Day 3 - API Integration and Data Migration Report

Prepared by: Ammara Rajput

Objective

Today, my main goal was to integrate the assigned APIs into a Next.js project and migrate data into Sanity CMS. I focused on ensuring that the APIs were properly connected, the data structures aligned with Sanity schemas, and everything worked seamlessly on the frontend.

Key Outcomes

1. **API Integration:**
   - Integrated the given APIs into the Next.js project.
   - Built reusable utility functions to fetch data and dynamically render it on the frontend.
2. **Data Migration:**
   - Migrated the provided API data into Sanity CMS.
   - Used both automated scripts and manual uploads to ensure accuracy.
3. **Error Handling:**
   - Implemented error-handling mechanisms to catch issues during API calls and migrations.
   - Validated and tested endpoints to ensure everything was working as expected.

Steps I Followed

Step 1: Understanding the API

First, I reviewed the API documentation thoroughly to understand how to interact with the endpoints. The key endpoints I used were:

- /products for product data.
- /categories for category information.

This step helped me figure out what kind of data I'd be working with and how it should be mapped to the Sanity schema.

Step 2: Schema Validation and Adjustments

Next, I compared the API responses with the existing schema in Sanity CMS. There were a few mismatches in field names and structures, so I made adjustments where needed.

For example:

- **API Field:** product_title → **Schema Field:** name

- Updated relationship fields to ensure everything was connected properly.

Step 3: Data Migration into Sanity CMS

For this step, I used a mix of automated scripts and manual methods:

1. **Automated Approach:**
   - I used the migration scripts provided in the template.
   - Modified the scripts slightly to map the fields correctly.
2. **Manual Import:**
   - Exported the API data as JSON files.
   - Imported these files into Sanity using their import tool to validate smaller datasets.

Once the data was imported, I double-checked everything to ensure consistency and accuracy in the CMS.

Step 4: API Integration with Next.js

After completing the migration, I moved on to integrating the APIs with the Next.js project.

1. **Utility Functions:**
   - I wrote reusable utility functions for API calls using both fetch and axios.
2. **Dynamic Rendering:**
   - Used the API data to dynamically display product listings and categories in the components.
3. **Testing:**
   - I tested the API calls using Postman and browser dev tools to ensure everything was working correctly.

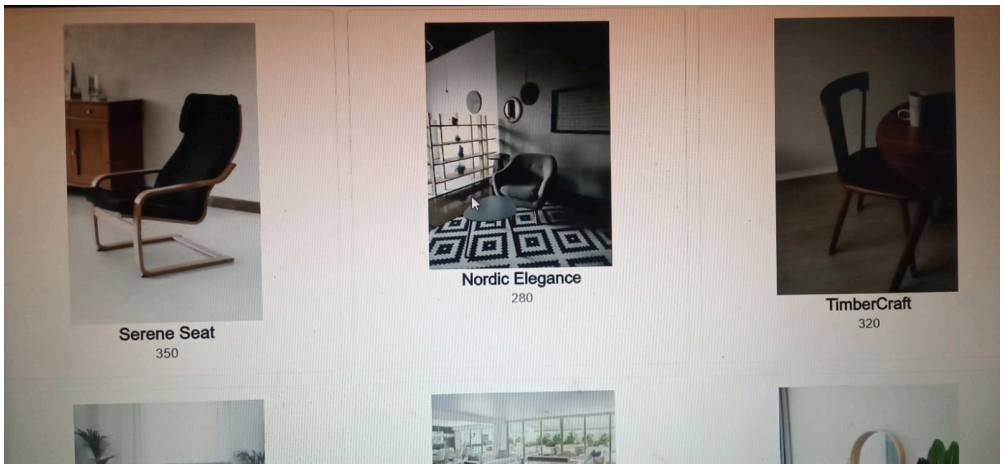
Step 5: Error Handling and Validation

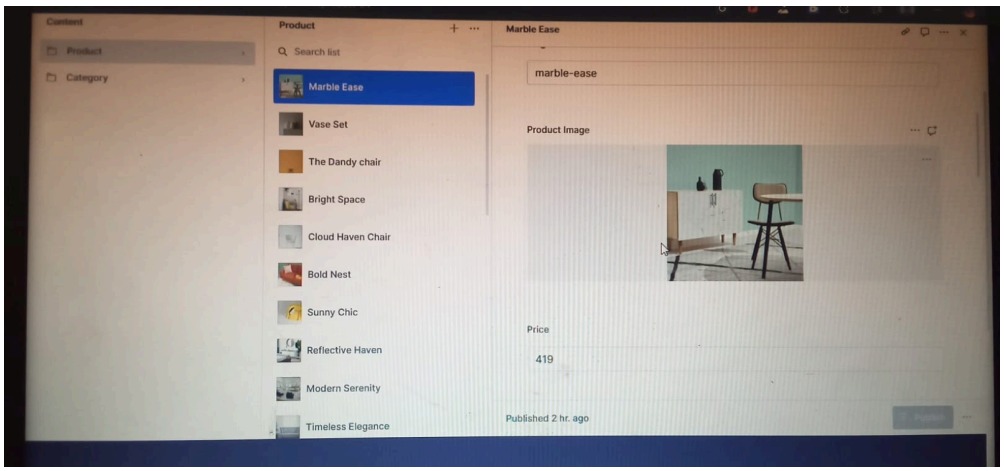
To make the system more robust, I implemented error-handling mechanisms:

