Hackathon Day 03 Task

DAY 3 - API INTEGRATION AND DATA MIGRATION

On Day 3 of the hackathon, I focused on integrating APIs and transferring data into Sanity CMS to build the backend for a functional marketplace. I learned how to:

1. Connect APIs to my Next.js project.

2. Move data from APIs into Sanity CMS.

3. Use data from platforms like Shopify, Magento, WooCommerce, WordPress, Salesforce, custom backends, or mock APIs.

4. Work with and validate schemas to ensure they match the data sources.

Through this, I understood real-world practices, like using headless APIs and migrating data from popular eCommerce platforms, which will help me handle client needs in the future.

API Integration and Data Migration (Template 2)

1. API Integration:

For the hackathon, I worked on Template 2, guided by Sir Bilal Muhammad Khan and Sir Aneeq. Here's how I approached the task:

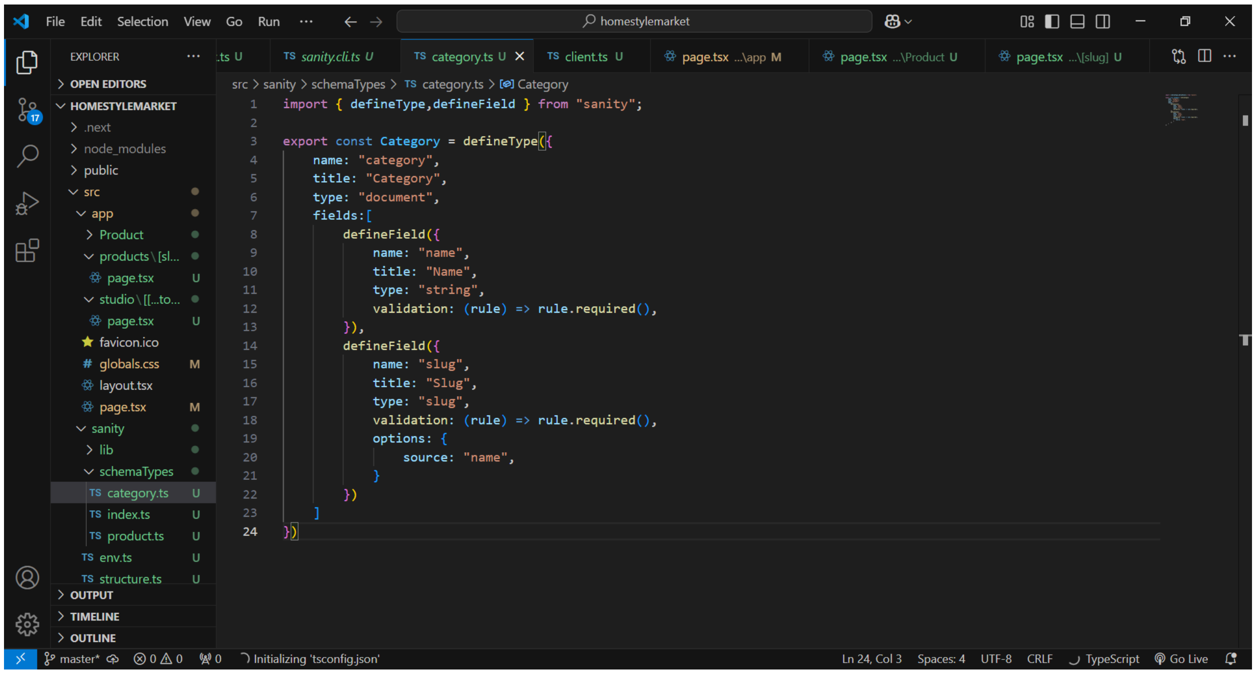
I used the provided API:
https://hackathon-apis.vercel.app/api/products
to fetch data.
This API provided product details like title, price, slug, description, and category, which were necessary for building my marketplace.

