Day 3 - API Integration Report

Introduction:

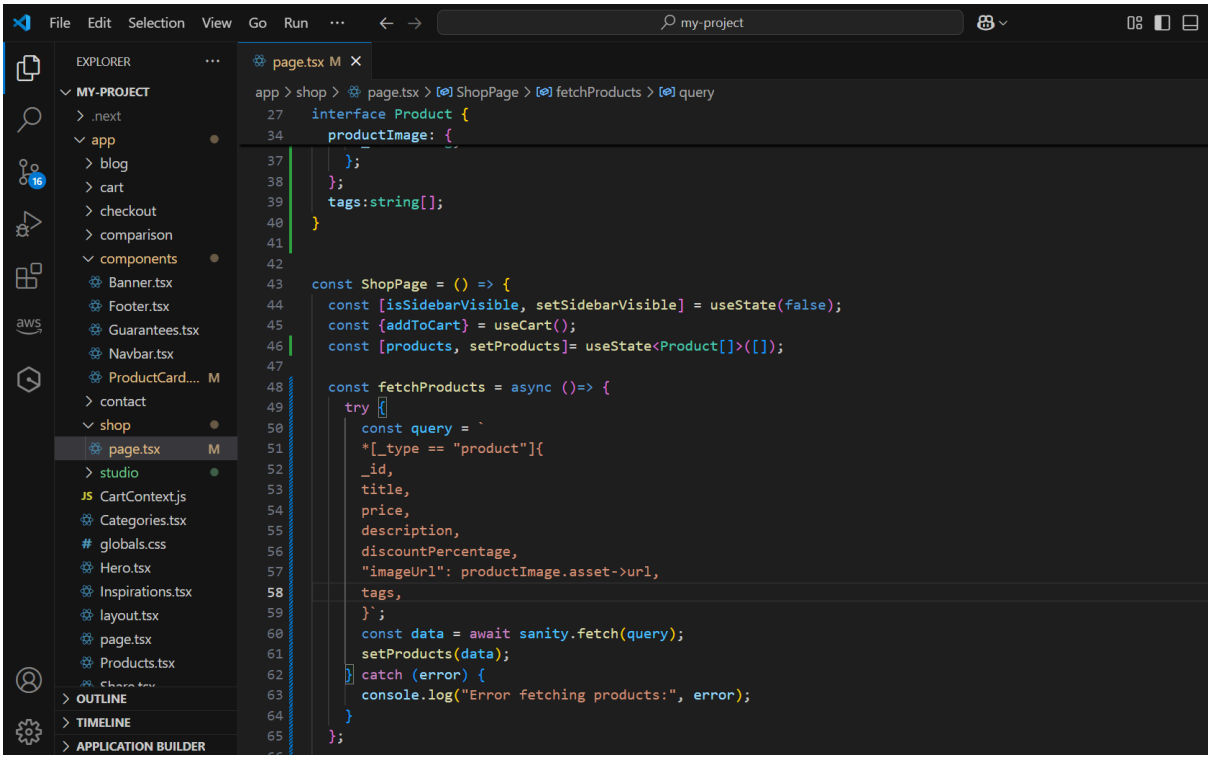
This document details the technical foundation of my e-commerce marketplace, focusing on API integration, schema adjustments, and data migration processes. It highlights how the marketplace effectively manages product data using Sanity CMS and ensures seamless frontend display, offering a scalable solution for a diverse product catalog.

- API integration process
- Adjustments to product schemas
- Data migration steps and tools
- Screenshots illustrating API calls, data display in the frontend, and CMS population
- Code snippets for API integration and data migration scripts

Technical Foundation

1. API Integration Process

- **API Details:**
  - **Hosted at: https://template6-six.vercel.app/api/products**
  - Provides product details, including:
    - Product name
    - Price
    - Description
    - Tags
    - Image
- **Integration Approach:**
  - Configured the Sanity CMS to interact with the API.
  - Utilized the `fetch` function to retrieve product data dynamically.



- Implemented error handling to ensure smooth data fetching.
- Verified successful integration by testing API calls within Sanity Vision.

