

Hackathon Day 3: API Integration and Data Migration

On the third day of the hackathon, the primary objective was to integrate APIs and migrate data into Sanity CMS to establish a functional backend for the marketplace. Below is an in-depth account of the steps taken to achieve this goal, along with key learnings and challenges addressed.

Step 1: Understanding the Task and API Documentation

The process began with a thorough review of the API documentation provided. This step was critical to:

1. **Identify Key Endpoints:**
 - APIs related to product listings, categories, and additional metadata were prioritized.
 - Each endpoint's structure, authentication requirements, and data payloads were noted.
2. **Analyze Data Structures:**
 - I mapped the API response fields to the data model required by Sanity CMS.
 - For example, `product_name` in the API corresponded to the `title` field in the product schema, and `price` was directly mapped to the `price` field.
3. **Plan Integration:**
 - I created a checklist of tasks to ensure all necessary API endpoints were accounted for.
 - Detailed notes on rate limits, API keys, and pagination were recorded for reference during the migration process.

Step 2: Validating and Refining Sanity CMS Schemas

Once the API structure was clear, the next step was to validate and adjust the existing schemas in Sanity CMS:

1. **Schema Compatibility Check:**
 - I compared the API's fields and data types with the fields defined in my Sanity CMS schema.
 - For instance, the API used `stock_status` as a string, which I mapped to a Boolean `in_stock` field in the schema.
2. **Schema Adjustments:**
 - Added fields such as `tags` and `discount_price` to accommodate additional API data.
 - Ensured all required fields in the schema were present and correctly aligned.

3. Validation:

- Used Sanity Studio's preview feature to confirm that all schema updates appeared correctly.
- Documented all schema changes for future reference and version control.

Step 3: Data Migration into Sanity CMS

The migration of data from the API into Sanity CMS was a multi-step process that ensured accuracy and security:

1. Environment Setup:

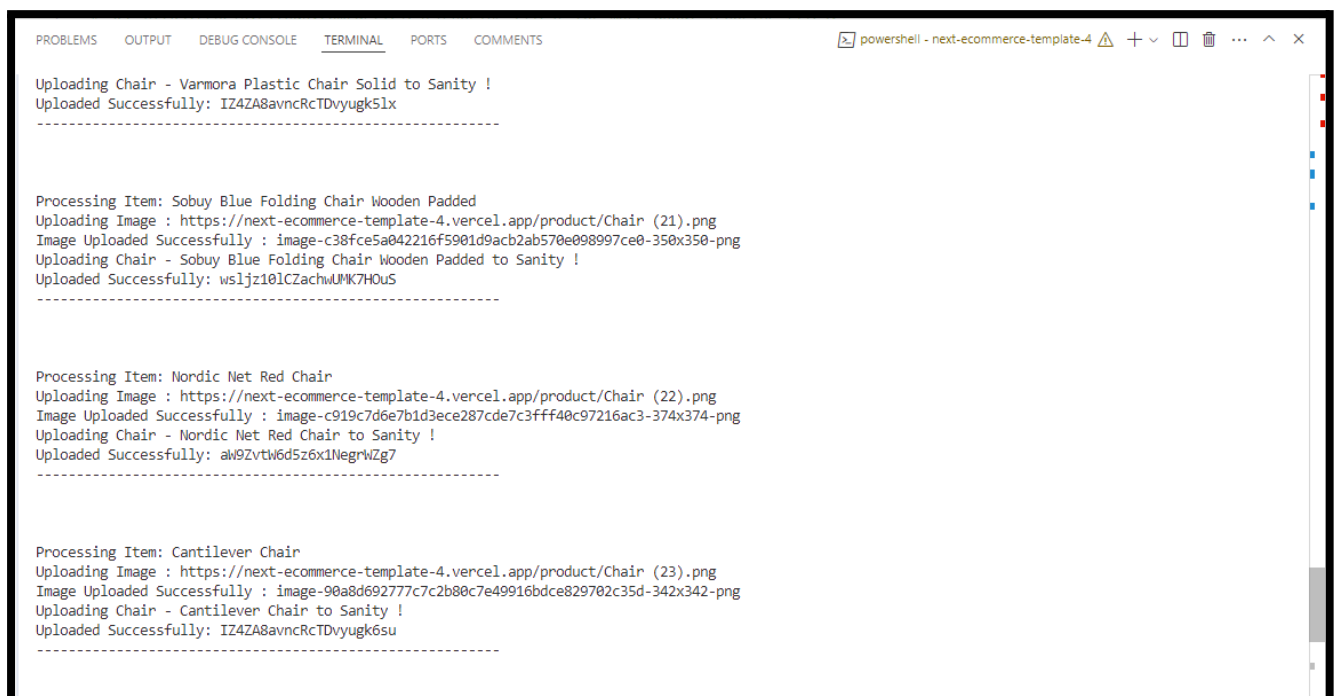
- Created a .env file to securely store the Sanity project ID and token.
- This precaution ensured that sensitive credentials remained secure and were not exposed in the codebase.

