Day 3 - API Integration and Data Migration Report - [Bandage-App]

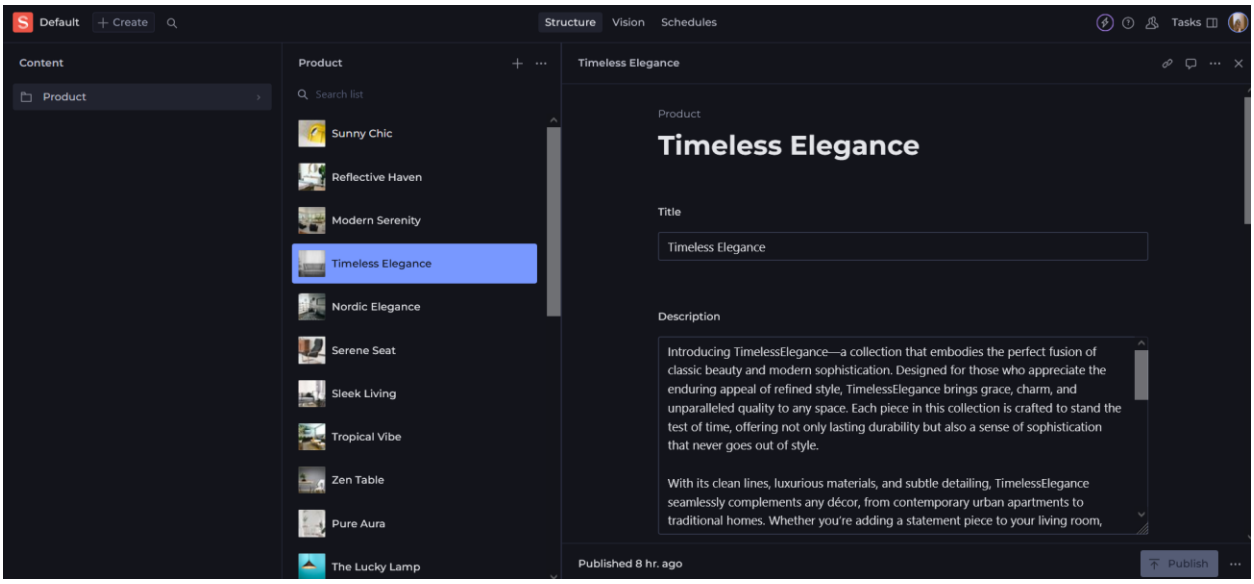
Overview

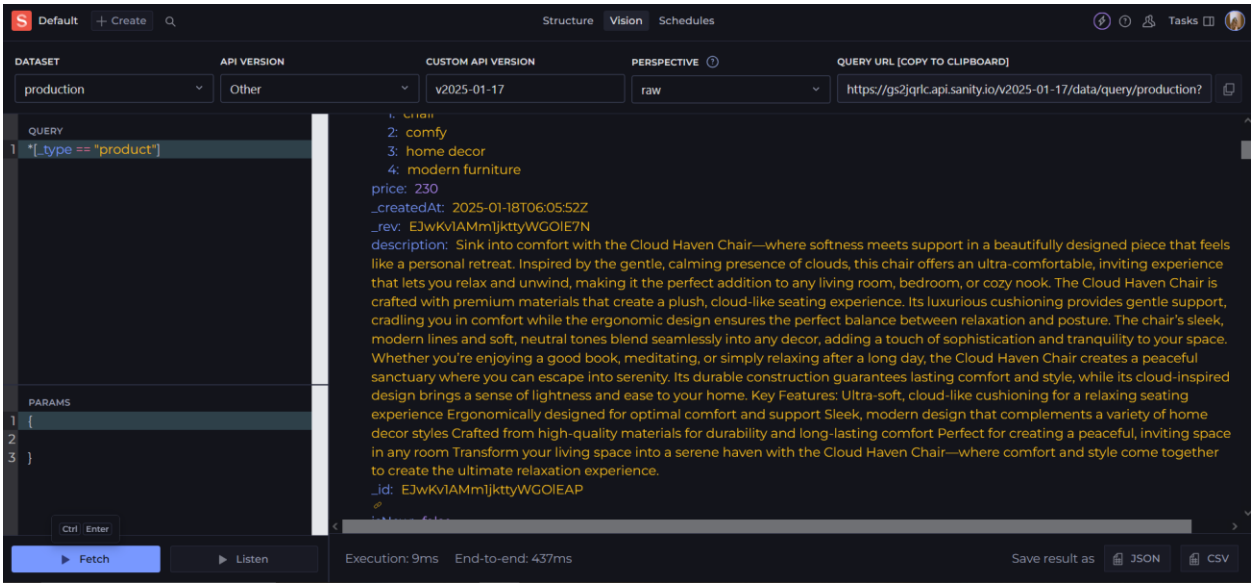
This document summarizes the process I followed to integrate APIs and migrate data into the Sanity CMS while ensuring a functional Next.js frontend. The objective was to populate the CMS with accurate data, align schemas, and build a robust frontend for data display.