2. Adjustments Made to Schemas

The product schema in Sanity CMS was tailored to align with the API structure:

- **Title:** Product name
- **Price:** Product cost
- **Description:** Detailed product information
- **Tags:** Relevant keywords for categorization
- **Product Image:** Reference to image assets uploaded to Sanity CMS

This customization ensures accurate and seamless data storage and retrieval.

```typescript
import { defineType } from "sanity"

export const product = defineType({
    name: "product",
    title: "Product",
    type: "document",
    fields: [
        {
            name: "title",
            title: "Title",
            validation: (rule) => rule.required(),
            type: "string"
        },
        {
            name:"description",
            type:"text",
            validation: (rule) => rule.required(),
            title:"Description",
        },
        {
            name: "productImage",
            type: "image",
            validation: (rule) => rule.required(),
            title: "Product Image"
        },
        {
            name: "price",
            type: "number",
            validation: (rule) => rule.required(),
            title: "Price",
        },
        {
```

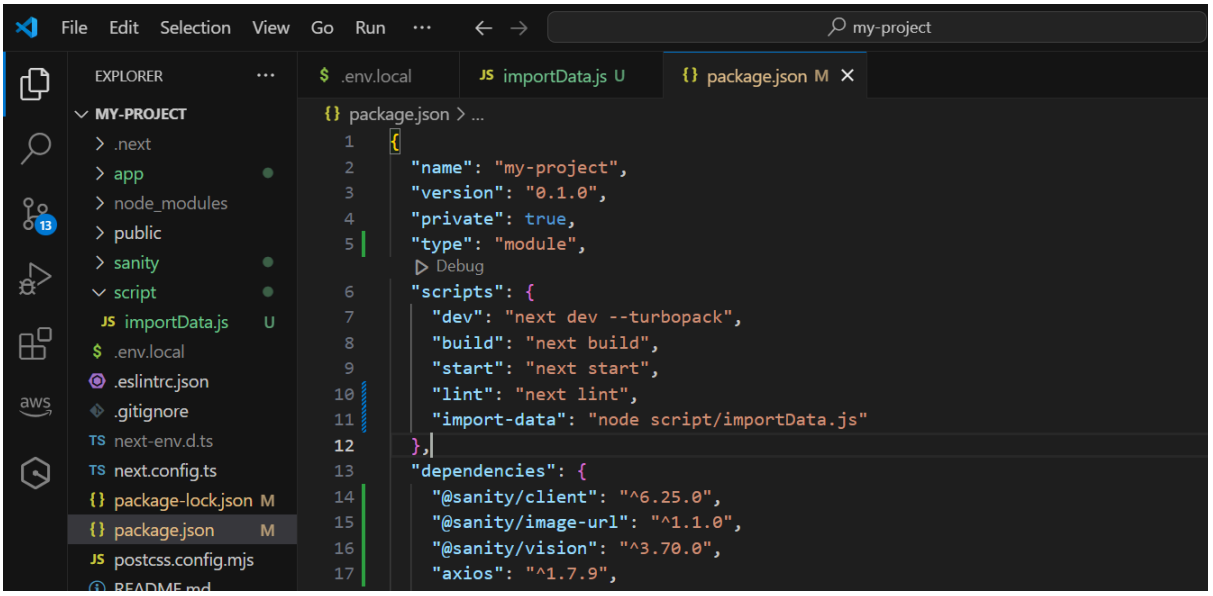
3. Migration Steps and Tools Used

- **Steps Followed for Data Migration:**
    - Created a migration script (`importData.js`) to fetch data from the API and upload it to Sanity CMS.

```javascript

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);

    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }

    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);

    const asset = await client.assets.upload('image', bufferImage, {
      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;
  }
}

async function uploadProduct(product) {
  try {
```

    - Installed necessary dependencies for Sanity client interaction.
    - Updated the project configuration to include `type: "module"` for compatibility.

- Added a custom script in the `package.json` file to automate the migration process.

