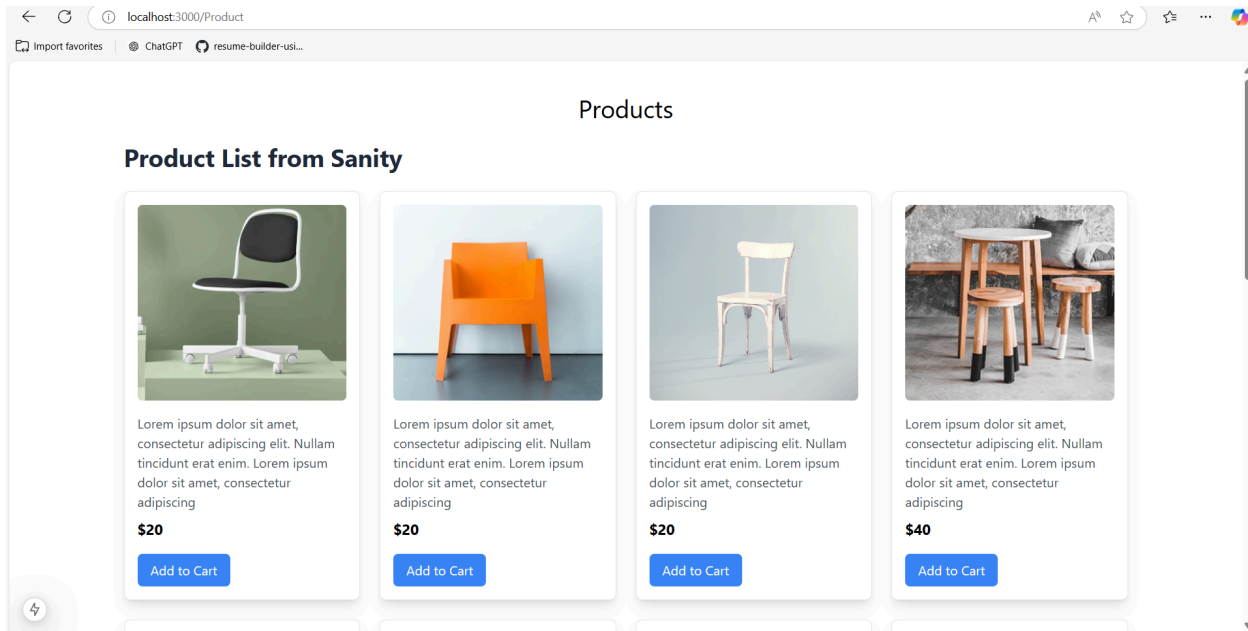


Frontend Display:



Challenges Faced

1. Data Structure Differences:

Some fields in the API response had to be mapped to different names in the Sanity schema.

2. Image Handling:

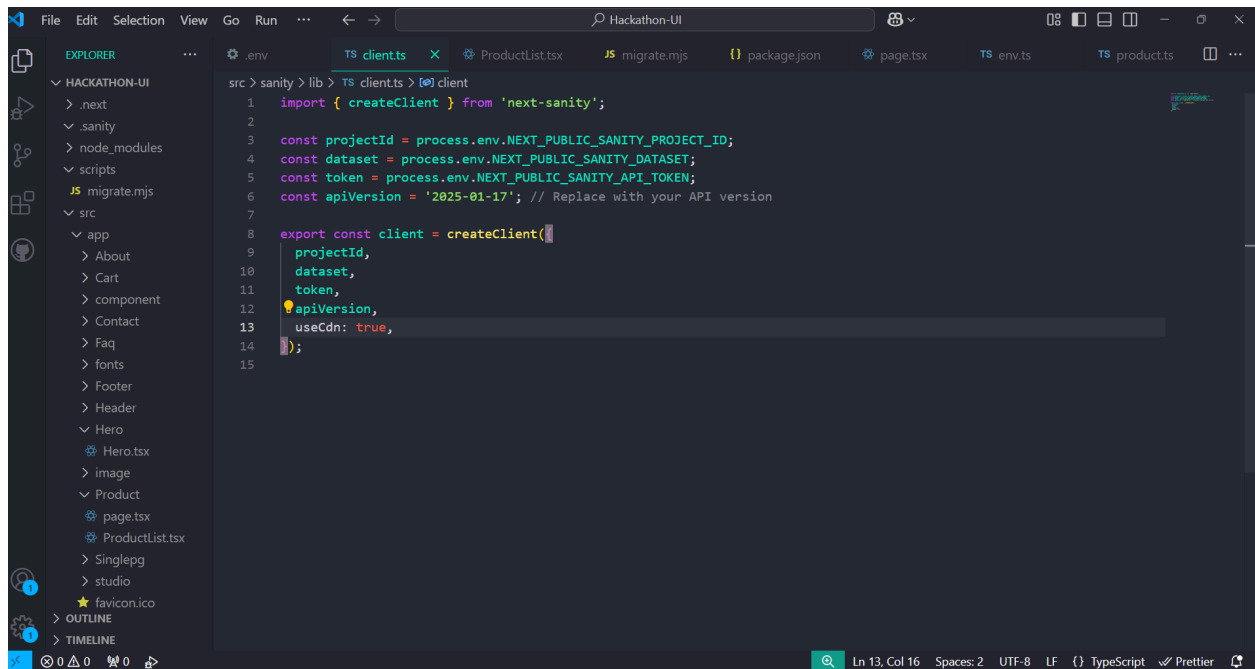
Migrating product images required converting URLs into Sanity's image reference format.