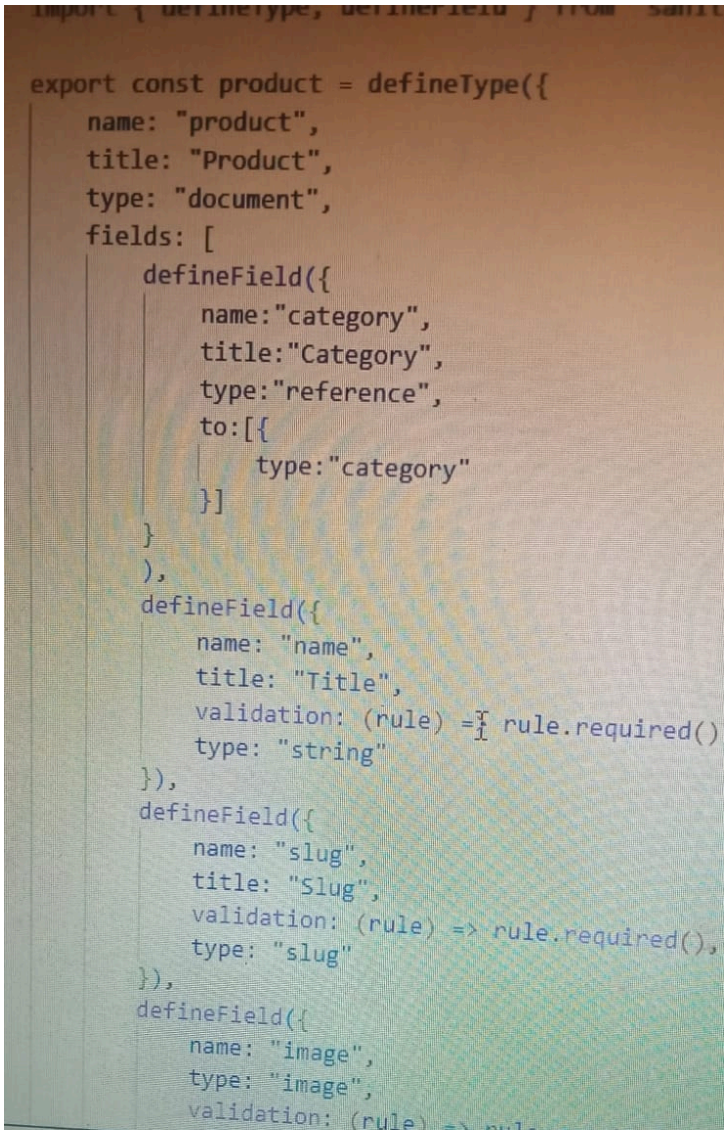
- Logged errors in a centralized file for easy debugging.
- Displayed fallback UI with a user-friendly message in case of API failures.
- Used placeholder data to ensure the frontend didn't break if something went wrong.

Deliverables and Submission

Here's what I've prepared for submission:


Serene Seat
350
Nordic Elegance
280
TimberCraft
320


Content
Product
Category
Product
Search list
Marble Ease
Vase Set
The Dandy chair
Bright Space
Cloud Haven Chair
Bold Nest
Sunny Chic
Reflective Haven
Modern Serenity
Timeless Elegance
Marble Ease
marble-ease
Product Image
Price
419
Published 2 hr. ago



```
export const product = defineType({
    name: "product",
    title: "Product",
    type: "document",
    fields: [
        defineField({
            name:"category",
            title:"Category",
            type:"reference",
            to:[{
                type:"category"
            }]
        }
        ),
        defineField({
            name: "name",
            title: "Title",
            validation: (rule) => rule.required()
            type: "string"
        }),
        defineField({
            name: "slug",
            title: "Slug",
            validation: (rule) => rule.required(),
            type: "slug"
        }),
        defineField({
            name: "image",
            type: "image",
            validation: (rule)
```

Best Practices I Followed

1. **Secure API Keys:**
   - Stored all sensitive information like API keys in .env files.
2. **Clean Code Practices:**
   - Wrote modular utility functions for reusability.
   - Added comments to explain tricky parts of the code.
3. **Data Validation:**
   - Checked field types during migration.
   - Logged issues for later review and debugging.
4. **Version Control:**
   - Made frequent commits with clear, meaningful messages to keep track of changes.
5. **Testing:**
   - Tested API responses using Postman and browser tools.
   - Validated data on both the frontend and the CMS.

Challenges I Faced

- **Schema Mismatches:**
  Some fields in the API didn’t match the Sanity schema. To fix this, I adjusted the field names and relationships to ensure proper mapping.
- **Manual Data Validation:**
  After migrating the data, I manually checked some records in Sanity CMS to make sure everything looked right.
- **Error Handling:**
  Ensuring graceful fallback behavior in case of API failures was a bit challenging, but implementing skeleton loaders and user-friendly messages helped.

What I Achieved Today

- Successfully migrated data into Sanity CMS.
- Integrated APIs with the Next.js project to dynamically display data.
- Documented all steps with relevant screenshots and code snippets.