Day 3 - API Integration Report – Furnishful

API Integration Process

I created a custom backend where I developed my own APIs. Then I used these APIs to migrate data to Sanity, a headless CMS, and finally displayed the data on the frontend.

Steps

Custom Backend Creation

- Built a custom backend where I defined various APIs.
- Developed these APIs to handle different types of product data, implementing CRUD (Create, Read, Update, Delete) operations.

Data Migration to Sanity

- Fetched data from the custom backend using these APIs.
- Migrated the data to Sanity, a headless CMS platform.
- Properly formatted the data to align with the Sanity schema during migration.

API Integration in Frontend

- Integrated these APIs into the frontend to fetch and display data.
- Displayed all products on the frontend using the fetched data.
- Displayed individual products on separate pages using their IDs.
- Displayed a limited number of products on specific pages based on certain criteria.

Example Use Cases

1. **Displaying All Products**: Displayed all products on the frontend.
2. **Displaying Individual Products**: Displayed details of a specific product on a separate page using the product ID.
3. **Displaying Limited Products**: Displayed specific products on the frontend based on defined criteria.

In this way, I completed the API integration process and displayed the data on the frontend.

Schema Adjustments

During the API integration process, several adjustments were made to the provided schema to better fit the requirements of the project. Here are the key changes:

Field Types

- Changed `id` field type from string to number for better numeric handling.

Field Names

- Renamed `imagePath` to `image` for simplicity and consistency.

New Fields Added

- `tags`: Added to store an array of product tags.
- `sizes`: Added to store product sizes as a number.
- `rating`: Added to store product ratings, with an appropriate description.
- `stockQuantity`: Added to store stock quantity, replacing `stockLevel`.

Renamed Fields

- Renamed `name` to `Product Name` for better clarity.
- Changed `description` field type from text to string.

These adjustments ensured that the schema met the specific needs of the application and allowed for seamless integration with the custom backend and Sanity CMS.

Migration Steps and Tools Used

Tools Used

- **Sanity Client**: Helps you interact with Sanity CMS to upload your product data securely.
- **dotenv**: Loads environment variables from a file, so you can keep sensitive information like API keys safe and separate from your code.
- **Node.js Fetch API**: Used to fetch product data from your custom backend.
- **Custom Scripts**: Automate fetching, processing, and uploading your product data to Sanity CMS.

Migration Steps

1. **Load Environment Variables**

    - Use dotenv to load all necessary environment variables, such as API keys and project IDs. This keeps your sensitive data secure and out of your main code.
2. **Set Up Sanity Client**
    - Configure the Sanity Client with your project details, so you can securely connect to Sanity CMS and upload your data.
3. **Fetch Product Data from Custom Backend**
    - Use the Fetch API to make requests to your custom backend and get the product data.
4. **Process and Format Data**
    - Take the product data you fetched and format it to match the schema you have defined in Sanity CMS.
5. **Upload Data to Sanity**
    - Use the Sanity Client to upload the formatted product data to Sanity CMS. Each product becomes a new document in your Sanity database.
6. **Handle Errors**
    - Catch and log any errors that happen during the fetching, processing, or uploading steps. This helps you debug issues and ensure data integrity.

