

Hackathon Day 03 Task

By Aimal Khan

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 given 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, 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 3, guided by Sir Fahad and Sir Ali Aftab Sheikh. Here's how I approached the task:

I used the provided API:

1. Shoes

<https://template-03-api.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.

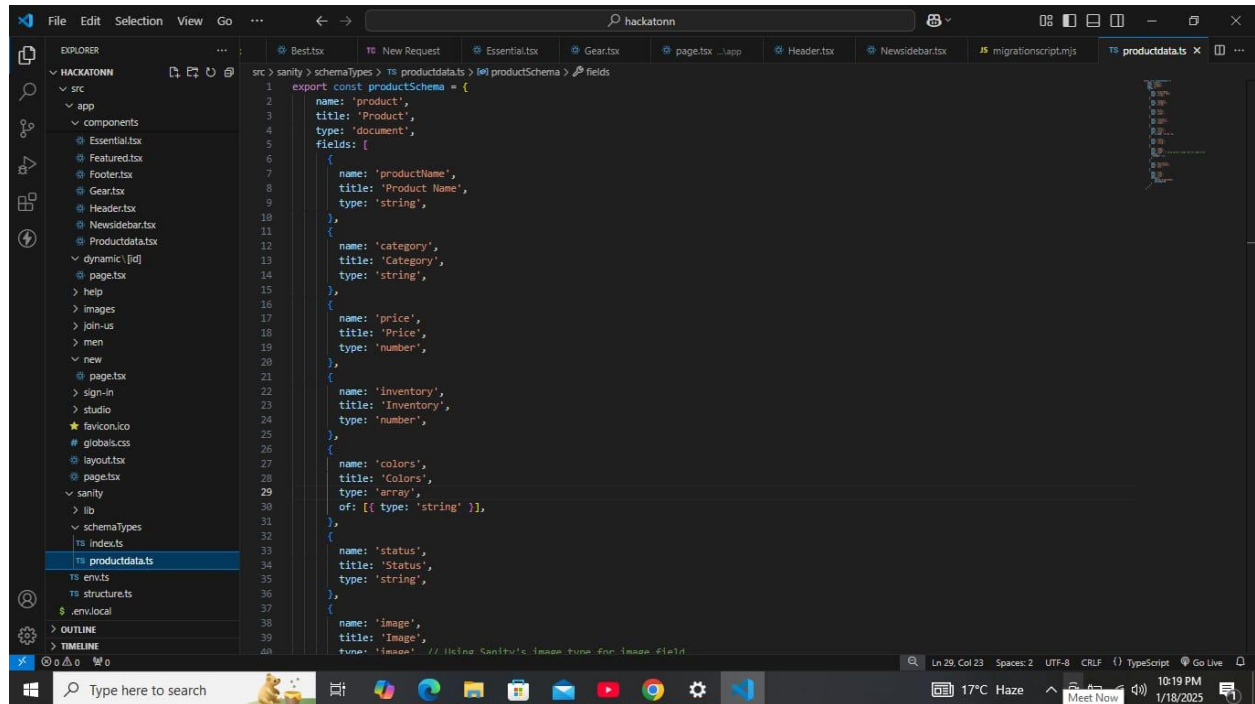
2. Sanity CMS Schemas Configuration:

I referred to the following schema definitions provided in the GitHub

repository to structure my Sanity CMS:

Category Schema:

productdata.ts

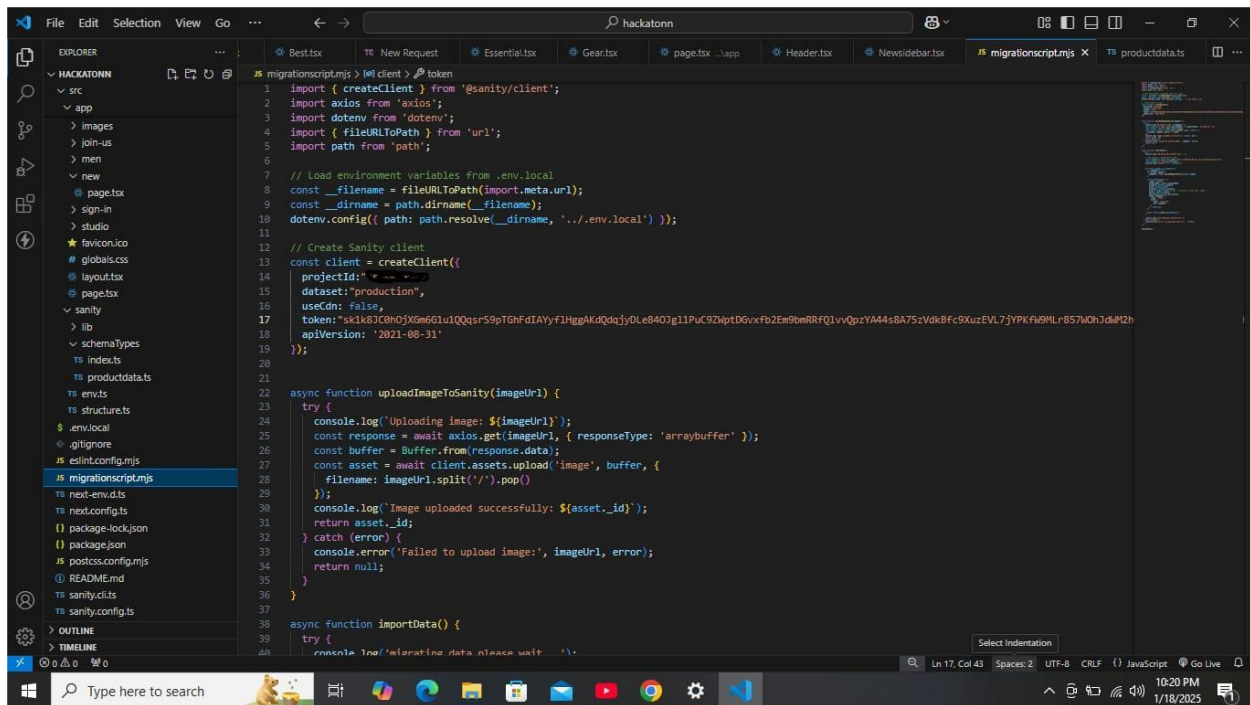
A screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project structure with folders like 'src', 'app', 'components', and 'sanity'. The 'sanity' folder is expanded, showing 'productdata.ts' selected. The main editor area displays the content of 'productdata.ts', which is a TypeScript file defining a Sanity schema for a product. The schema includes fields for 'product' (name, title, type), 'category' (name, title, type), 'price' (name, title, type), 'inventory' (name, title, type), 'colors' (name, title, type), 'status' (name, title, type), and 'image' (name, title, type). The file is named 'productdata.ts' and is located at 'src > sanity > schemaTypes > productdata.ts'. The status bar at the bottom shows 'Ln 29, Col 23', 'Spaces: 2', 'UTF-8', 'CRLF', 'TypeScript', and 'Go Live'.

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

3. Data Migration:

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

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

A screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project structure with folders like 'src', 'app', 'images', and 'sanity'. The file 'migrationscript.mjs' is selected. The main editor area displays the content of this file, which is a JavaScript migration script. The script imports necessary modules, creates a Sanity client with a specific token and project ID, and defines two asynchronous functions: 'uploadImageToSanity' for uploading image data and 'importData' for fetching product data from an external API. The script is written in a clean, readable style with proper indentation and comments.

```
1 import { createClient } from '@sanity/client';
2 import axios from 'axios';
3 import dotenv from 'dotenv';
4 import { fileURLToPath } from 'url';
5 import path from 'path';
6
7 // Load environment variables from .env.local
8 const __filename = fileURLToPath(import.meta.url);
9 const __dirname = path.dirname(__filename);
10 dotenv.config({ path: path.resolve(__dirname, '../.env.local') });
11
12 // Create Sanity client
13 const client = createClient({
14   projectId: '...',
15   dataset: 'production',
16   useCdn: false,
17   token: 'skk8jC0hOjXGm6Glu1QQsr59pTGHfDIAYyflHggAKQdajYdLe840jg11PuC9ZlptDGvxfB2Em9bmRRFQ1vvQpZyA44s8A75zVdkBfc9XuzEVL7jYPKFq9MLr857WOhJdM2h',
18   apiVersion: '2021-08-31'
19 });
20
21
22 async function uploadImageToSanity(imageUrl) {
23   try {
24     console.log('Uploading image: ${imageUrl}');
25     const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
26     const buffer = Buffer.from(response.data);
27     const asset = await client.assets.upload('image', buffer, {
28       filename: imageUrl.split('/').pop()
29     });
30     console.log('Image uploaded successfully: ${asset._id}');
31     return asset._id;
32   } catch (error) {
33     console.error('Failed to upload image: ', imageUrl, error);
34     return null;
35   }
36 }
37
38 async function importData() {
39   try {
40     console.log('Importing data from external API...');
```

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.

