

# MARKET PLACE E-COMMERCE FURNITURE

18-01-2025

Ayan Sheikh

## DAY-3 API INTEGERATION

### 1. Overview

**API integration** is one of the most crucial aspects of any website, and it's not just about integrating an API. The real challenge lies in **handling the API**, which is often one of the toughest tasks. Interestingly, this is also my **favorite part** of development. API integration is not as simple as just dealing with data; it's about ensuring smooth communication between the backend and frontend, managing data flow, and making sure everything works seamlessly.

Today, I worked on integrating the **API for the furniture website**, focusing on **product** and **category** data. This step is essential to provide dynamic content on the platform and ensure that products and categories are displayed accurately for the users.

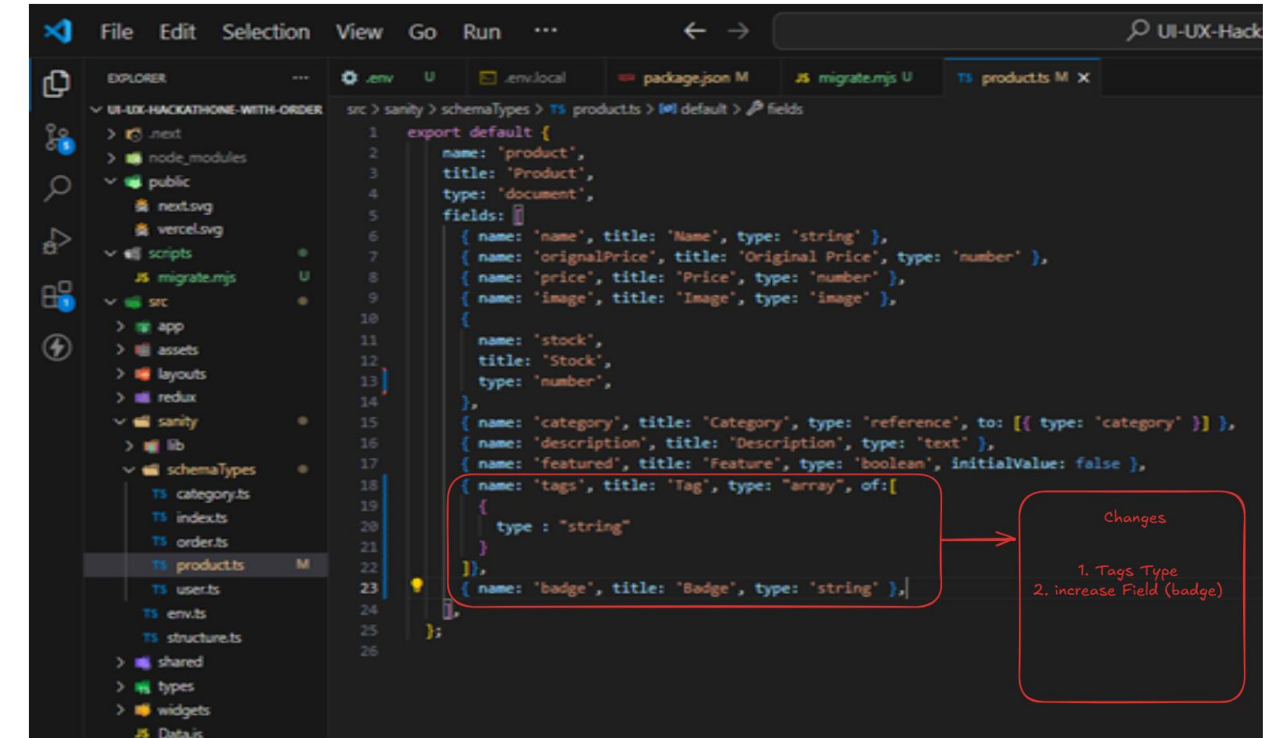
### 2. Api Understanding

- We gained a solid understanding of API functionality and how to integrate it within our system
- Identified key endpoints for managing **product** and **category** data.
- Planned integration points with the frontend to fetch and display product details dynamically.