Process Overview

1. Understanding the Provided API

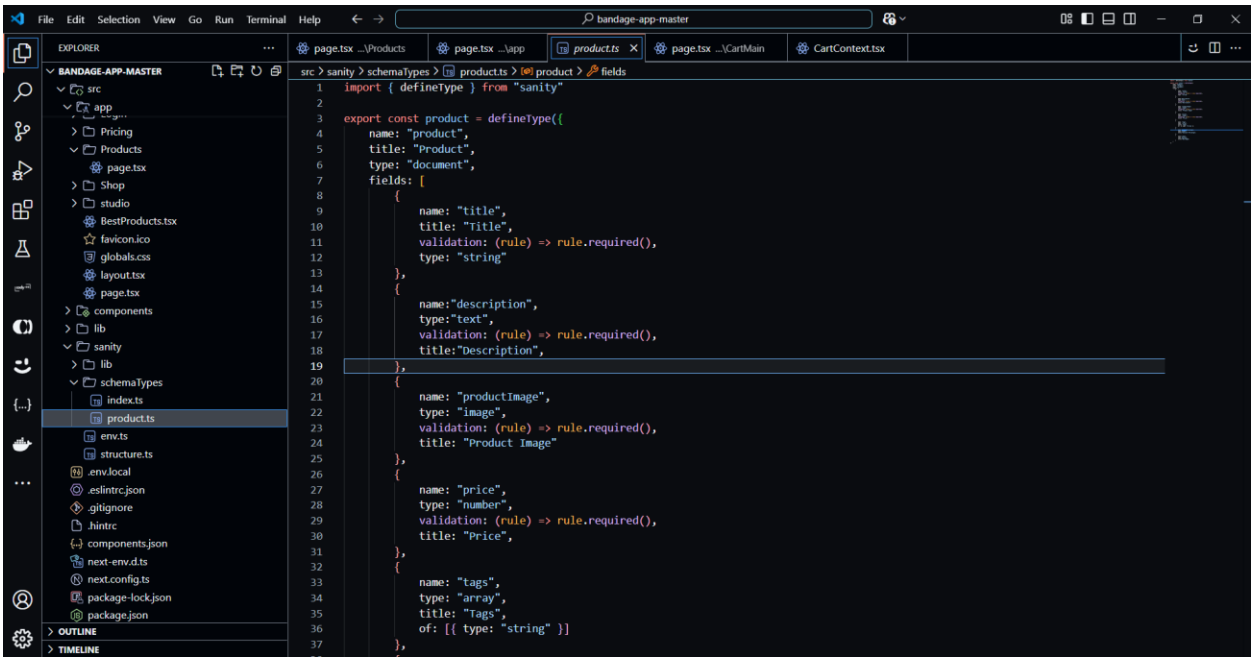
- **API Documentation Review:**
  - Identified key endpoints:
    - `/products`: For product listings.
    - `/categories`: For categories.
    - `/orders`: For order history.
  - **Tools Used:** Postman and browser developer tools to test endpoints and responses.





2. Schema Validation and Adjustments

- Compared the Sanity CMS schema with the API data structure.
    - Example Adjustment:
        - API Field: `product_title`
        - Schema Field: `name`
        - **Action:** Updated the schema to match API field names and types.
- Ensured relationships between categories and products were established.



3. Data Migration Methods

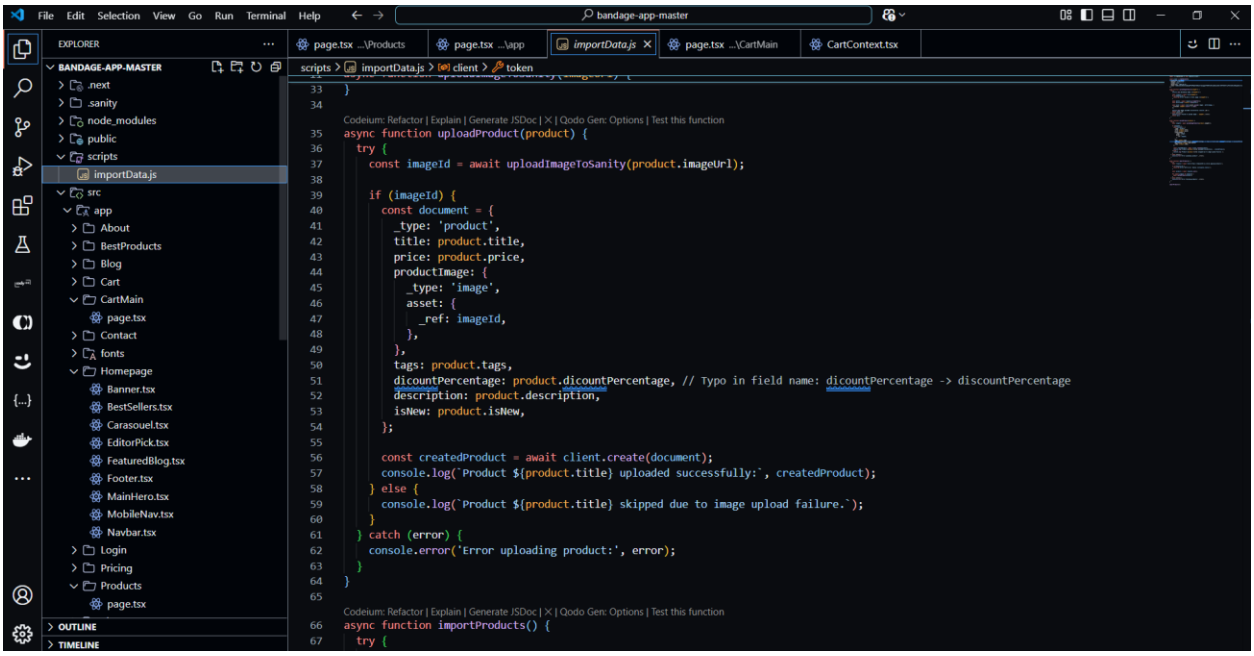
- **Provided API:**
  - Wrote scripts to fetch, transform, and upload data into Sanity CMS.
- **Manual Import:**
  - Exported data as JSON/CSV and imported it using Sanity's import tools.
- **External Platform APIs:**
  - Used Shopify to fetch data, transform fields, and migrate it into the CMS.

