Document Title: "Day 3 - API Integration Report -E-COMMERCE (Nike Store)"

1. Introduction

This report outlines the activities and progress made on Day 3 of the project, specifically focusing on API Integration and Data Migration for the given template 3. The objective of this phase was to successfully integrate external APIs, adjust backend schemas, and migrate data into the Sanity CMS for further use on the frontend.

2. API Integration Process

2.1 API Selection

The first step in the API integration process involved selecting appropriate external APIs to fetch essential marketplace data. The key APIs chosen were responsible for retrieving information such as:

2.2 Setting up API Clients

The integration began with setting up API calls using the fetch() method. The calls were made asynchronous to ensure the smooth execution of the application without blocking other processes. Below is an example of the basic structure used for fetching API data:

Name, Category, price, Description, Available Colors and inventory of each product.

given api: https://template-03-api.vercel.app/api/products

2.3 Handling API Responses

Once the data was fetched from the external API, it was processed and stored in the application's state using React hooks (useState and useEffect). This allowed for dynamic updates to the frontend based on the fetched data.

2.4 Error Handling and Fallbacks

To ensure a smooth user experience, error handling was implemented. In case of any issues with the API call, fallback content was provided to maintain the integrity of the user interface.

3.No Adjustments Made to Schemas

Provided Scheme is used to integrate the data fetched from the external API into the backend.

3.1 Frontend State Adjustment

In the frontend, I adjusted the state structure to store the fetched API data. The useState and useEffect hooks were used to handle the dynamic nature of the product data. The following code snippet demonstrates the use of these hooks:

```
"use client"

import Image from "next/image";

import React, { useEffect, useState } from "react";

import Link from 'next/link';


const fetchProducts = async () => {

  const response = await fetch("https://template-03-api.vercel.app/api/products");

  const data = await response.json();
```

```
  return data.data; // Returning products from the response
};

const ProductList: React.FC = () => {
  const [products, setProducts] = useState<any[]>([]); // Initialize as empty array
  const [loading, setLoading] = useState<boolean>(true); // For loading state
  const [error, setError] = useState<string>("");

  useEffect(() => {
    const getProducts = async () => {
      try {
        const data = await fetchProducts(); // Fetch products from the API
        setProducts(data); // Update state with fetched products
        setLoading(false); // Set loading to false once data is fetched
      } catch (err) {
        console.error("Error fetching products:", err);
        setError("Failed to load products");
        setLoading(false); // Set loading to false even if there is an error
      }
    };

    getProducts();
  }, []);

  // Show loading message if still fetching data
  if (loading) {
```

```
  return <div>Loading products...</div>;
 }

 // Show error message if fetching failed
 if (error) {
  return <div>{error}</div>;
 }

 return (
  <div className="container mx-auto p-4">
   <h1 className="text-center font-bold text-3xl mb-10">Nike Products</h1>
   <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-8 pb-10 border-b-2">
    {products.map((product: any) => (
     <div
      key={product.productName}
      className="transform hover:scale-105 transition duration-300 ease-in-out p-4 rounded-lg bg-white shadow-lg hover:shadow-2xl hover:bg-gray-50"
     >
      <Link href={`http://localhost:3000/overallproducts/${product.productName}`} passHref>
       <div>
        <Image src={product.image} alt={product.productName} width={500} height={500} className="w-full h-48 object-contain mb-4 rounded-md" />
        <h2 className="text-xl font-semibold">{product.productName}</h2>
        <p className="text-sm text-gray-500">{product.category}</p>
        <p className="text-lg font-semibold mt-2">${product.price}</p>
```

```
            <p className="text-gray-700 mt-4">{product.description}</p>
            <div className="mt-4">
              <p className="text-sm text-gray-600">Available colors:
{product.colors.join(", ")}</p>
              <p className="text-sm text-gray-600">Inventory: {product.inventory}</p>
            </div>
          </div>
        </Link>
        <div className="mt-6 flex justify-center">
          <button className="bg-blue-600 text-white font-semibold py-2 px-6
rounded-lg transition duration-300 hover:bg-blue-700 focus:outline-none">
            Add to Cart
          </button>
        </div>
      </div>
    ))}
  </div>
</div>
);
};

export default ProductList;
```

4. Data Migration Steps and Tools Used

4.1 Migration Plan

The process of migrating data from the external API to the Sanity CMS involved several steps:

Data Structure Review: A thorough review of the API data structure was done to ensure compatibility with the existing Sanity schema.

Data Population: A custom Node.js script was written to fetch the API data and insert it into the Sanity CMS using the Sanity client.

4.2 Tools Used

The following tools were used during the data migration process:

Sanity CLI: To manage and push data into the Sanity CMS.

Node.js Migration Script: A provided script was utilized to automate the process of fetching data from the API and migrating it to the CMS.

4.3 Verification of Data

Once the data migration was complete, I manually verified that the data had been correctly inserted into the Sanity CMS. This included confirming that all fields, such as product names, prices, and descriptions, were populated correctly.

5. Screenshots

API Calls:

-Screenshot from the network tab showing the successful API call and response.

Data Displayed in the Frontend:

-Screenshot showing the dynamic display of product data on the marketplace frontend.

Populated Sanity CMS Fields:

-Screenshot from the Sanity dashboard showing the populated fields for the product document.

6. Code Snippets for API Integration and Migration Scripts are also attached.

7. Conclusion

The API integration and data migration process for Day 3 was successfully completed, with all required data fetched from the external API and migrated into the Sanity CMS. The frontend was updated to dynamically display the product information, and the system was thoroughly tested to ensure smooth functionality.