2. Sanity CMS Schemas Configuration:

I referred to the following schema definitions provided in the GitHub repository to structure my Sanity CMS:

Category Schema:

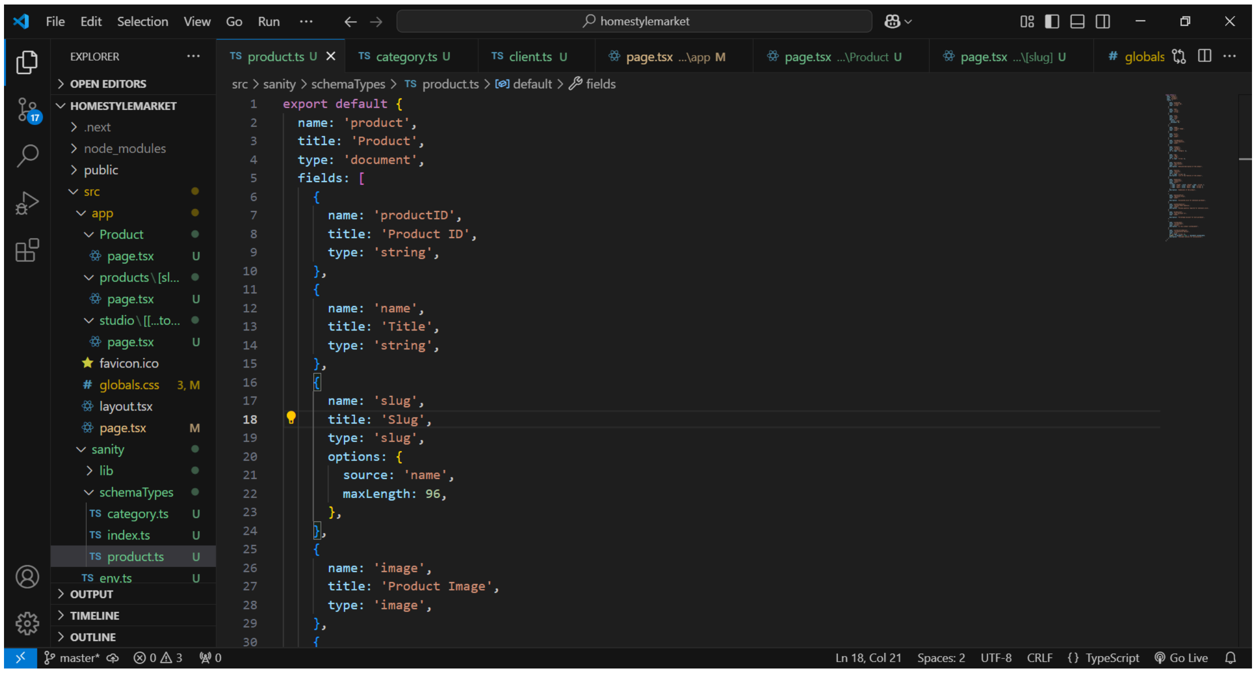
category.ts



```
import { defineType,defineField } from "sanity";

export const Category = defineType({
    name: "category",
    title: "Category",
    type: "document",
    fields:[
        defineField({
            name: "name",
            title: "Name",
            type: "string",
            validation: (rule) => rule.required(),
        }),
        defineField({
            name: "slug",
            title: "Slug",
            type: "slug",
            validation: (rule) => rule.required(),
            options: {
                source: "name",
            }
        })
    ]
})
```

