Day 3 - API Integration Report - TIMELESS TIMBER

Introduction:

This report documents the integration of the external API (`https://template-0-beta.vercel.app/api/product`) with our application and the adjustments made to the Sanity schema to support the data structure and requirements. The goal was to fetch product data, transform it to fit our schema, and store it in the CMS for further use.

Api Integration Process:

Template-0 api which was provided in the day-3 pdf is used in this project.

API Overview

- **API Endpoint**: api/product
- **Authentication**: None
- **Response Format**: JSON
- **Key Data Points**:
  - `id`: Unique identifier
  - `name`: Product name
  - `price`: Price as a number
  - `description`: Product description
  - `category`: Category of the product

ADJUSTED SCHEMA ACCORDING TO THE PROVIDED API:

Inserted data from the api into the schema fields, added more fields according to the api and added data in the remaining schema fields manually in sanity studio.

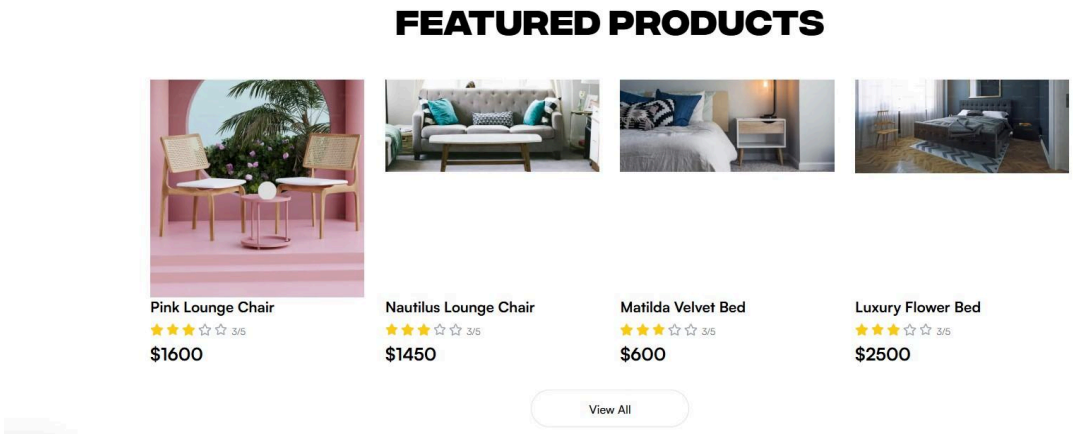
MIGRATION STEPS AND TOOLS USED:

- Used Postman for testing APIs and reviewing JSON.
- Researched about migration, found 2 methods: first api route and second script code.
- Used script method with ETL **(Extract, Transform, Load)** pattern in this Marketplace project.
- Stored Sanity token and projectId in a .env file.
- Used fetch method to get the data from the external api.
- Convert the data into JSON with data.json()
- Used sanity client.create method with a for-of loop to insert the JSON data into my Sanity Database.