### 3. Schema Adjustment

#### Define the Schema:

- After making the necessary adjustments based on the provided API, we updated the schema for the furniture products. The schema includes important fields such as **Product Name**, **Price**, **Size**, **Availability**, **Category**, and **Image**.
- These fields were chosen carefully to ensure all critical information about the products is captured



### Check List:

Task Category	Task Description	Status
API Setup	Set up API to fetch furniture product data from the backend and ensure proper endpoints are created.	✓
API Integration	Integrate API with frontend using Axios/Fetch. Ensure correct response and error handling.	✓
Data Migration	Migrate product and category data to the given api and verify data in sanity.	✓
Fetching Data	Set up data fetching using Axios/Fetch API. Handle loading states and errors.	✓
Dynamic Data Display	Map over fetched data to display product info (name, price, category, image, etc.).	✓
Styling & Responsiveness	Style the page using Tailwind CSS. Make the layout responsive.	✓
Error Handling	Display loading spinners and error messages when necessary.	✓

### 4. Migration

- Successfully migrated product and category data into **Sanity CMS**
- Extracted Data from given api and store in Sanity through migration

- Transformed the data to align with the updated schema structure.

```

1 // Import environment variables from .env.local
2 import "dotenv/config";
3
4 // Import the Sanity client to interact with the Sanity backend
5 import { createClient } from "@sanity/client";
6
7 // Load required environment variables
8 const {
9   NEXT_PUBLIC_SANITY_PROJECT_ID, // Sanity project ID
10  NEXT_PUBLIC_SANITY_DATASET, // Sanity dataset (e.g., "production")
11  NEXT_PUBLIC_SANITY_AUTH_TOKEN, // Sanity API token
12  BASE_URL = "https://giaic-hackathon-template-08.vercel.app", // API base URL for products and categories
13 } = process.env;
14
15 // Check if the required environment variables are provided
16 if (!NEXT_PUBLIC_SANITY_PROJECT_ID || !NEXT_PUBLIC_SANITY_AUTH_TOKEN) {
17   console.error("Missing required environment variables. Please check your .env.local file.");
18   process.exit(1); // Stop execution if variables are missing
19 }
20
21 // Create a Sanity client instance to interact with the target Sanity dataset
22 const targetClient = createClient({
23   projectId: NEXT_PUBLIC_SANITY_PROJECT_ID, // Your Sanity project ID
24   dataset: NEXT_PUBLIC_SANITY_DATASET || "production", // Default to "production" if not set
25   useCdn: false, // Disable CDN for real-time updates
26   apiVersion: "2023-01-01", // Sanity API version
27   token: NEXT_PUBLIC_SANITY_AUTH_TOKEN, // API token for authentication
28 });
29
30 // Function to upload an image to Sanity
31 async function uploadImageToSanity(imageUrl) {
32   try {
33     // Fetch the image from the provided URL
34     const response = await fetch(imageUrl);
35     if (!response.ok) throw new Error("Failed to fetch image: ${imageUrl}");
36
37     // Convert the image to a buffer (binary format)
38     const buffer = await response.arrayBuffer();
39
40     // Upload the image to Sanity and get its asset ID
41     const uploadedAsset = await targetClient.assets.upload("image", Buffer.from(buffer), {
42       filename: imageUrl.split("/").pop(), // Use the file name from the URL
43     });
44
45     return uploadedAsset.id; // Return the asset ID
46   } catch (error) {
47     console.error("Error uploading image:", error.message);
48     return null; // Return null if the upload fails
49   }
50 }
51
52 // Main function to migrate data from REST API to Sanity
53 async function migrateData() {

```

## 5. Fetch Data in Next js

- Implemented API calls to fetch **furniture product** and **category** data dynamically.
- Set up proper state management to store the fetched data and enhance the user experience.