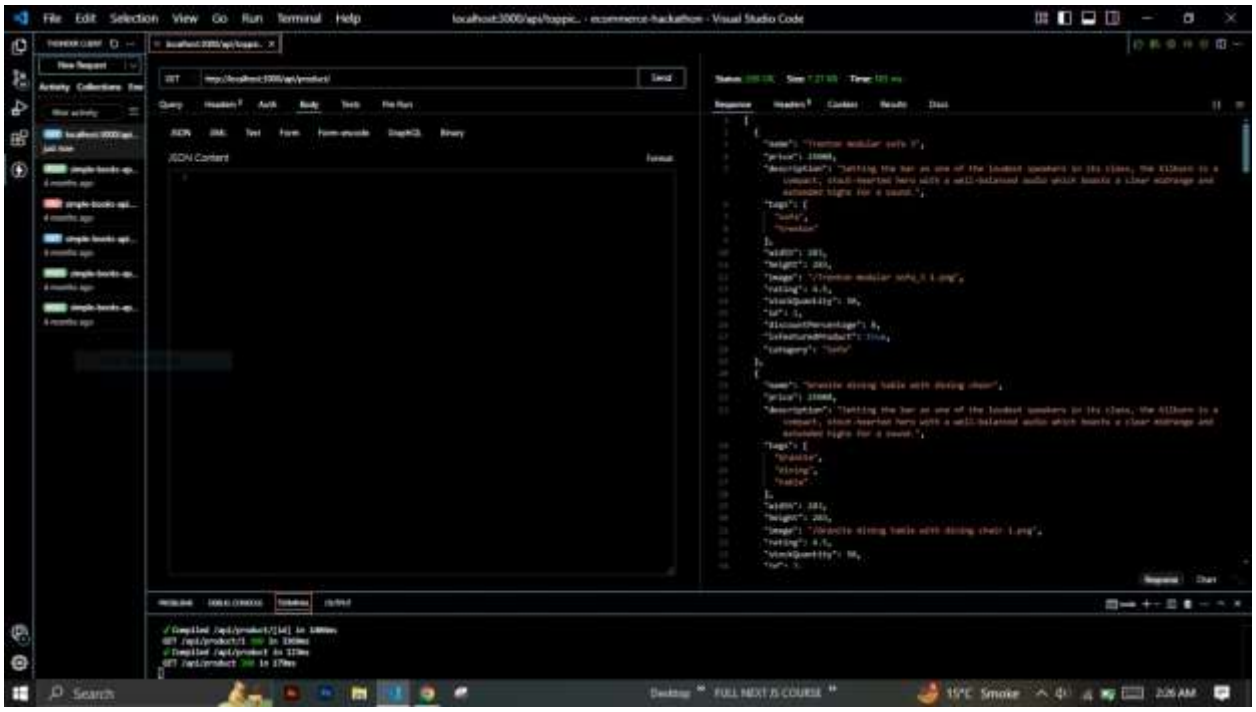
In Simple Steps

1. **Securely Load Your Keys and IDs**
    - Use dotenv to load environment variables so you don't expose sensitive data in your code.
2. **Connect to Sanity**
    - Set up the Sanity Client with the right project information to interact with Sanity CMS.
3. **Get Your Data**
    - Fetch product data from your backend using the Fetch API.
4. **Prepare the Data**
    - Process and format the data according to your Sanity schema.
5. **Upload the Data**
    - Use the Sanity Client to upload the data to Sanity CMS.
6. **Fix Problems**
    - Implement error handling to log and resolve any issues that come up.

