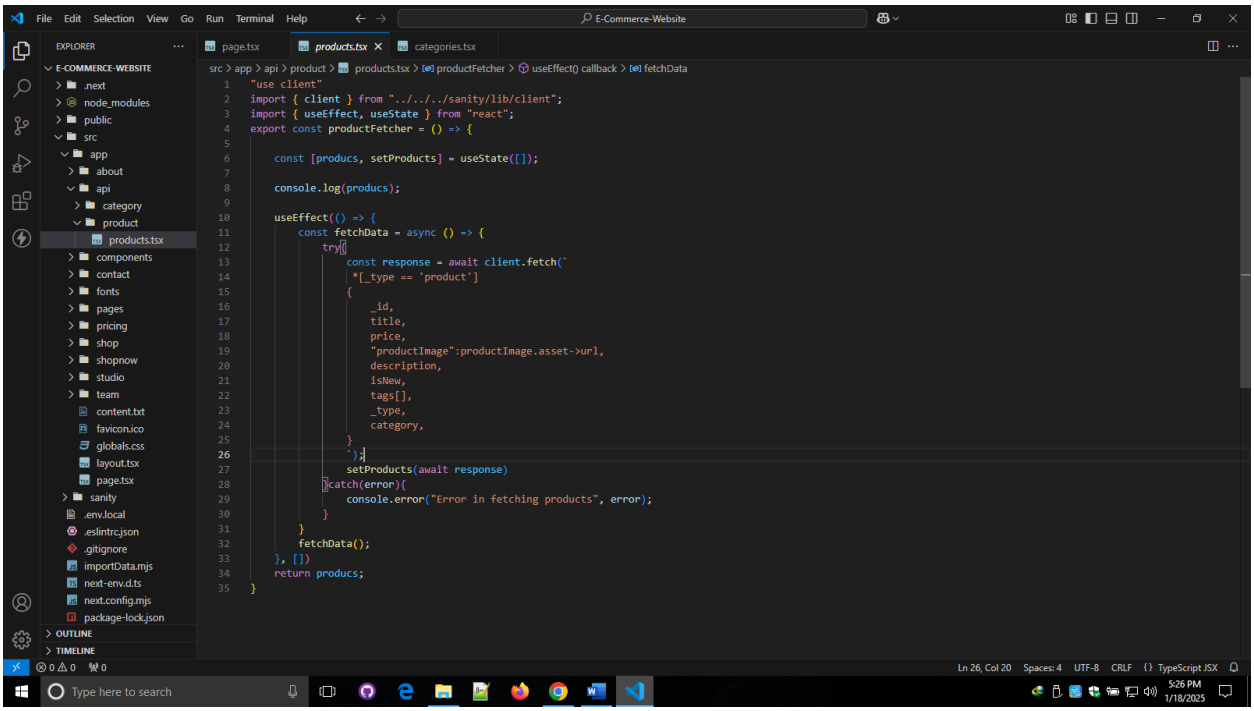
NEXT_PUBLIC_SANITY_DATASET="xyz "

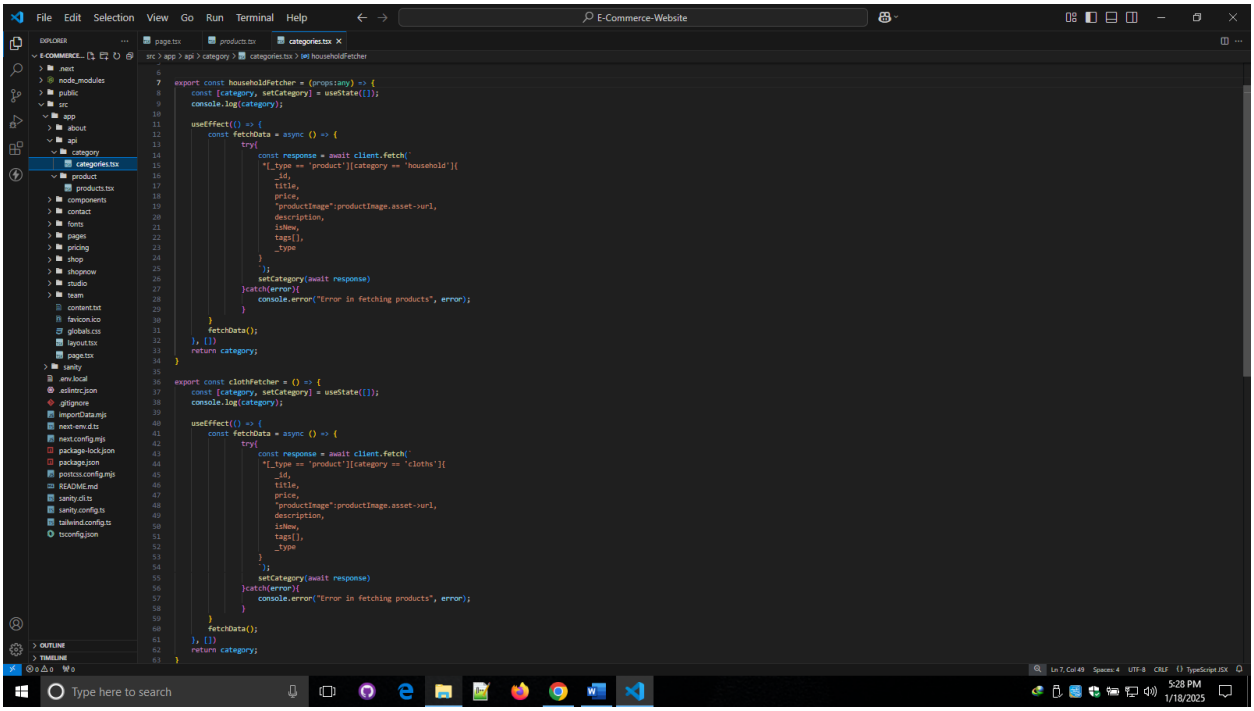
SANITY_API_TOKEN="a-z "