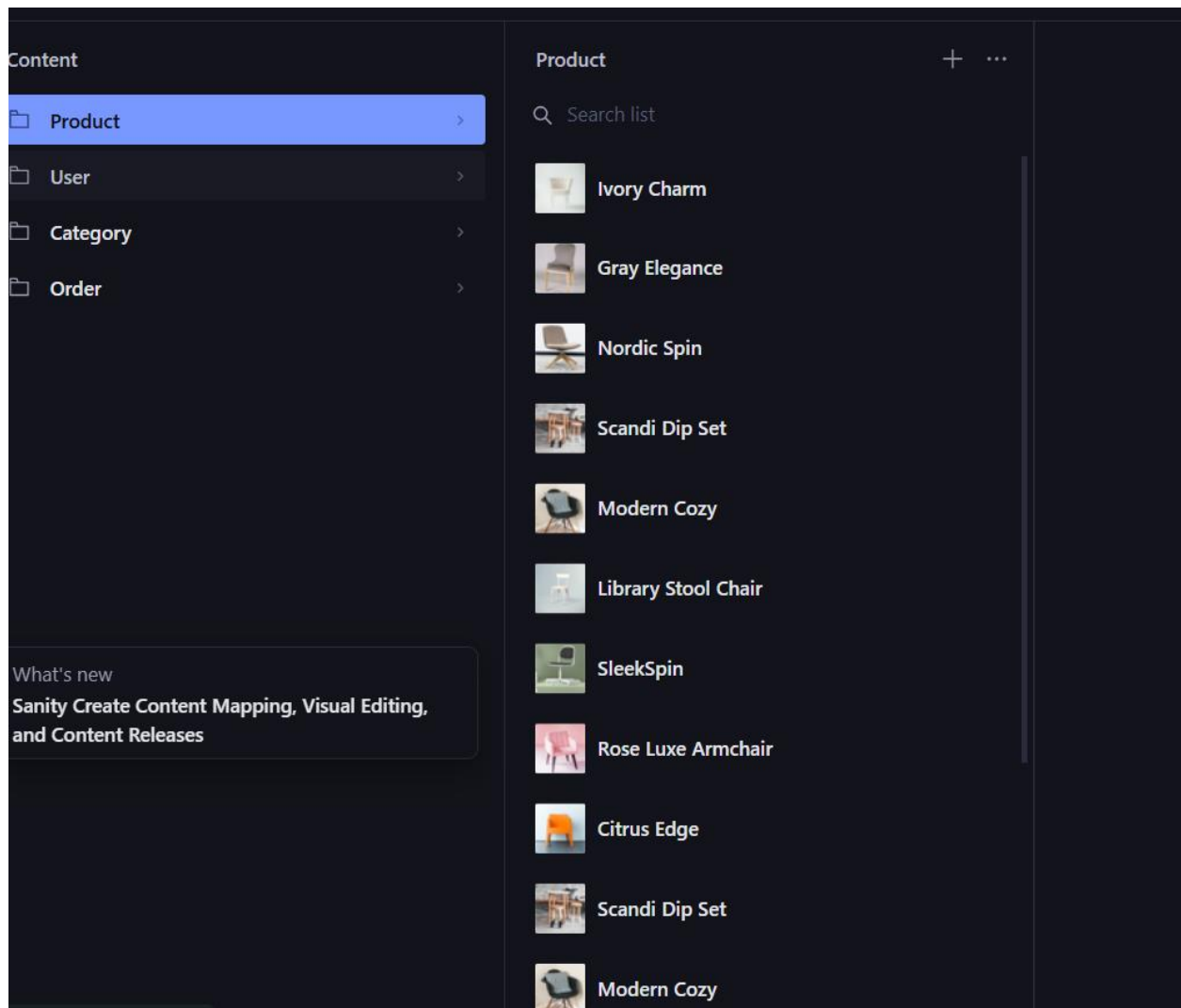
- Conducted tests to ensure the API integration is efficient and data is accurately displayed

## Code Screenshot:

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'app', 'api', 'products', and 'route.ts'. The 'api' directory is expanded, showing 'products' and 'route.ts'. The 'route.ts' file is selected, and its content is displayed in the code editor. The code is a TypeScript route handler for a GET request, using Next.js and Sanity. It fetches products from Sanity and returns a JSON response. The code is as follows:

```
src > app > api > products > TS route.ts > GET
1  import { NextResponse } from 'next/server';
2  import { client } from '@sanity/lib/client';
3
4
5
6  export async function GET(req:any) {
7    try {
8
9
10     // Fetch products from Sanity
11     const products = await client.fetch(
12       `*[_type == "product"]{
13         _id,
14         name,
15         price,
16         "category": category->name,
17         stock,
18         description,
19         featured,
20         originalPrice,
21         "imageUrl": image.asset->url
22       }`
23     );
24
25     // Return the response
26     return NextResponse.json(
27       { success: true, data: products },
28       { status: 200 }
29     );
30   } catch (error:any) {
31     // Handle errors
32     return NextResponse.json(
33       { success: false, message: error.message },
34       { status: 401 }
35     );
36   }
37 }
38
```

## 6. Verify Data in Sanity



## 7. Show On Frontend