API INTEGRATION

Hackathon 3 (Day 3)

OVERVIEW

Objectives:

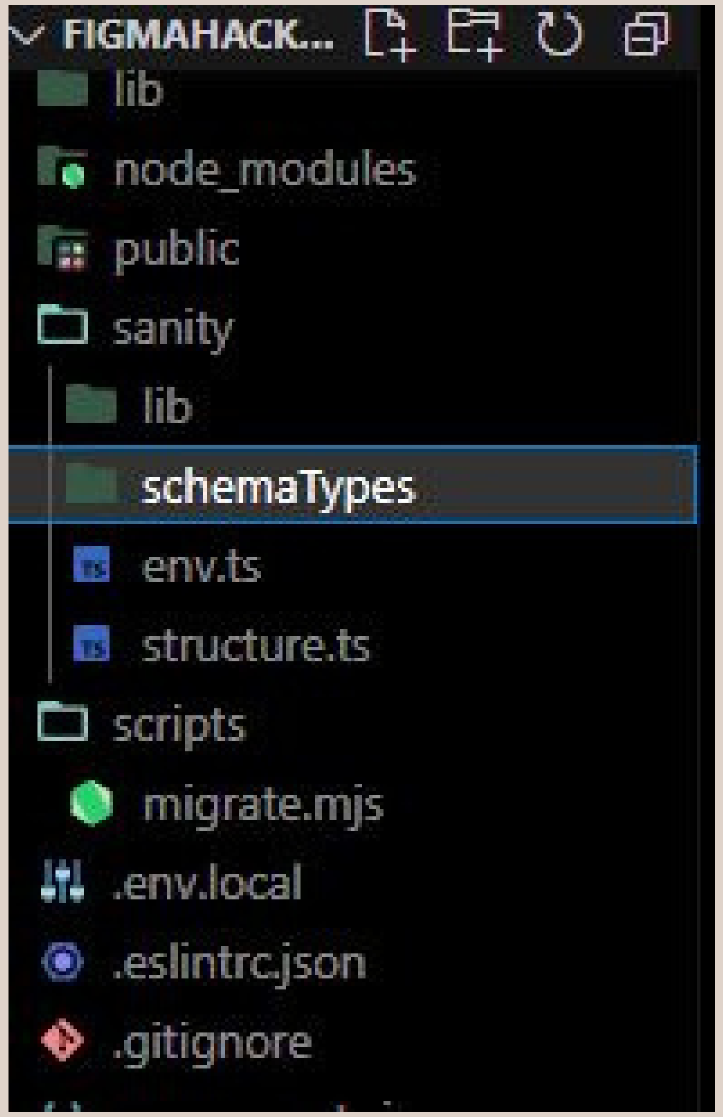
1.Integrate API data into Sanity CMS
. 2.Migrate and validate data from Template 6 API.
3.Ensure a fully functional CMS backend.

Key Tasks:

•Set up and connect Sanity CMS
. •Add the API token securely
. •Create and implement the provided schema.
•Run a migration script to import data.

Setting Up Sanity CMS

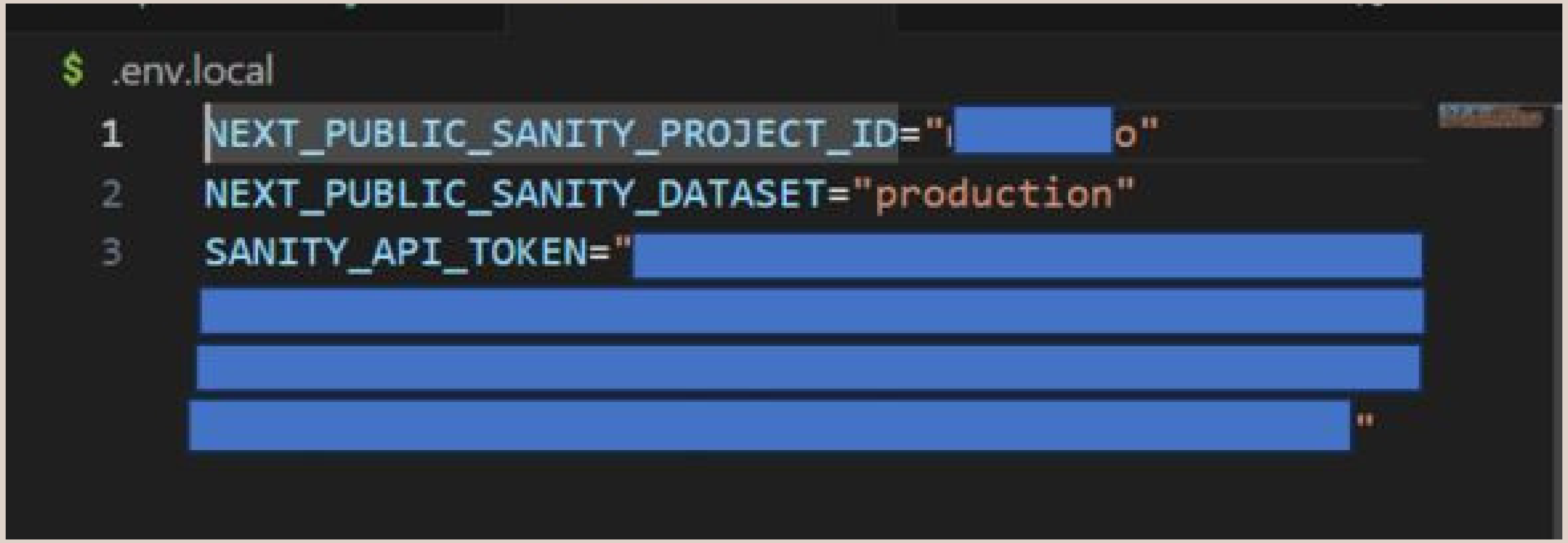
Created a Sanity CMS project on sanity.io and connected it to my local Next.js project. During the setup, I initialized the project locally using the Sanity CLI, which created a sanity folder with all the necessary configuration files.



```
FIGMAHACK...
lib
node_modules
public
sanity
  lib
  schemaTypes
  env.ts
  structure.ts
scripts
  migrate.mjs
.env.local
.eslintrc.json
.gitignore
```

