Prepared by : Shahzeena Samad
Date : 17th January 2025

Day 3 - API Integration and Data Migration Report - Shoe World

1. API Integration Process:

The API integration process for Shoe World involved multiple stages to fetch product data from an external API and display it on the frontend. Here's an overview of the steps followed:

1. **API Provided:** The external API provided access to product-related data, including product names, descriptions, prices, and images.

2. **Sanity Project Creation:** A project was created on Sanity.io to manage the product data fetched from the external API. This CMS was chosen to dynamically display products in the store.

3. **API Token Configuration:** The provided API token was securely stored in the `.env` file to ensure authorized access to the external API.

4. **API Integration:**
   - GET requests were sent to fetch product data using the API token.
   - The data fetched was processed and formatted on the backend before being inserted into the system.
   - Handling errors and formatting data to ensure proper frontend display was part of the integration process.

5. **Frontend Display:** After processing the data, it was dynamically rendered on the Shoe World frontend, showcasing product names, descriptions, prices, and images on the store pages.

```tsx
import { sanityFetch } from "@/sanity/lib/fetch";
import { allproducts } from "@/sanity/lib/queries";
import ProductsClient from "../components/productClient"; // Client-Side Component for Products

type Product = {
  _id: string;
  productName: string;
  description: string;
  price: number;
  colors: string[];
  inventory: number;
  category: string;
  status: string;
  imageUrl: string;
};

export default async function ProductsPage() {
  // Fetch products data server-side
  const products: Product[] = await sanityFetch({ query: allproducts });

  // Pass products to the client component
  return (
    <div className="min-h-screen p-8">
      <ProductsClient products={products} />
    </div>
  );
}
```

2. Adjustments Made to Schemas:

The schema in Sanity.io was adjusted to store the product data retrieved from the external API. Key schema modifications included:

- **Sanity Schema Creation:** A new schema for storing product details was created. This schema includes fields such as:
  - **productId:** Unique identifier for each product.
  - **name:** Name of the product.
  - **description:** Detailed description of the product.
  - **price**: Price of the product.
  - Additional fields like imageUrl and category were added to reflect the marketplace's needs.
  -
- **Schema Modification:** The schema was aligned with the API's data structure to ensure smooth data storage and retrieval.

```
export const productSchema = {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'productName',
      title: 'Product Name',
      type: 'string',
    },
    {
      name: 'category',
      title: 'Category',
      type: 'string',
    },
    {
      name: 'price',
      title: 'Price',
      type: 'number',
    },
    {
      name: 'inventory',
      title: 'Inventory',
      type: 'number',
    },
    {
      name: 'colors',
      title: 'Colors',
      type: 'array',
      of: [{ type: 'string' }],
    },
```

3. Migration Steps and Tools Used:

Data migration was carried out to move product data from the external API to Sanity CMS:

- **Data Fetching:** Product data like name, description, price, and image was retrieved via the external API.

- **Migration Script:** A script was created to map the fetched data to Sanity's schema. The steps included:
  - Fetching data from the external system.
  - Mapping the fetched data to the new schema in Sanity CMS.
  - Inserting the data into Sanity Studio

  .

- **Build and Data Import:**
  - The `npm run build` command was executed to trigger the migration, ensuring the data was imported into Sanity CMS.
  - The Sanity Vision tab was used to verify that the data was correctly imported.

- **GROQ Query:** After migration, a GRoQ query was used to fetch the data from Sanity CMS and display it on the frontend.

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

// Define __filename and __dirname for ES Modules
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

// Load environment variables from .env.local
dotenv.config({ path: path.resolve(__dirname, '../.env.local') });

// Create Sanity client
const client = createClient({
  projectId:"[redacted]",
  dataset:"production",
  useCdn: false,
  token:"skVsOmeVK6DFxv7XEC7Hmg2WSvGuPuquNWUffI9gTaWtyyiorJEx3j5R8nfNzGi6HeguUhyDC7uRX2G3UyAWVcRgkTZEbYmJOZNFBmPOTcjQp53OfyewsxfnSnjHAKpJ
  apiVersion: '2021-08-31',
});