Product Schema:

product.ts

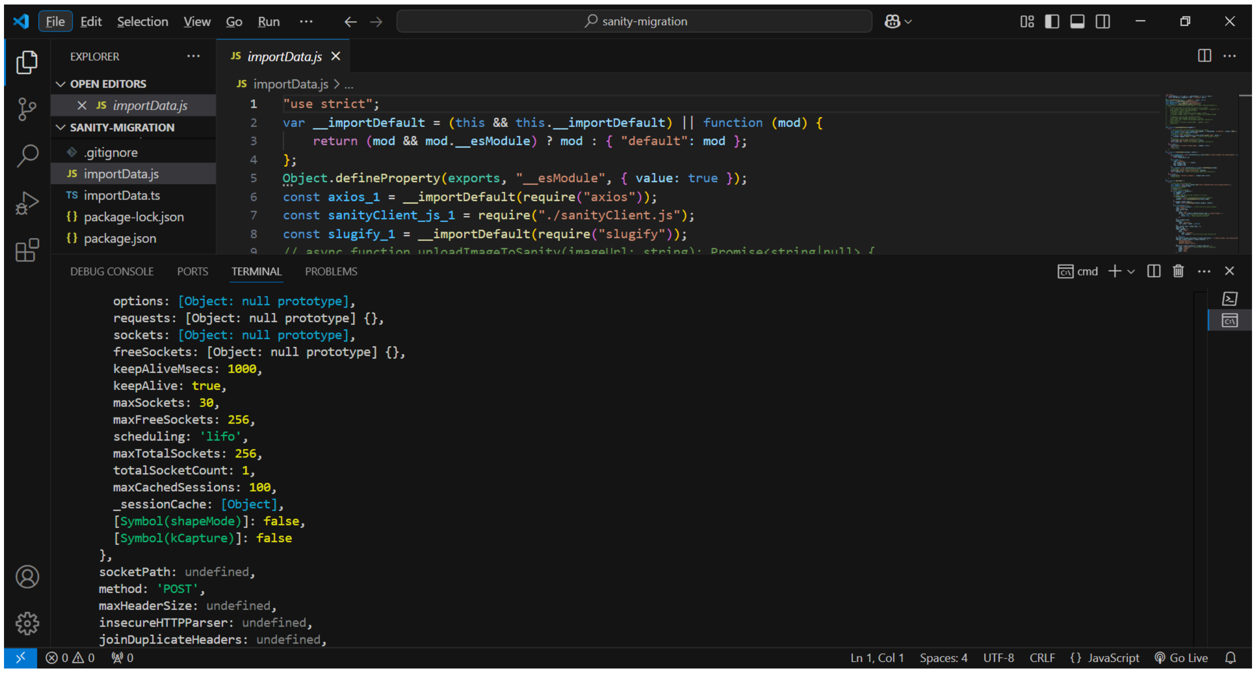

```
export default {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'productID',
      title: 'Product ID',
      type: 'string',
    },
    {
      name: 'name',
      title: 'Title',
      type: 'string',
    },
    {
      name: 'slug',
      title: 'Slug',
      type: 'slug',
      options: {
        source: 'name',
        maxLength: 96,
      },
    },
    {
      name: 'image',
      title: 'Product Image',
      type: 'image',
    },
    {
```

These schemas ensured that the data from the API could be accurately mapped and stored in Sanity CMS.

3. Data Migration:

Using the migration script, I successfully transferred data from the API into Sanity CMS.

The script mapped API fields (e.g., title, price, slug, description) to the corresponding Sanity schema fields and used references for category relationships.



```javascript
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
const axios_1 = __importDefault(require("axios"));
const sanityClient_js_1 = require("./sanityClient.js");
const slugify_1 = __importDefault(require("slugify"));
// async function uploadImageToSanity(imageUrl: string): Promise<string|null> {
```

```
      options: [Object: null prototype],
      requests: [Object: null prototype] {},
      sockets: [Object: null prototype],
      freeSockets: [Object: null prototype] {},
      keepAliveMsecs: 1000,
      keepAlive: true,
      maxSockets: 30,
      maxFreeSockets: 256,
      scheduling: 'lifo',
      maxTotalSockets: 256,
      totalSocketCount: 1,
      maxCachedSessions: 100,
      _sessionCache: [Object],
      [Symbol(shapeMode)]: false,
      [Symbol(kCapture)]: false
    },
    socketPath: undefined,
    method: 'POST',
    maxHeaderSize: undefined,
    insecureHTTPParser: undefined,
    joinDuplicateHeaders: undefined,
```

I cloned the repository and set up the environment.