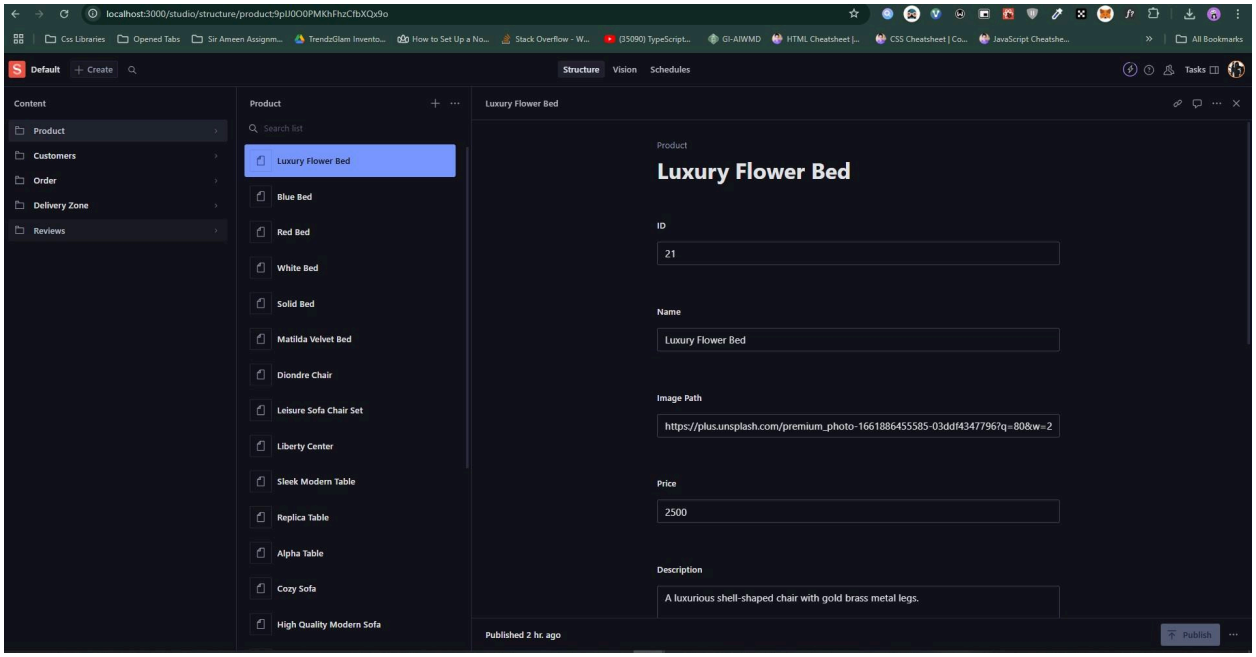
SANITY API CALL:

```
const getFeaturedProducts = async () => {
  const featuredProducts = await client.fetch(`
      *[_type == "product" && isFeaturedProduct][0..3] {
  _id,
  id,
    name,
    imagePath,
    price,
    description,
    category,
    stockLevel,
    rating,
    isFeaturedProduct,
    discountPercentage,
}`);
  return featuredProducts;
};
```

DATA DISPLAYED ON THE FRONTEND:



SANITY FIELDS:



CODE SNIPPET OF MIGRATION SCRIPT:

SANITY CLIENT:

```
import { createClient } from '@sanity/client';
import dotenv from "dotenv"

dotenv.config();
const client = createClient({
    projectId: process.env.projectid,
    dataset: process.env.dataset,
    apiVersion: "2025-01-18",
    useCdn: false,
    token: process.env.token,
});

export default client;
```

MIGRATION:

```
import client from './sanityClient.js';

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }
    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);
    const asset = await client.assets.upload('image', bufferImage, {
      filename: imageUrl.split('/').pop(),
    });
    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
```

```
    return null;
  }}

async function uploadProduct(product) {
  try {
    const imageId = await uploadImageToSanity(product.imagePath);
    if (imageId) {
      const document = {
        _type: 'product',
        id: product.id,
        name: product.name,
        imagePath: product.imagePath,
        price: parseFloat(product.price),
        description: product.description,
        discountPercentage: product.discountPercentage,
        isFeaturedProduct: product.isFeaturedProduct,
        stockLevel: product.stockLevel,
        category: product.category,
      };


      const createdProduct = await client.create(document);
      console.log(`Product ${product.title} uploaded successfully:`,
createdProduct);
    } else {
      console.log(`Product ${product.title} skipped due to image upload
failure.`);
    }
  } catch (error) {
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {
  try {
    const response = await
fetch('https://template-0-beta.vercel.app/api/product');

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
```

```
    }

    const products = await response.json();

    for (const product of products) {
      await uploadProduct(product);
    }
  } catch (error) {
    console.error('Error fetching products:', error);
  }
}

importProducts();
```