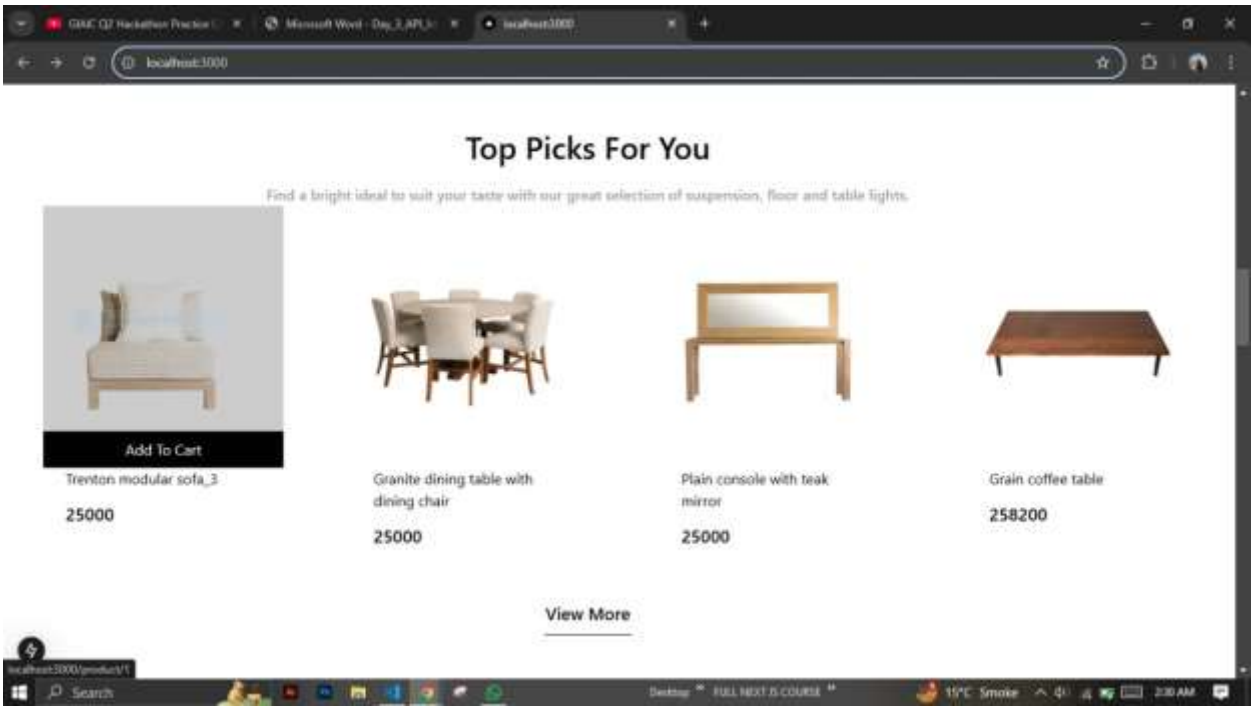
Section: API Integration Success

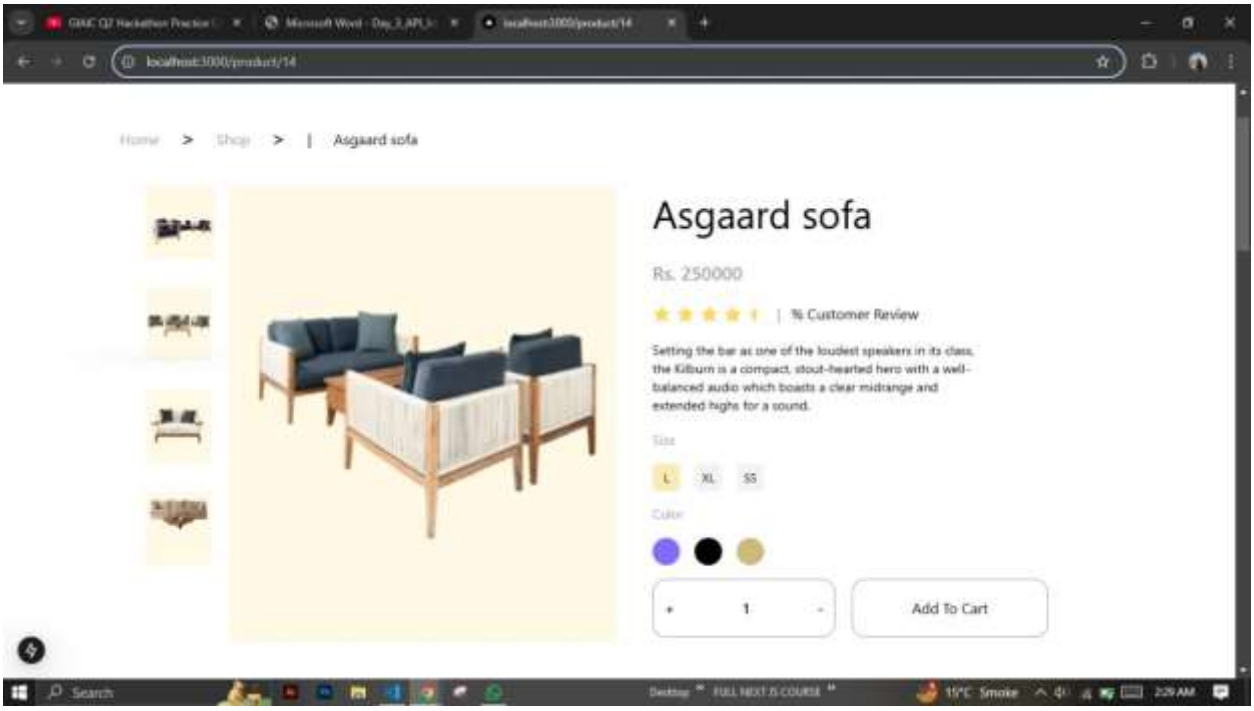
1. **API Call Screenshot**
    - **Description**: This screenshot shows our custom backend API call that fetches product data.