After understanding the schema structure, I ran the migration script to fetch product data from the API and populate my Sanity CMS instance.

I validated the migrated data to ensure it was consistent with the schema definitions.

Outcome:

I successfully migrated the API data into Sanity CMS and ensured it aligned with the defined schemas.

This allowed me to build a functional backend for my marketplace while gaining hands-on experience with real-world practices like API integration, data migration, and schema validation.

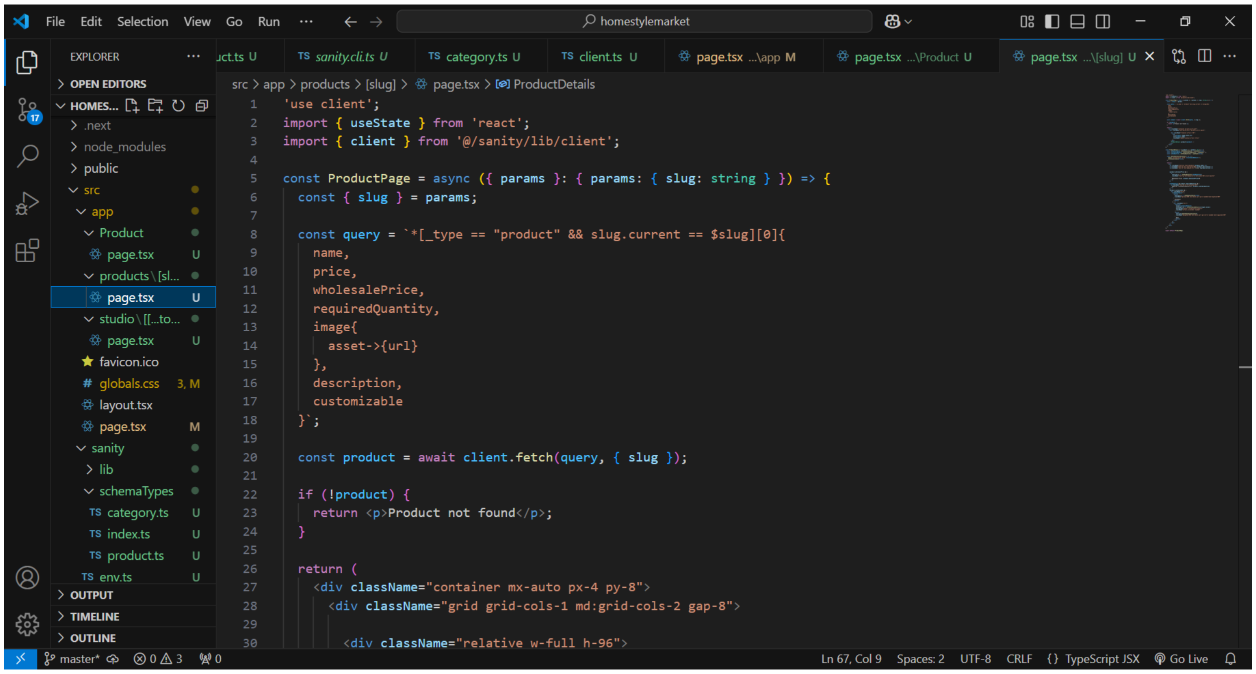
This process gave me a clear understanding of handling data sources, customizing APIs, and ensuring compatibility with headless CMS platforms like Sanity.