Solutions:

- Adjusted the schema field names to match the API.
- Used Sanity's `_type: 'image'` format to handle image uploads.

Client.ts Details:

The `client.ts` file in your project is essential for securely connecting your frontend application to the **Sanity backend**. This file contains the configuration and authentication details that allow your app to fetch and manage data from Sanity CMS.



```
File Edit Selection View Go Run ... Hackathon-UI
EXPLORER
  HACKATHON-UI
    .next
    .sanity
    node_modules
    scripts
    migrate.mjs
    src
      app
        About
        Cart
        component
        Contact
        Faq
        fonts
        Footer
        Header
        Hero
        Hero.tsx
        image
        Product
        page.tsx
        ProductList.tsx
        Singlepg
        studio
      favicon.ico
    OUTLINE
    TIMELINE

src > sanity > lib > TS clients > client
1 import { createClient } from 'next-sanity';
2
3 const projectId = process.env.NEXT_PUBLIC_SANITY_PROJECT_ID;
4 const dataset = process.env.NEXT_PUBLIC_SANITY_DATASET;
5 const token = process.env.NEXT_PUBLIC_SANITY_API_TOKEN;
6 const apiVersion = '2025-01-17'; // Replace with your API version
7
8 export const client = createClient({
9   projectId,
10  dataset,
11  token,
12  apiVersion,
13  useCdn: true,
14 });
15
```

Ln 13, Col 16 Spaces: 2 UTF-8 LF {} TypeScript Prettier

Steps Taken Today

1. Sanity Installation:

- I started by installing Sanity in the project to use it as the backend for managing data.

2. Schema Creation:

- After setting up Sanity, I created schemas to define the structure for storing product data (e.g., name, price, image, and description).

3. Migration File Creation:

- I created a `migrate.mjs` file to handle the migration of data from the API into Sanity's backend.

4. API Integration:

- Imported the product API into the project.
- Created a `ProductList` page to display all the products dynamically by fetching data from the API.

5. Image Display Error:

- While displaying products on the browser, an issue occurred with the images not showing up.
- Solved this by adjusting the Sanity image settings and making necessary updates to the code.

6. API Integration with Sanity:

- Integrated the product API with Sanity CMS, ensuring all the product data from the API was stored correctly in the backend.

7. Fetching Data from Sanity:

- Successfully fetched data from Sanity CMS and displayed it on the browser.
- Used `client.ts` to securely fetch data from Sanity in the `ProductList` page.

8. Environment File Setup:

- Secured the project by adding sensitive keys and configurations to the `.env` file.

9. Final Browser Output:

- After solving multiple issues, including image rendering and API errors, the product list (with names, prices, images, and descriptions) was displayed correctly on the browser.
-

Challenges Solved

1. Image Rendering Issue:

- Resolved image display problems by adjusting Sanity image settings and securely fetching data using `client.ts`.

2. API Integration with Sanity:

- Overcame challenges related to mapping API data with Sanity's structure.

3. Secure Configuration:

- Ensured all sensitive data (like API keys) was securely stored in the `.env` file.