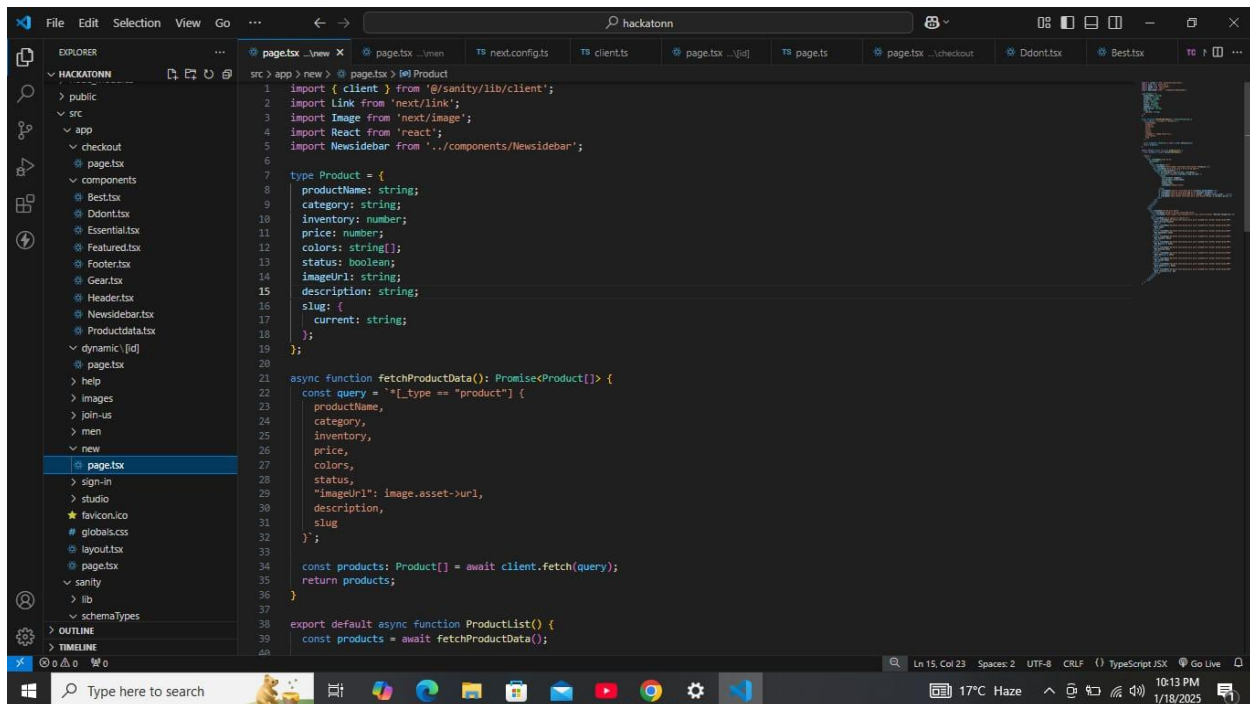
This process gave me a clear understanding of handling data sources, customizing APIs, and ensuring compatibility with headless CMS platforms like Sanity.

4. Frontend Implementation

Frontend Steps:

1. Fetching Products Data:

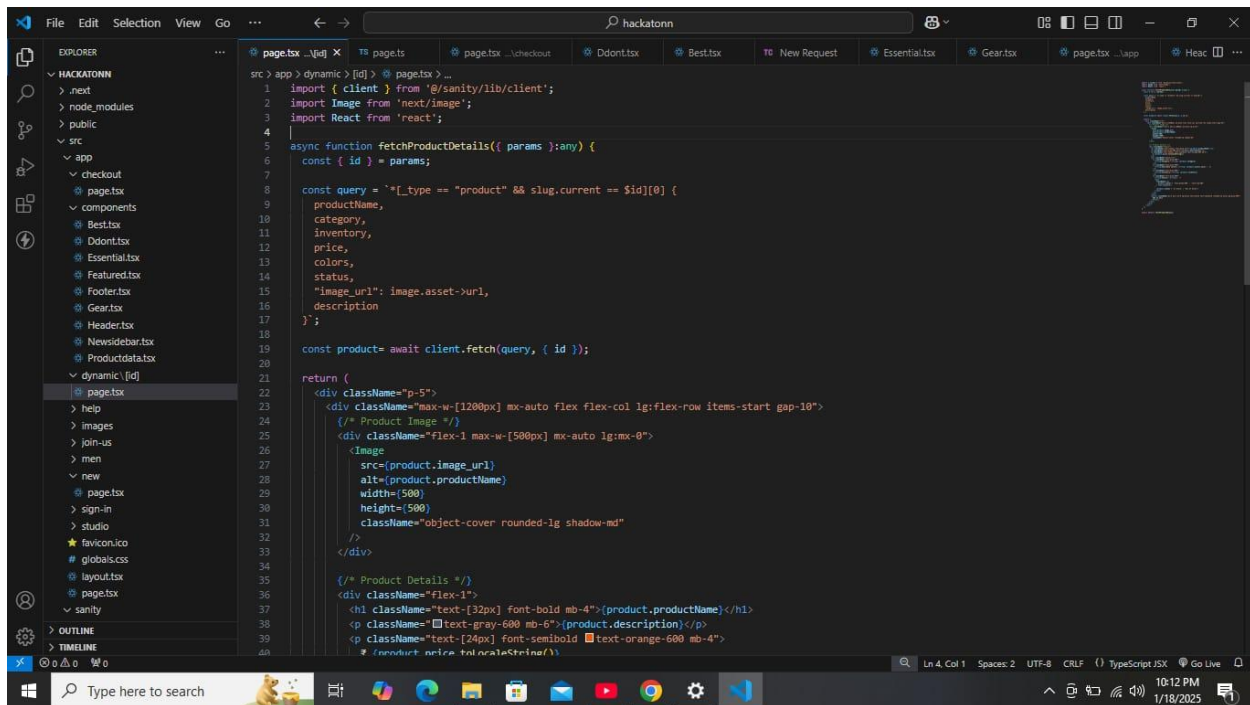
I used Sanity's GROQ queries in my Next.js project to fetch the data directly from the CMS.



```
1 import { client } from '@sanity/lib/client';
2 import Link from 'next/link';
3 import Image from 'next/image';
4 import React from 'react';
5 import Newsletter from '../components/Newsletter';
6
7 type Product = {
8   productName: string;
9   category: string;
10  inventory: number;
11  price: number;
12  colors: string[];
13  status: boolean;
14  imageUrl: string;
15  description: string;
16  slug: {
17    current: string;
18  };
19 };
20
21 async function fetchProductData(): Promise<Product[]> {
22   const query = `*[ _type == "product" ] {
23     productName,
24     category,
25     inventory,
26     price,
27     colors,
28     status,
29     "imageUrl": image.asset->url,
30     description,
31     slug
32   }`;
33
34   const products: Product[] = await client.fetch(query);
35   return products;
36 }
37
38 export default async function ProductList() {
39   const products = await fetchProductData();
40 }
```

2. Dynamic Product Pages:

I created a dynamic route to display detailed product information using Sanity's GROQ query based on the slug



5. Sanity Studio View After Migration

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

Each product is stored with fields such as:

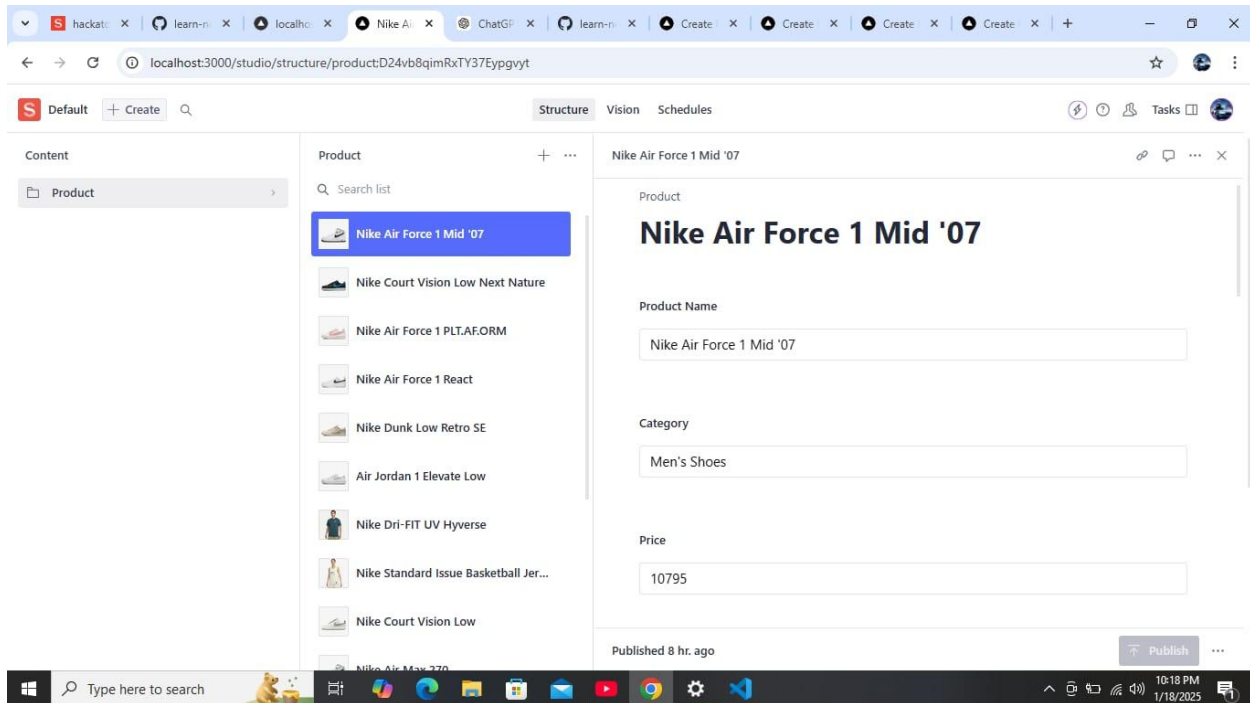
Product name: The product name (e.g., " Nike Air Force 1").

Slug: A unique identifier for dynamic routing (e.g., shoes).

Price: The cost of the product (e.g., \$650).

Description: A detailed product description.

Image: The product image, stored as an asset.



Sanity Studio Interface:

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

Search: I can search for products by 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:

1. Product List Page:

A page displaying all products fetched from Sanity CMS

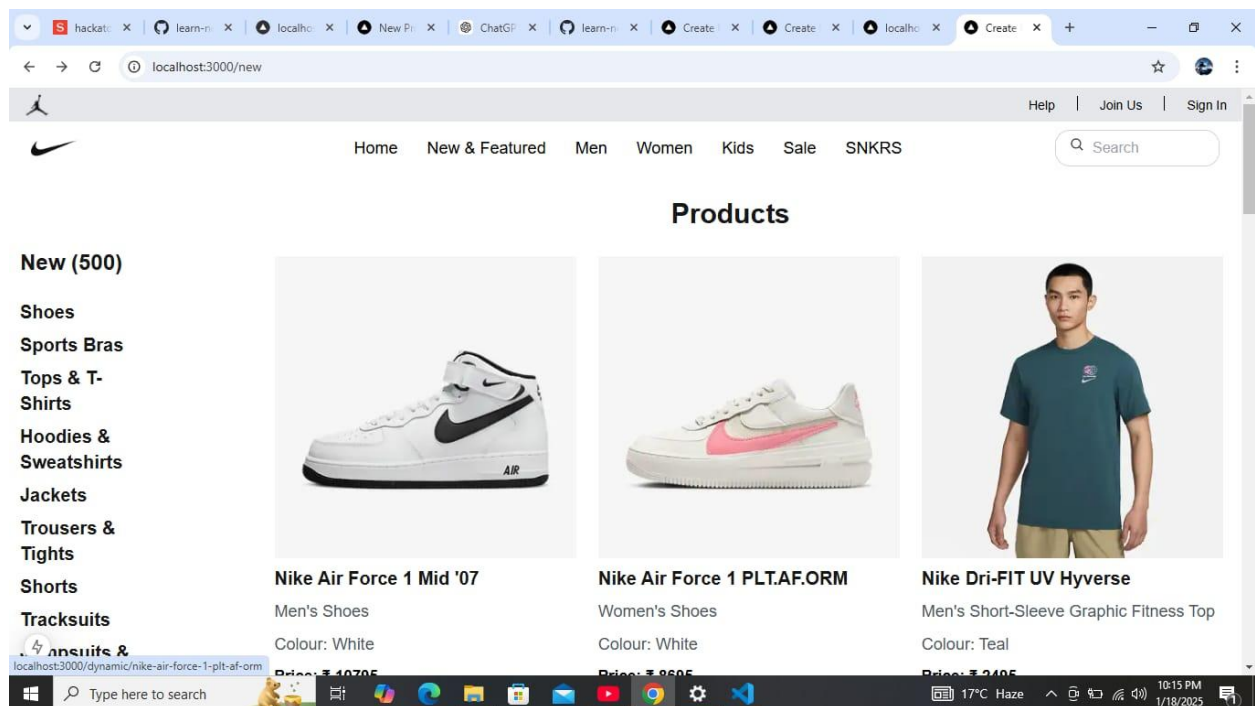
Each product card includes the productName, price, image, and a "View Details" page.