2. Script-Based Migration:

- Cloned a repository provided by the faculty, which included a pre-built migration script.
- Customized the script to:
 - Fetch data from the API.
 - Transform the data to match the schema requirements.
 - Handle special cases like null values and nested objects.

3. Data Importation:

- Ran the migration script, which imported API data into Sanity CMS.
- Ensured efficient import by batching requests to avoid exceeding API rate limits.



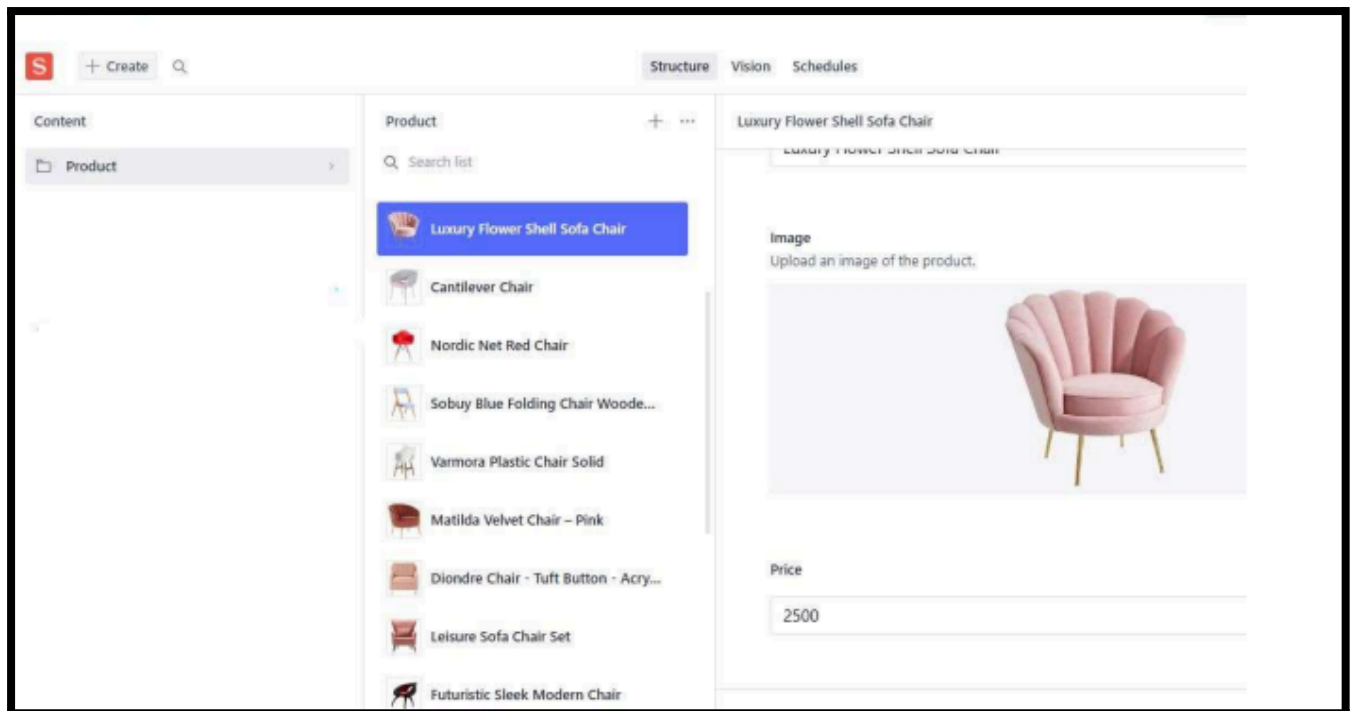
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
powershell - next-ecommerce-template-4