Adding the API Token

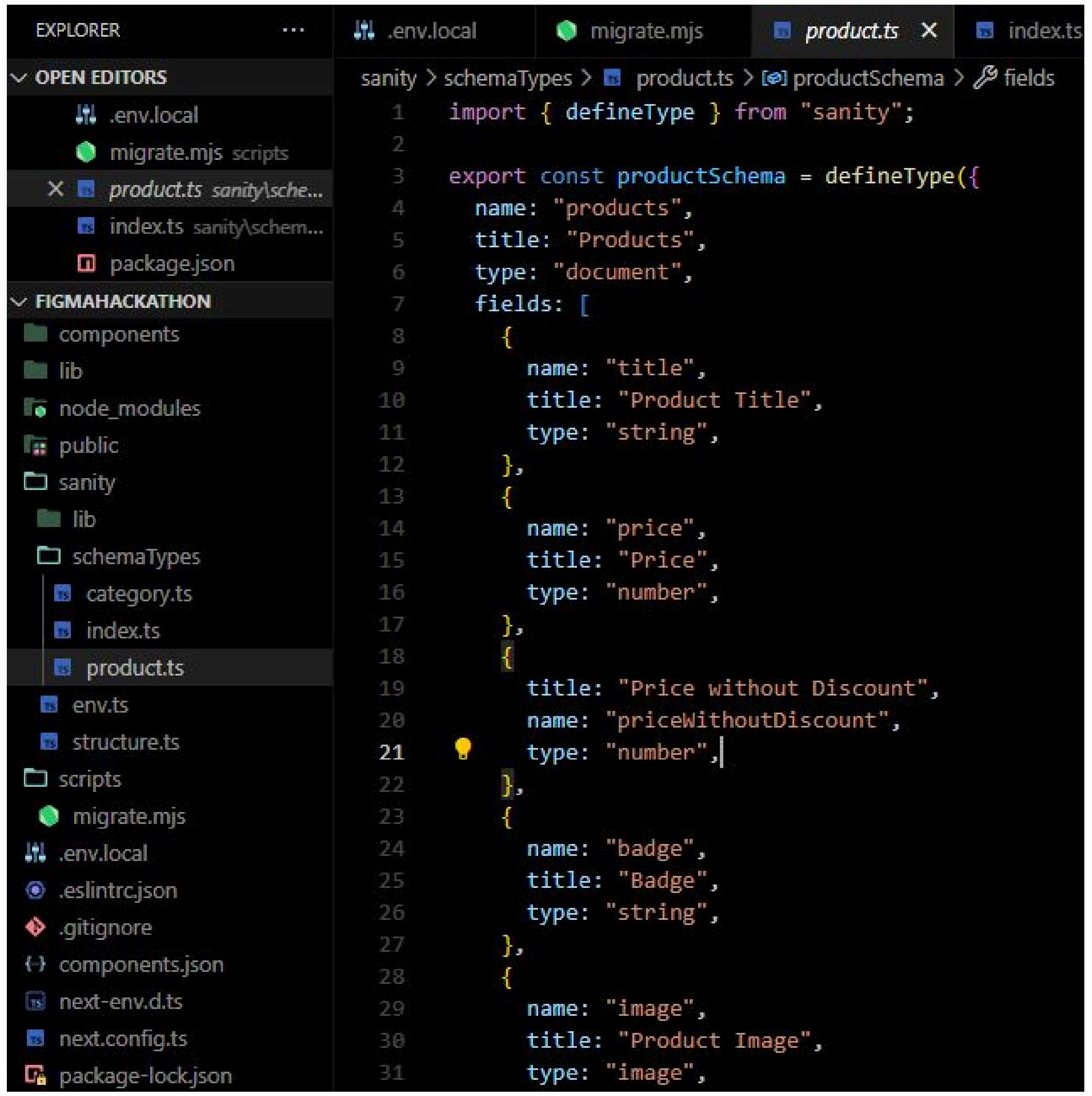
To securely access the Sanity CMS project, generated an API token on the Sanity dashboard. Then added the Project ID, Dataset Name, and API Token to the .env.local file for secure access.



```
$ .env.local
NEXT_PUBLIC_SANITY_PROJECT_ID="[redacted]o"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="[redacted]"
```