4. Frontend Implementation

Frontend Steps:

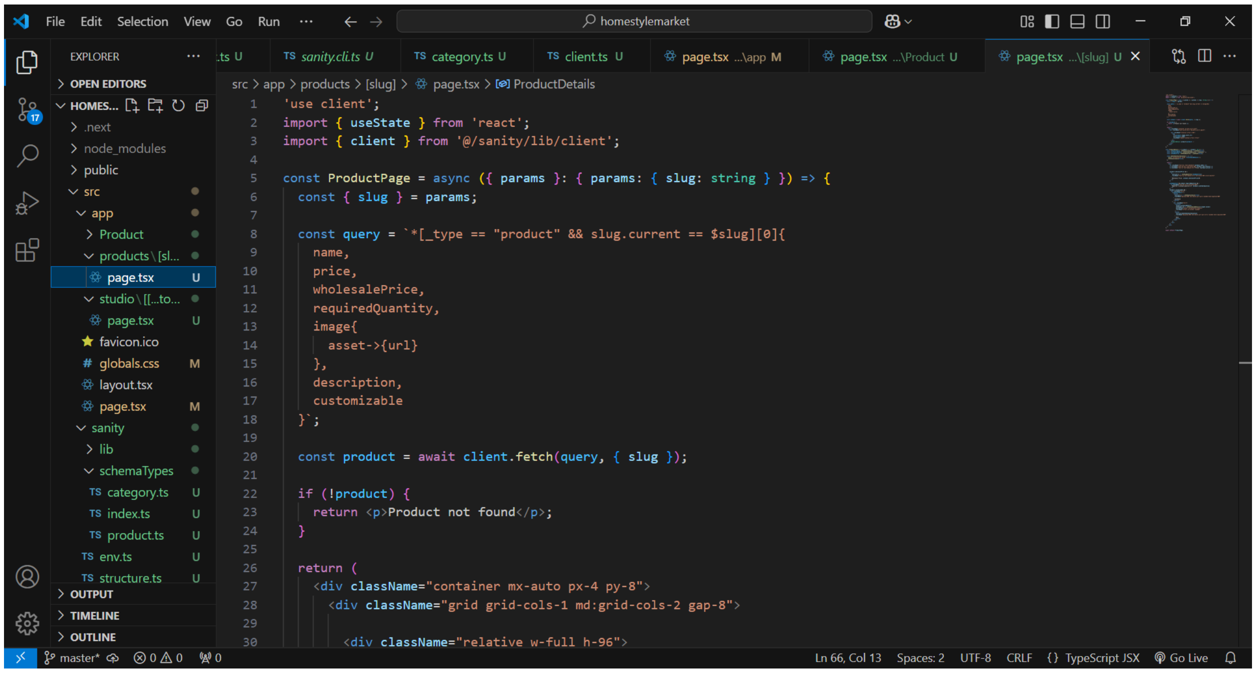
1. Fetching Products Data:

I used Sanity’s GROQ queries in my Next.js project to fetch the data directly from the CMS.


```
'use client';
import { useState } from 'react';
import { client } from '@/sanity/lib/client';

const ProductPage = async ({ params }: { params: { slug: string } }) => {
  const { slug } = params;

  const query = `*[_type == "product" && slug.current == $slug][0]{
    name,
    price,
    wholesalePrice,
    requiredQuantity,
    image{
      asset->{url}
    },
    description,
    customizable
  }`;

  const product = await client.fetch(query, { slug });

  if (!product) {
    return <p>Product not found</p>;
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-8">

        <div className="relative w-full h-96">
```

2. Dynamic Product Pages:

I created a dynamic route ([slug].tsx) to display detailed product information using Sanity's GROQ query based on the slug