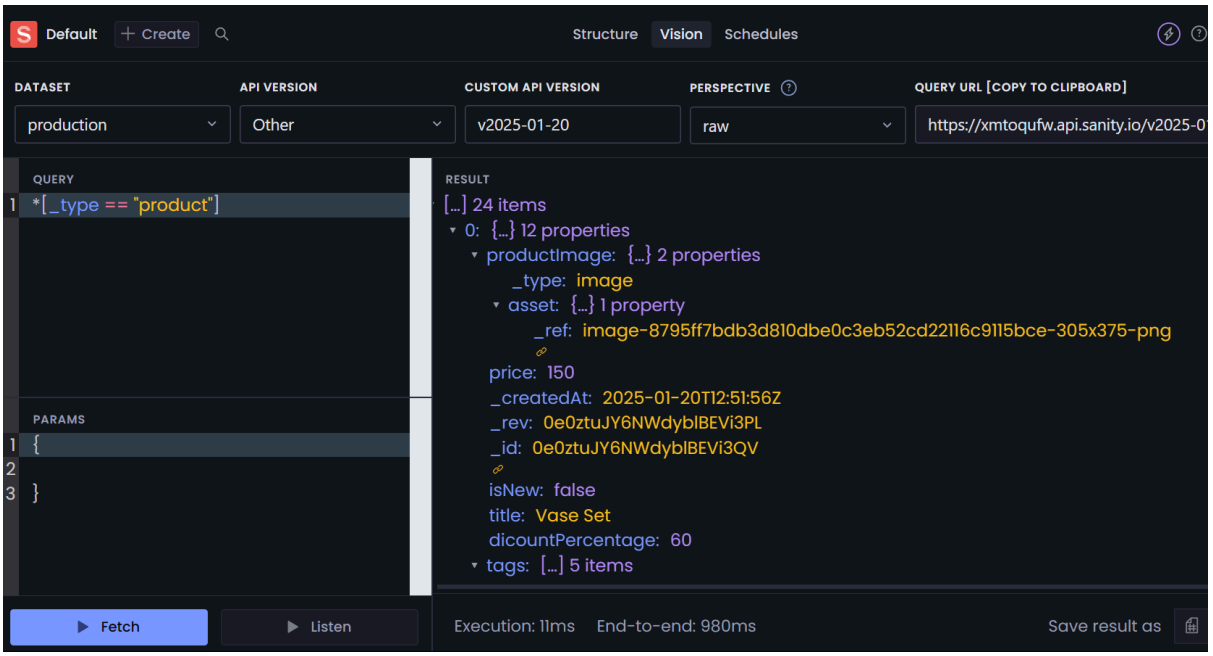


- Executed the migration script using the command `npm run import-data`.
- Validated the imported data in the Sanity CMS to ensure accuracy and completeness.

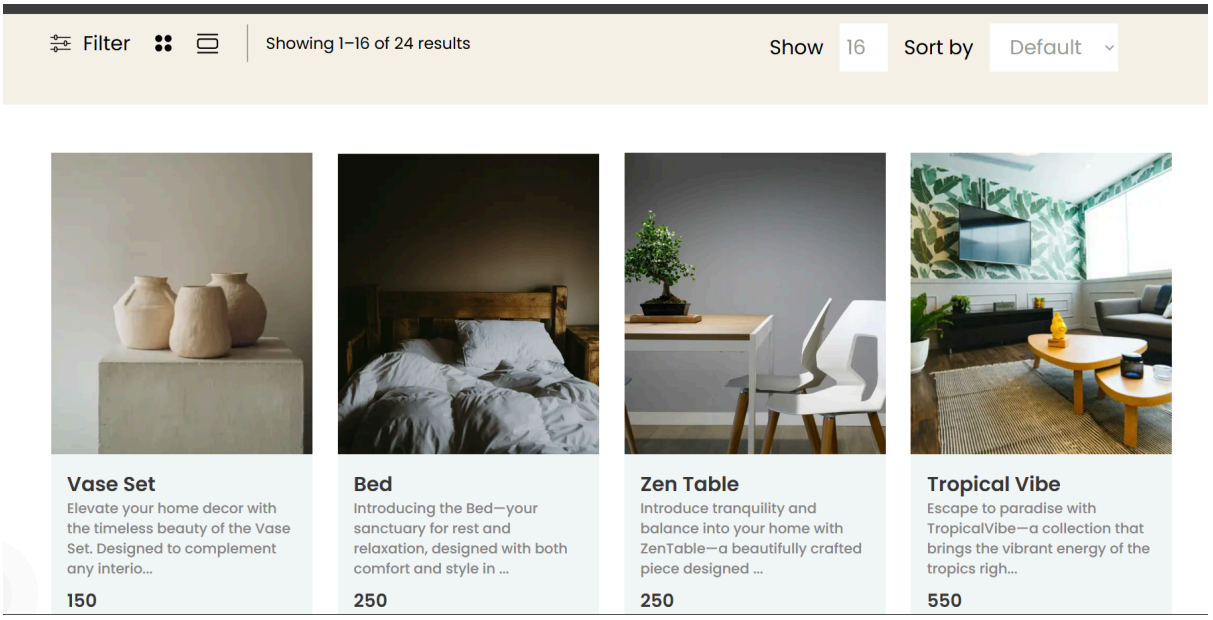
- **Tools Used:**
  - **Sanity CMS:** For content management and hosting.
  - **Sanity Vision:** To test API calls and data integrity.
  - **Environment Variables:** Secured sensitive keys via `.env.local`.