4. API Integration in Next.js

- Created **utility functions** to fetch data from the API.
  - Example:

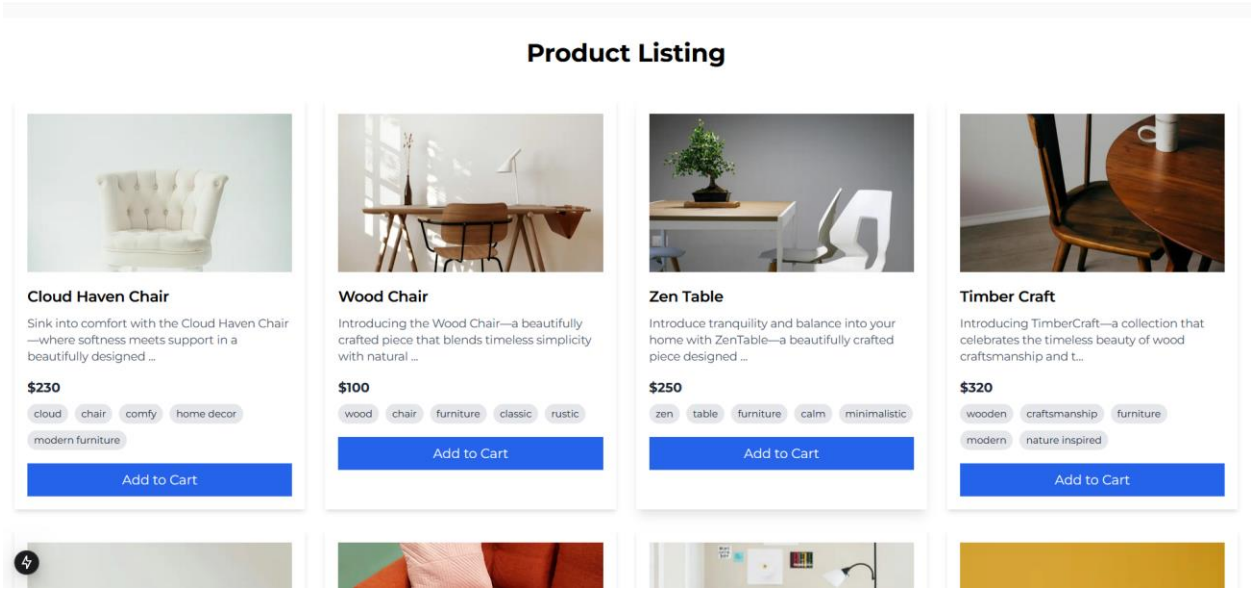
```
export async function fetchProducts() {
  const response = await fetch("https://api.example.com/products");
  return response.ok ? await response.json() : [];
}
```

- Rendered data in components and tested functionality with Postman.
- Implemented **error handling** to log issues and provide fallback options.



Results

1. **Sanity CMS:** Successfully populated with data using automated and manual methods.
2. **Next.js Frontend:** Functional API integration displaying product listings and categories with fallback mechanisms.



Best Practices Followed

- **Sensitive Data Management:** Stored API keys in `.env` files.
- **Clean Code:** Used modular functions and descriptive variables with comments.
- **Data Validation:** Ensured data alignment with the schema before migration.
- **Version Control:** Committed frequently with detailed messages.
- **Thorough Testing:** Addressed edge cases and validated endpoints with Postman.

Submission Checklist

1. **Documentation:**
    a. API integration and schema adjustments.
    b. Migration methods and tools.
2. **Screenshots:**
    a. API responses.
    b. Data displayed in the frontend.
    c. Populated fields in the CMS.
3. **Code Snippets:**
    a. Scripts for data migration and API integration.

Conclusion

Through meticulous planning and execution, I ensured the Sanity CMS was accurately populated and fully integrated into a functional Next.js application. Robust practices and thorough testing were key to achieving a scalable and efficient system.