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



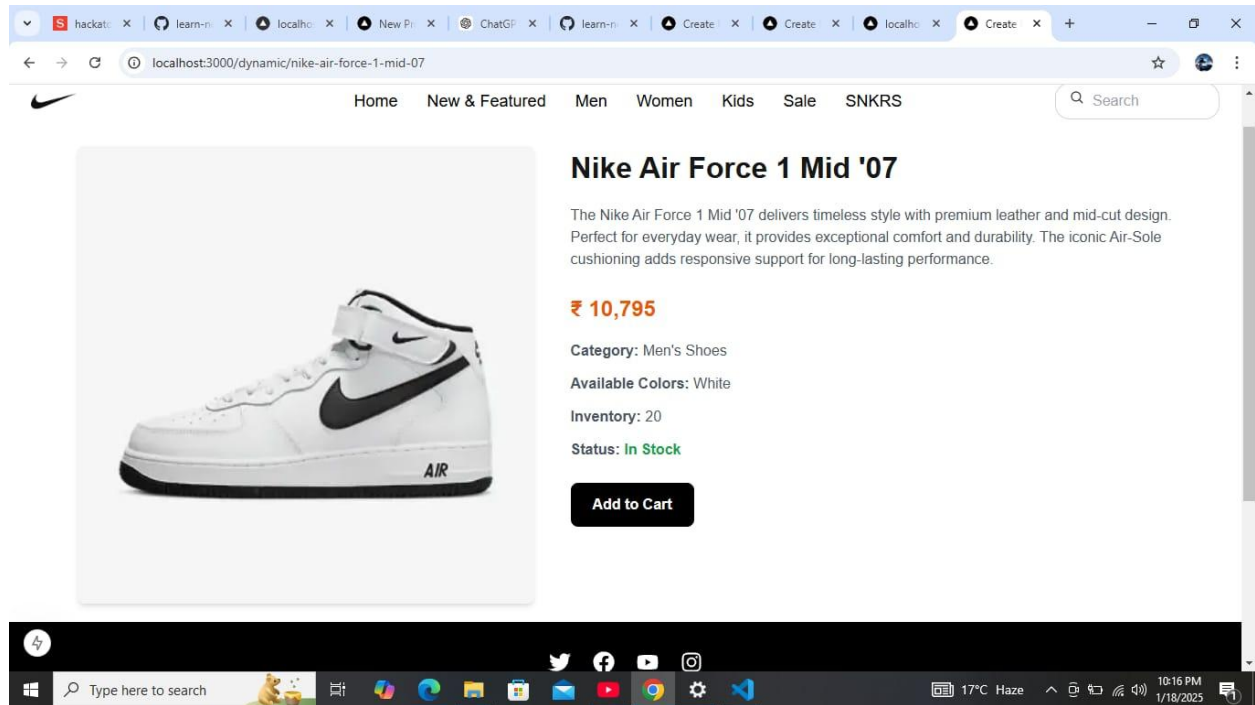
2. Dynamic Product Pages:

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

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

Result in Browser:

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



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 (productName, 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.