Screenshots Provided

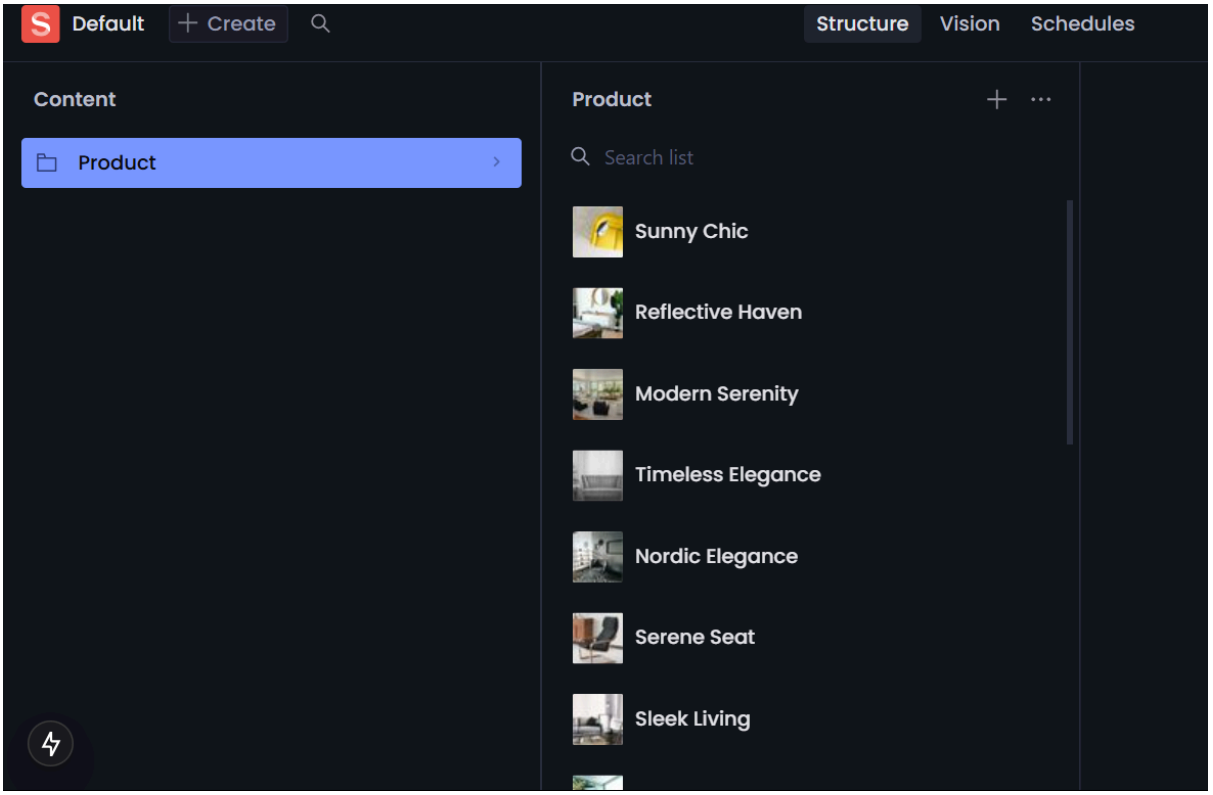
- **API Call Results in Sanity Vision:** Displays successful data fetching from the API.



- **Frontend Data Display:** Validates dynamic rendering of product data on the website.



- **Populated Sanity CMS Fields:** Confirms that the imported data is stored correctly in the CMS.



Conclusion:

This document captures the essential steps and processes that underpin the development of a robust and user-friendly e-commerce platform. By leveraging the power of APIs, customizing schemas, and ensuring smooth data migration, we have built a solid foundation to provide a seamless shopping experience.