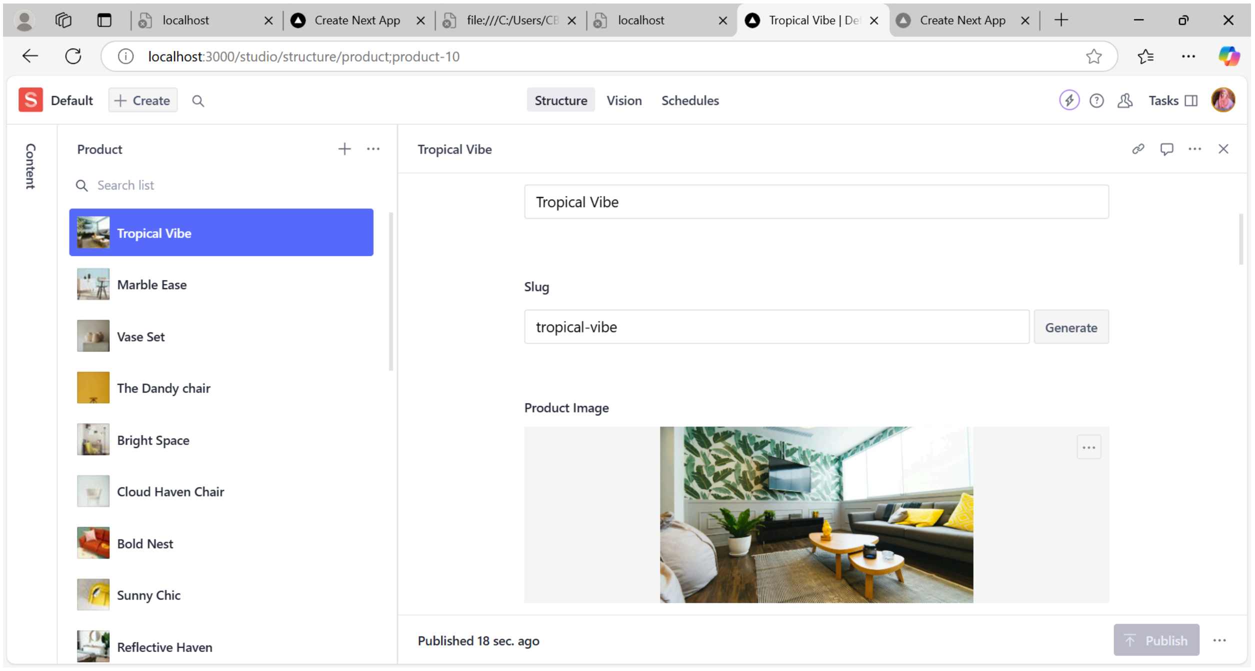


```tsx
'use client';
import { useState } from 'react';
import { client } from '@/sanity/lib/client';

const ProductPage = async ({ params }: { params: { slug: string } }) => {
  const { slug } = params;

  const query = `*[_type == "product" && slug.current == $slug][0]{
    name,
    price,
    wholesalePrice,
    requiredQuantity,
    image{
      asset->{url}
    },
    description,
    customizable
  }`;

  const product = await client.fetch(query, { slug });

  if (!product) {
    return <p>Product not found</p>;
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-8">

        <div className="relative w-full h-96">
```

5. Sanity Studio View After Migration

After running the migration script, the data appears in the Sanity Studio under the respective schemas (Product and Category).

Each product is stored with fields such as:
Title: The product name (e.g., "Modern Sofa").
Slug: A unique identifier for dynamic routing (e.g., modern-sofa).
Price: The cost of the product (e.g., $299.99).
Description: A detailed product description.
Category Reference: Linked to a specific category in the Category schema.
Image: The product image, stored as an asset.


localhost:3000/studio/structure/product;product-10

Product

- Tropical Vibe
- Marble Ease
- Vase Set
- The Dandy chair
- Bright Space
- Cloud Haven Chair
- Bold Nest
- Sunny Chic
- Reflective Haven

Tropical Vibe

Tropical Vibe

Slug

tropical-vibe

Product Image

Published 18 sec. ago

<u>Sanity Studio Interface:</u>

Preview: Each product entry includes an image preview and basic details (title, price).

Search: I can search for products by title or slug, making it easy to verify data.

