

Hackathon Day 3

API Integration and Data Migration Report

Introduction:

On Day 3 of our hackathon, the focus was on integrating APIs and migrating data into Sanity CMS to build a functional marketplace backend. This report outlines the steps taken for API integration and data migration in the marketplace project.

The goal of this task was to:

- Integrate the provided API, validate and adjust the schema, and migrate data to Sanity CMS.
 - Additionally, the frontend was integrated with the APIs to display product listings, categories, and other relevant data.
-

API Integration Process:

1. Understanding the Provided API:

We began by reviewing the API documentation to understand the available endpoints. Key endpoints identified include:

- **Product Listings:** `/products` – This endpoint provides product data, including titles, descriptions, prices, images, discounted prices, discounted percentage, and new product status (`isNew`), etc.

2. API Integration in Next.js:

This section outlines the step-by-step process of integrating the API into the frontend application to fetch and display data dynamically.

1. Backend API Setup:

- Created a Next.js API route (`/api/products`) to query data from Sanity CMS using a Sanity client.
- Returned the fetched product data as a JSON response.

2. Frontend API Call:

- Used React's `useEffect` hook to call the `/api/products` endpoint via `fetch()`.

- Stored the fetched data in the state (products) using `useState()`.

3. State Management and Rendering:

- Managed the loading state (`isLoading`) for a smooth user experience.
- Rendered `ProductCard` components dynamically with the fetched data.

4. Error Handling:

- Implemented error handling in both the backend and frontend to manage API failures and connection issues.

3. Testing the API Integration:

To ensure data consistency, we used tools like **Postman** and browser developer tools to test the API endpoints. We logged the responses and checked for issues related to missing or incorrect data.

Schema Validation and Adjustments

Comparing the API Data with the Existing Sanity Schema:

We reviewed the Sanity CMS schema and compared it with the structure of the data provided by the API.

A key adjustment involved renaming fields to match the existing schema:

- **API Field:** `imageUrl` → **Schema Field:** `productImages`

This adjustment ensured that data could be mapped correctly when imported into Sanity.

Data Migration Script

The core of the data migration process involved creating a script called `importData.js`. This script was responsible for:

- Fetching data from the API.
- Transforming it to match the Sanity schema.
- Importing it into Sanity CMS.

Script Overview

1. Environment Variables Setup:

To securely manage sensitive data such as API tokens, I set up environment variables in my project. This ensures sensitive information is not hardcoded into the script or source code, which is a best practice for security.

- **Setting up the .env file:**

In the root directory of the project, I created a .env file where I stored the Sanity API token and other relevant environment variables.

Example .env file:

```
NEXT_PUBLIC_SANITY_PROJECT_ID  
NEXT_PUBLIC_SANITY_DATASET  
NEXT_PUBLIC_TOKEN_API
```

Migration Process:

The script migrates product data from an external API to Sanity CMS. It fetches product details, uploads images, and stores the products in Sanity, ensuring the data matches the Sanity schema.

Key Functions:

1. **uploadImageToSanity(imageUrl)** – Uploads the product image to Sanity and returns the image asset ID.
 2. **uploadProduct(product)** – Transforms the product data to match the Sanity schema and creates a product document in Sanity with the image reference.
 3. **importProducts()** – Fetches product data from the API, processes each product, and calls `uploadProduct()` to upload the product data to Sanity.
-

Importing Data into Sanity:

- The script used Sanity's JavaScript client (`@sanity/client`) to insert the transformed data into the Sanity CMS.
- The transformed product listings and categories were added to their respective datasets in Sanity.

Process Flow:

1. Fetch data from the external API.

2. For each product, upload the image to Sanity and create a product document with the transformed data.
 3. Ensure all product information, including title, price, description, tags, and image, is stored correctly in Sanity.
-

Post-Migration:

After running the script, I checked the Sanity CMS to ensure that the data had been successfully migrated.

API Integration in Next.js:

In this section, we explain how the API was integrated into the Next.js frontend using API routes.

API Route (`api/products.js`):

- The GET method fetches product data from Sanity using a query that retrieves all documents of type "product".
- The products are returned as a JSON response.
- If an error occurs, a failure response is sent.

Product Page (`ProductPage.tsx`) Code - Key Steps:

1. Fetching Data from the API:

- The `useEffect` hook is used to fetch product data from the backend API when the component loads.
- The `fetch()` function makes a request to the `/api/products` endpoint to retrieve product data.
- The data is fetched as JSON, and if successful, it is stored in the state variable `products` using `setProducts()`.

2. Handling Loading State:

- While data is being fetched, a loading state (`isLoading`) is set to `true` to show a skeleton loader (`<Skeleton />`) until the data is ready.
- Once the data is fetched, `isLoading` is set to `false`, and the actual product data is rendered.

3. Displaying Products:

- The products are displayed in a grid using the ProductCard component.
 - Each product is passed as a prop to the ProductCard component, where its details (such as title, price, and image) are shown.
-

Display Product to UI Self-Validation Checklist

- ✓ **API Understanding**
 - ✓ **Schema Validation**
 - ✓ **Data Migration**
 - ✓ **API Integration in Next.js**
 - ✓ **Submission Preparation**
-

Conclusion:

This document captures the successful completion of **Day 3 - API Integration and Data Migration for Marketplace Furniro**. All required tasks, including API integration, schema adjustments, data migration, and frontend display verification, have been thoroughly executed and documented.

Screenshots, code snippets, and validation processes ensure a comprehensive understanding of the work done.