- **Imported data from the given API into Sanity CMS:**
    - **Created a custom script file (importData.mjs) to handle data transformation.**
    - **Executed the script with the command: node importData.mjs.**
    - **Successfully imported data into Sanity CMS.**

2. Tools Used:

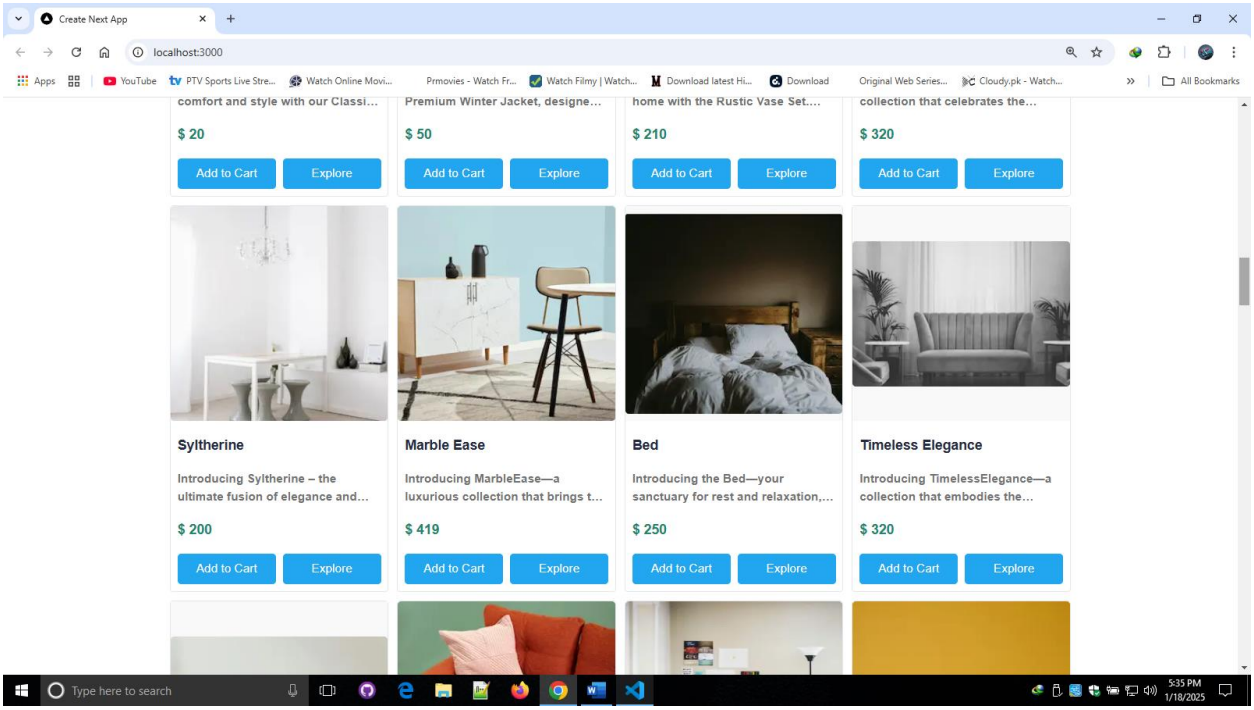
- **Sanity CLI: For exporting and importing datasets.**
- **Custom Scripts: Script written in importData.mjs for data migration and transformation.**
- **Postman: Used for testing API endpoints during the integration process.**