Edit: Fields are editable directly in Sanity Studio if updates are needed.

6. Browser Results After Migration

After integrating the data into Sanity CMS, I created a responsive frontend in Next.js to display the data. Here's what I achieved:

<u>1. Product List Page:</u>

A page displaying all products fetched from Sanity CMS.

Each product card includes the title, price, image, and a "View Details" button.

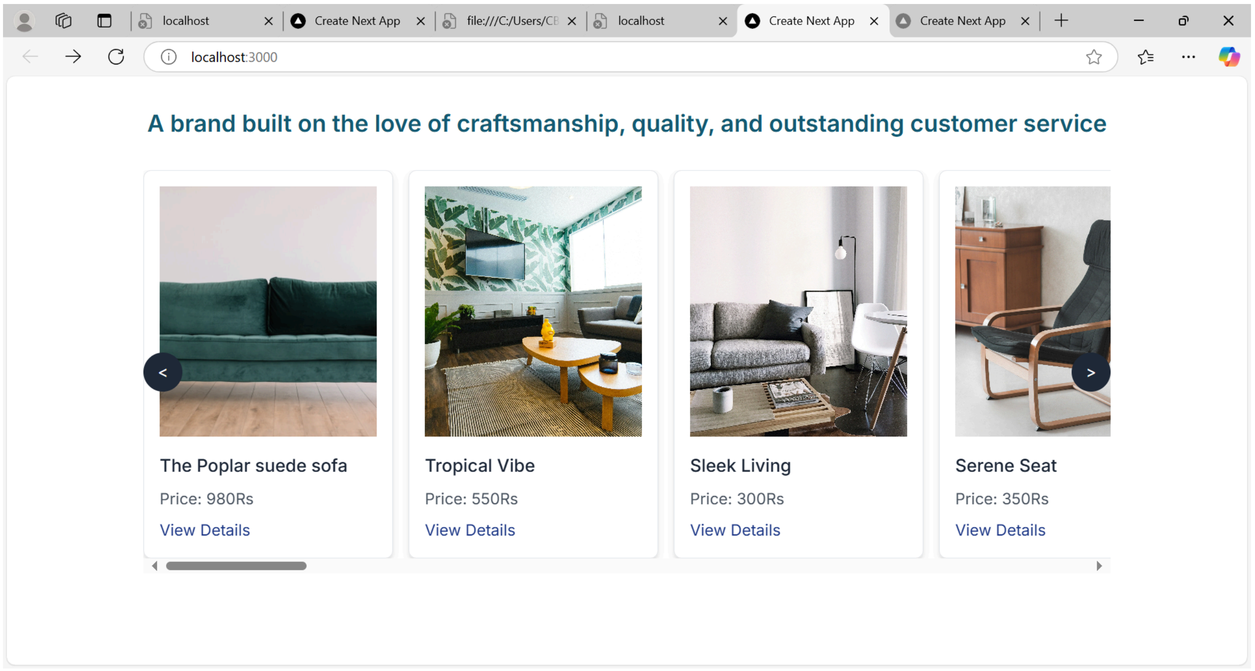
Result in Browser:

The product list page loads quickly, showing the following:

Images: Fetched from Sanity CMS and displayed responsively.

Title and Price: Properly formatted for each product.

"View Details" Button: Links to the individual product page using a dynamic slug.