Uploading Chair - Varmora Plastic Chair Solid to Sanity !
Uploaded Successfully: IZ4ZA8avncRcTDvyugk51x
-----

Processing Item: Sobuy Blue Folding Chair Wooden Padded
Uploading Image : https://next-ecommerce-template-4.vercel.app/product/Chair (21).png
Image Uploaded Successfully : image-c38fce5a042216f5901d9acb2ab570e098997ce0-350x350-png
Uploading Chair - Sobuy Blue Folding Chair Wooden Padded to Sanity !
Uploaded Successfully: wsl1jz101CZachwUMK7H0uS
-----

Processing Item: Nordic Net Red Chair
Uploading Image : https://next-ecommerce-template-4.vercel.app/product/Chair (22).png
Image Uploaded Successfully : image-c919c7d6e7b1d3ece287cde7c3fff40c97216ac3-374x374-png
Uploading Chair - Nordic Net Red Chair to Sanity !
Uploaded Successfully: aW9Zvtw6d5z6x1NegrWZg7
-----

Processing Item: Cantilever Chair
Uploading Image : https://next-ecommerce-template-4.vercel.app/product/Chair (23).png
Image Uploaded Successfully : image-90a8d692777c7c2b80c7e49916bdce829702c35d-342x342-png
Uploading Chair - Cantilever Chair to Sanity !
Uploaded Successfully: IZ4ZA8avncRcTDvyugk6su
-----
```



4. Validation:

- Verified the imported data within Sanity Studio to ensure all fields were populated correctly.
- Conducted command-line checks using the Sanity CLI to confirm the integrity of the imported data.

Step 4: API Integration in Next.js

With the data successfully migrated, the focus shifted to integrating the API with the Next.js frontend. This step involved:

1. Creating Utility Functions:

- Developed reusable JavaScript functions to:
 - Fetch data from Sanity CMS using GROQ queries.
 - Make API calls to retrieve dynamic data such as product availability and pricing updates.

2. Frontend Integration:

- Used server-side rendering (SSR) and static site generation (SSG) in Next.js to display the data dynamically.
- Created pages for product listings and categories, which pulled data directly from the backend.

3. Error Handling:

- Implemented robust error-handling mechanisms to manage scenarios like:
 - API failures.
 - Missing data fields.
 - Network timeouts.
- Logged errors to the console for debugging and provided user-friendly error messages on the frontend.

Step 5: Comprehensive Testing and Validation

To ensure the seamless operation of the integrated system, rigorous testing was conducted:

1. API Endpoint Testing:

- Used Postman to test all API endpoints.
- Verified response payloads, status codes, and performance metrics.

2. Data Validation:

- Cross-referenced API data with entries in Sanity CMS to confirm accuracy.
- Checked the frontend to ensure all data was displayed as intended.

3. Functional Testing:

- Tested all pages and components in the Next.js project.
- Ensured features like pagination, filtering, and sorting worked correctly with live data.

4. Stress Testing:

- Simulated high traffic to test the system's ability to handle concurrent API requests.

Final Outcome

The efforts on Day 3 resulted in a significant milestone for the marketplace project:

1. Successful Data Migration:

- All product, category, and metadata from the API were imported into Sanity CMS accurately.

2. Functional API Integration:

- The Next.js frontend displayed dynamic content retrieved from Sanity CMS.

3. Documented Process:

- Detailed documentation of all steps, scripts, and schema adjustments was prepared for future reference.

4. Enhanced System Resilience:

- Robust error handling and validation mechanisms ensured a seamless user experience.

Self-Validation Checklist

Task	Status
API Documentation Review	✓
Schema Validation and Refinement	✓
Data Migration	✓
API Integration in Next.js	✓
Testing and Debugging	✓
Final Documentation	✓

Key Takeaways

This exercise provided invaluable hands-on experience in:

- Designing schemas compatible with external APIs.
- Implementing secure and efficient data migration strategies.
- Integrating a CMS backend with a dynamic frontend using Next.js.
- Handling real-world challenges like error management and system scalability.

The skills developed during this phase have equipped me to tackle more complex backend integration tasks in the future.