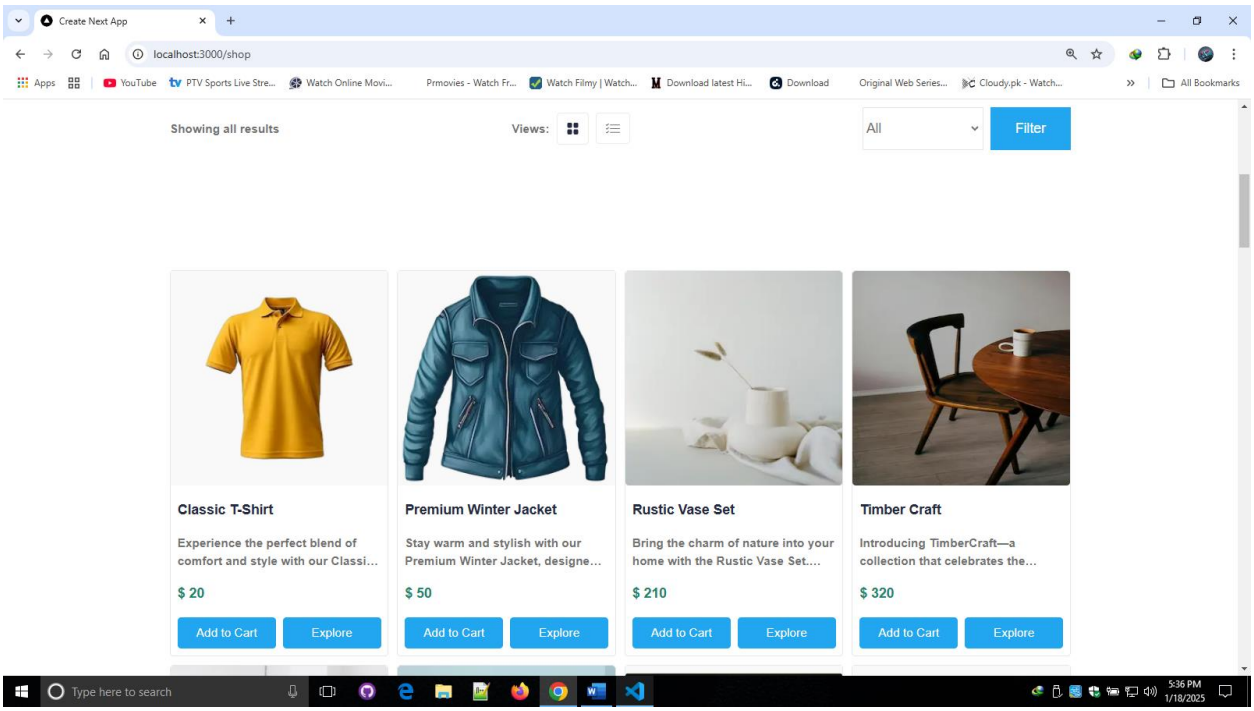

File Edit Selection View Go Run Terminal Help
E-Commerce-Website
EXPLORER
E-COMMERCE-WEBSITE
page.tsx
products.tsx
categories.tsx
src > app > api > product > products.tsx > productFetcher > useEffect() callback > fetchData
"use client"
import { client } from "../../../sanity/lib/client";
import { useEffect, useState } from "react";
export const productFetcher = () => {
const [producs, setProducts] = useState([]);
console.log(producs);
useEffect(() => {
const fetchData = async () => {
try{
const response = await client.fetch(`
*[_type == 'product']
{
_id,
title,
price,
"productImage":productImage.asset->url,
description,
isNew,
tags[],
_type,
category,
}
`);
setProducts(await response)
}catch(error){
console.error("Error in fetching products", error);
}
}
fetchData();
}, [])
return producs;
}
OUTLINE
TIMELINE
Ln 26, Col 20 Spaces: 4 UTF-8 CRLF TypeScript JSX
Type here to search
5:26 PM
1/18/2025


File Edit Selection View Go Run Terminal Help
E-Commerce-Website
export const householdFetcher = (props:any) => {
const [category, setCategory] = useState([]);
console.log(category);
useEffect(() => {
const fetchData = async () => {
try{
const response = await client.fetch(`
*[_type == 'product'][category == 'household']{
_id,
title,
price,
"productImage":productImage.asset->url,
description,
isNew,
tags[],
_type
}
`);
setCategory(await response)
}catch(error){
console.error("Error in fetching products", error);
}
}
fetchData();
}, [])
return category;
}
export const clothFetcher = () => {
const [category, setCategory] = useState([]);
console.log(category);
useEffect(() => {
const fetchData = async () => {
try{
const response = await client.fetch(`
*[_type == 'product'][category == 'cloths']{
_id,
title,
price,
"productImage":productImage.asset->url,
description,
isNew,
tags[],
_type
}
`);
setCategory(await response)
}catch(error){
console.error("Error in fetching products", error);
}
}
fetchData();
}, [])
return category;
}
Type here to search
5:28 PM
1/18/2025

2. Data Rendered on the Frontend:

- **Screenshots showcasing successfully fetched data displayed on the website's homepage, product listings, and category sections.**





3. Populated Sanity CMS Fields:

- **Screenshots highlighting the fields populated in the Sanity CMS dashboard after successful migration and API integration.**