A brand built on the love of craftsmanship, quality, and outstanding customer service

The Poplar suede sofa
Price: 980Rs

Tropical Vibe
Price: 550Rs

Sleek Living
Price: 300Rs

Serene Seat
Price: 350Rs

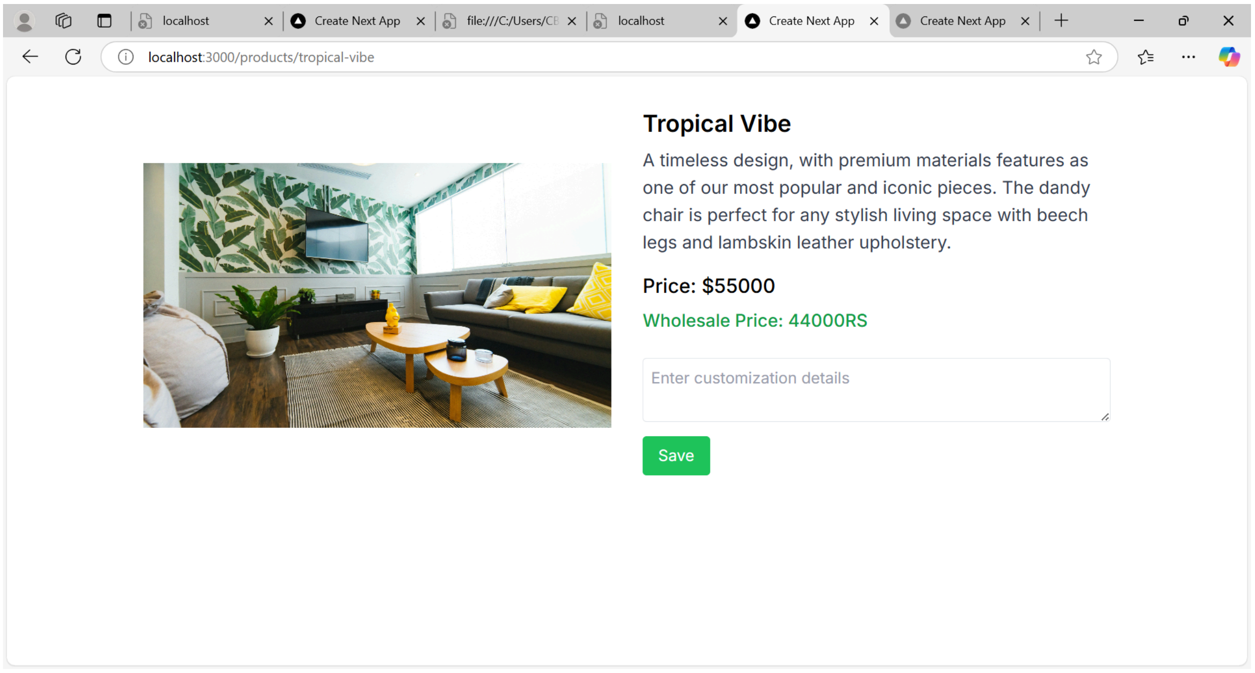
2. Dynamic Product Pages:

Each product detail page is dynamically generated based on its slug.

The page includes the title, description, price, and image.

Result in Browser:

Clicking "View Details" opens a new page showing detailed information about the selected product.



Tropical Vibe

A timeless design, with premium materials features as one of our most popular and iconic pieces. The dandy chair is perfect for any stylish living space with beech legs and lambskin leather upholstery.

Price: $55000

Wholesale Price: 44000RS

Enter customization details

Save

Conclusion (Key Learning Outcomes)

1. API Integration:

Successfully integrated the provided API into Next.js and fetched product data.

Validated API responses to ensure they matched the Sanity schema.

2. Data Migration:

Migrated API data into Sanity CMS using a script.

Verified that all fields (title, price, description, slug, etc.) were correctly populated in Sanity Studio.

3. Frontend Implementation:

Designed a responsive product list page and dynamic product detail pages using TailwindCSS and Next.js.

Verified browser results to ensure smooth navigation and accurate data display.

4. Sanity Studio Management:

Successfully mapped and stored data in Sanity Studio, with proper references and field validation.

Verified the schema's compatibility and ensured it was editable and searchable within the CMS.

This process enhanced my ability to integrate APIs, migrate data into headless CMSs like Sanity, and create a functional frontend, preparing me for real-world marketplace development challenges.