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}`);
```

4. Data Successfully Displayed on Frontend:

The data was displayed successfully on the Shoe World frontend:

- **Frontend Data Display:** Product data was dynamically fetched from Sanity CMS and rendered on product pages using frontend technologies like React.js.

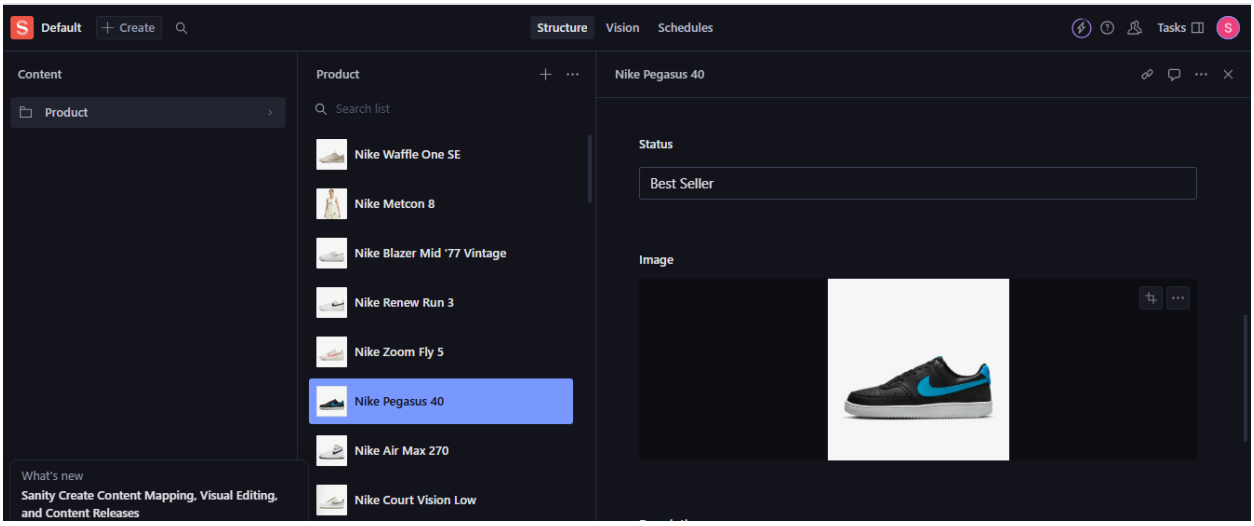
- **Frontend Testing:** The product details, including name, description, price, and images, were tested on the local server and displayed correctly.

5. Populated Sanity CMS Fields:

Once the migration was completed, the product data was populated into the fields of Sanity CMS:

- **Data Population:** Verified in Sanity Studio, the product information was correctly populated in fields such as productId, name, description, and price.

- **Verification:** All fields were checked to ensure the data appeared correctly in the system.



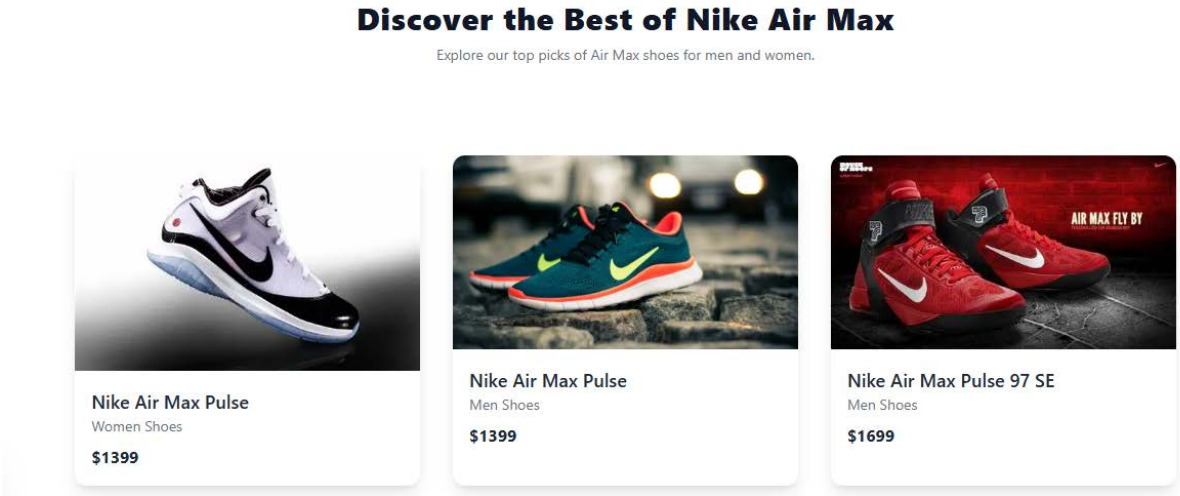
Code Snippets for API Integration and Migration Scripts:

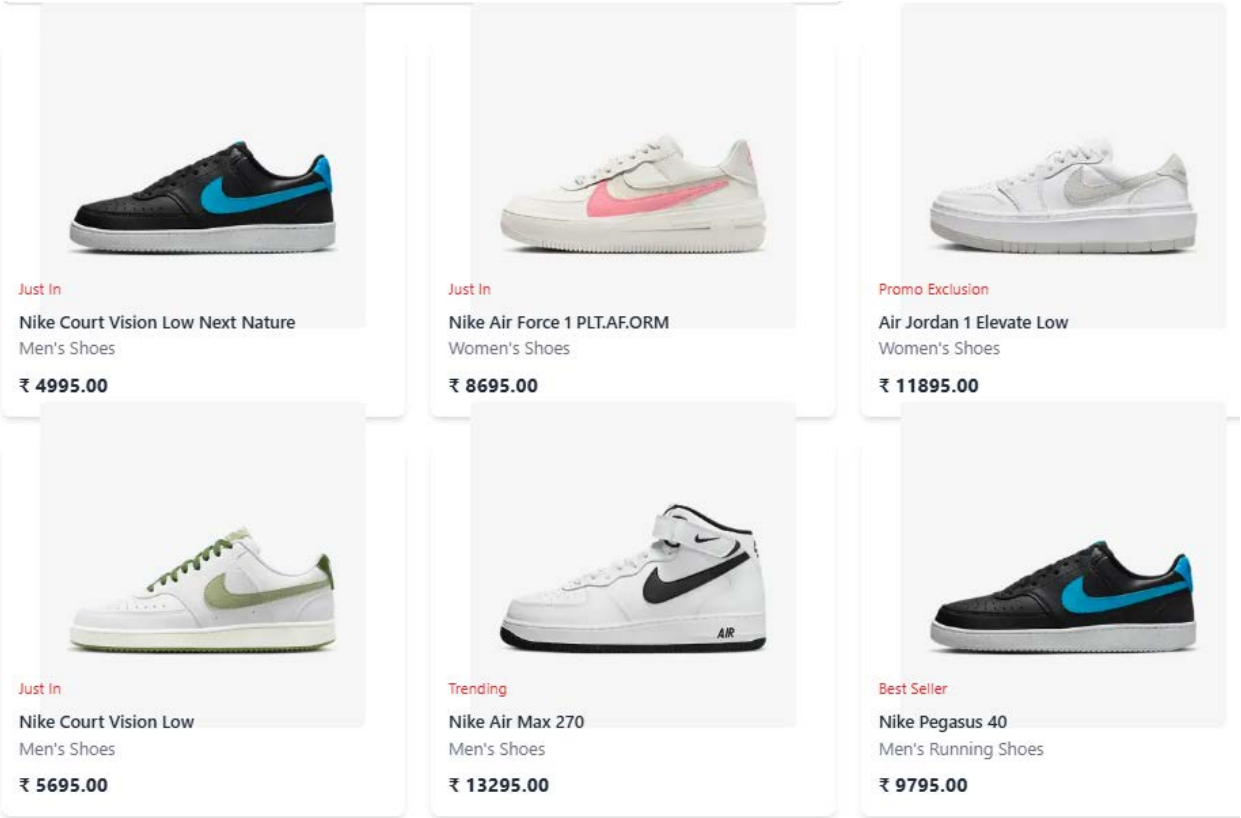
1. Queries Implementation