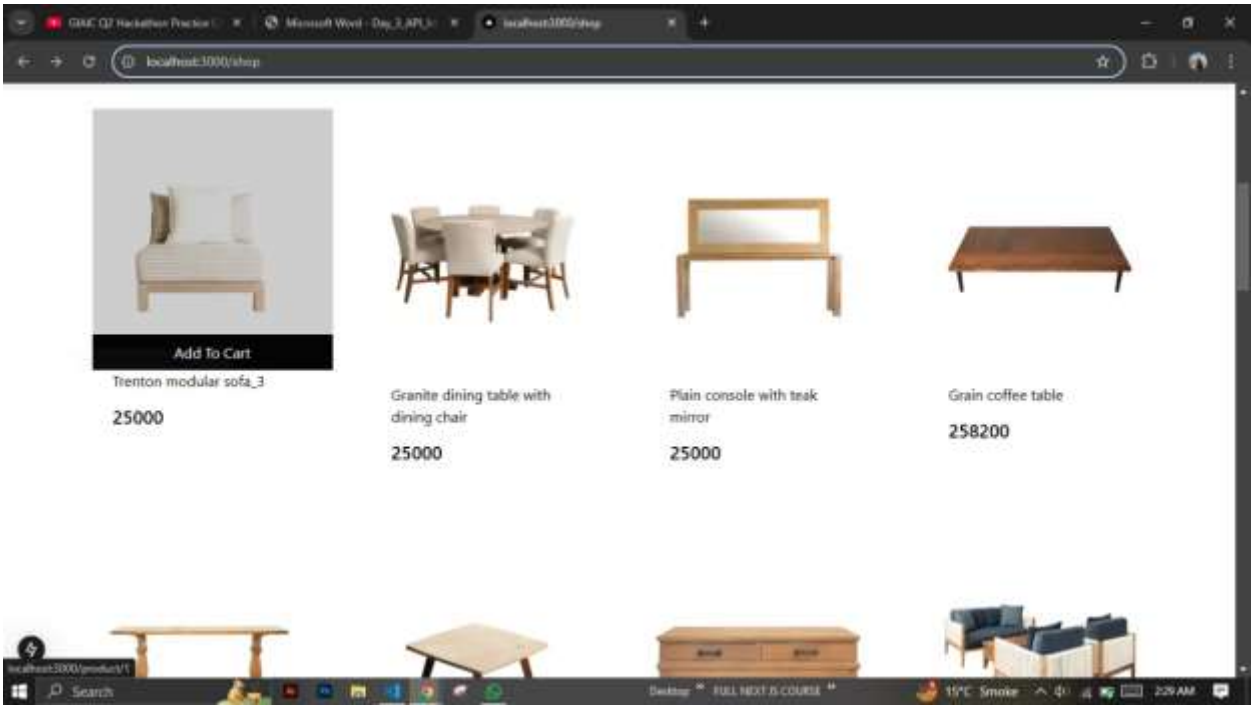
2. **Image**:



3. **Frontend Display Screenshot**
    - **Description**: These screenshot demonstrates how the fetched product data is displayed on our frontend.
    - **Image**:

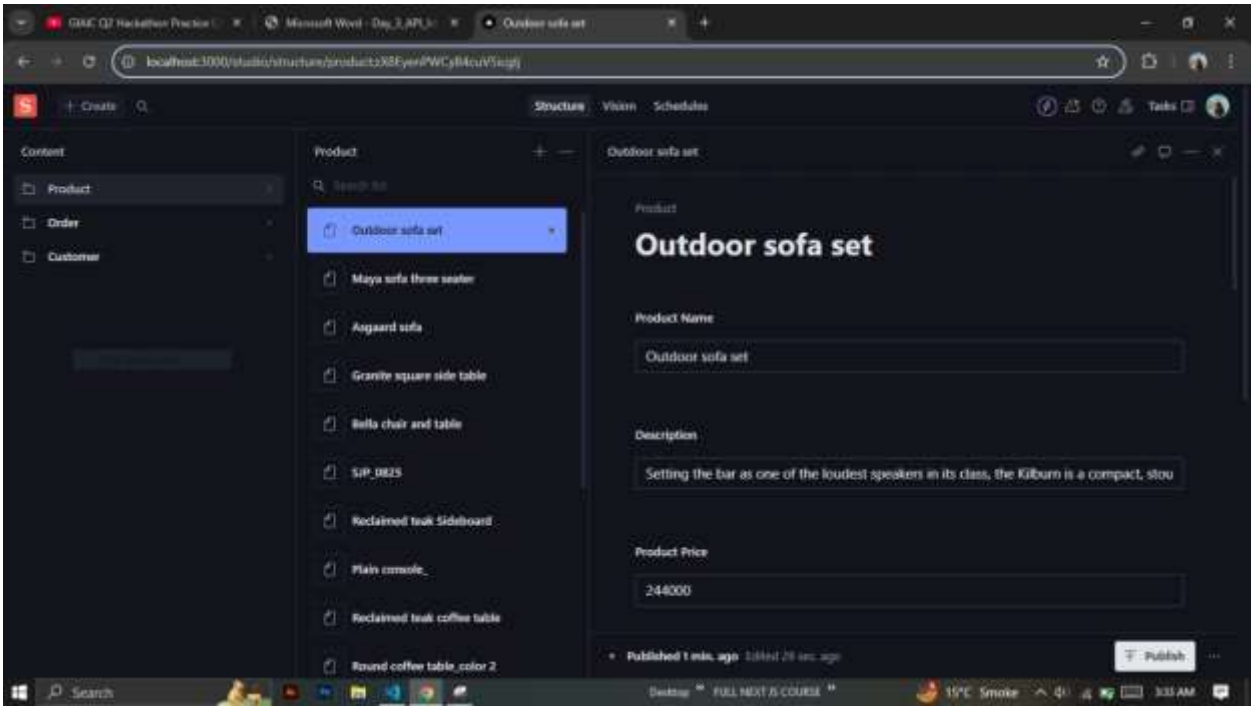






4. **Sanity CMS Field Screenshot**
    - **Description**: This screenshot shows the populated fields in Sanity CMS with data imported from our custom backend.

- **Image**:



Code Snippets

1. **Sanity Data Migration Code Snippet:**

```
import { createClient } from '@sanity/client';
import { fileURLToPath } from 'url'
import path from 'path';
import dotenv from 'dotenv';

// Load environment variables from .env.local
const __filename = fileURLToPath(import.meta.url)
const _dirname = path.dirname(_filename)
dotenv.config({ path: path.resolve(__dirname, '../.env.local') })
// Create Sanity client
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: false,
  token: process.env.NEXT_PUBLIC_SANITY_API_TOKEN,
  apiVersion: '2025-01-14'
});

async function importData() {
  try {
    console.log('Fetching products from API...')
```

```
    const response = await fetch('http://localhost:3000/api/product');
    const products = await response.json();
    for (const product of products) {
      console.log(Processing product: ${product.name})
      const sanityProduct = {
        _type: 'product',
        name: product.name,
        description: product.description,
        price: product.price,
        tags: product.tags,
        width: product.width,
        height: product.height,
        image: product.image,
        rating: product.rating,
        stockQuantity: product.stockQuantity,
        id: product.id,
        discountPercentage: product.discountPercentage,
        isFeaturedProduct: product.isFeaturedProduct,
        category: product.category,
      }
      console.log('Uploading product to Sanity:', sanityProduct.name)
      const result = await client.create(sanityProduct)
      console.log(Product uploaded successfully: ${result._id})
    }
    console.log('Data import completed successfully!')
  } catch (error) {
    console.error('Error importing data:', error)
  }
}
importData()
```

2. **Utility Functions Code Snippet:**

```
import { client } from "@/sanity/lib/client";

export async function getData_Toppick_and_Related() {
  const fetchData = await client.fetch(*[_type == "product"][0..3]);
  return fetchData;
}

export async function getData_Asgard_Sofa(productId: number) {
    const fetchData = await client.fetch(*[_type == "product" && id ==
${productId}])
    return fetchData;
}

export async function getAllData() {
    const fetchData = await
client.fetch(${process.env.NEXT_PUBLIC_FETCHING_URL})
```

```
    return fetchData;
}
```