```
import { defineQuery } from "next-sanity";


export const allproducts = defineQuery(`
    *[_type == "product"]{
    _id,
    productName,
    description,
    price,
    inventory,
    status,
    colors,
    category,
    "imageUrl": image.asset->url
    }`)
```

2. Frontend Display





Self-Validation Checklist:

1. **API Understanding:**
   ✓
2. **Schema Validation:**
   ✓
3. **Data Migration:**
   ✓
4. **API Integration in Next.js:**
   ✓
5. **Submission Preparation:**

Conclusion:

In conclusion, this report outlines the successful execution of API integration and data migration for Shoe World, showcasing a robust and scalable solution for managing product data. The external API was seamlessly integrated to fetch product data, which was efficiently migrated to Sanity CMS for centralized storage and content management. A well-structured schema was designed and implemented in Sanity CMS to ensure compatibility, enabling smooth data storage, retrieval, and management.

The frontend dynamically displayed the product data, leveraging the power of Sanity's GROQ queries for optimized data fetching and validation. Each step of the process—from API integration to data migration and frontend rendering—was carefully verified to ensure accuracy and consistency. This approach not only enhanced the overall system performance but also established a reliable framework for handling product information.

The successful completion of this project highlights the importance of strategic API integration, thoughtful schema design, and the use of modern CMS tools like Sanity to create a seamless and efficient user experience. These advancements pave the way for future scalability, providing Shoe World with a robust platform to manage and display product data dynamically.