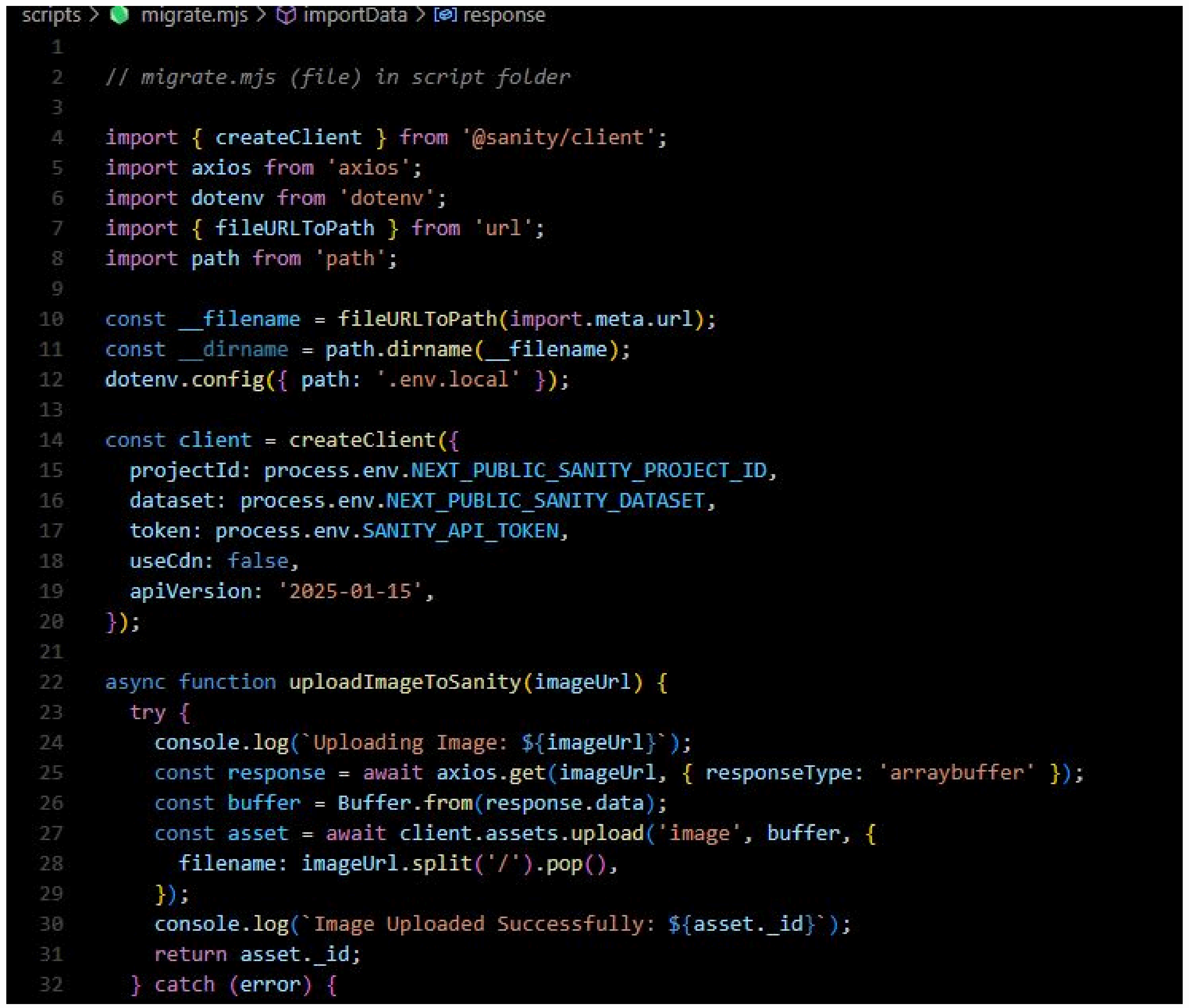
Creating the Schema

Added the provided product.ts schema for Template 6 and added it to the sanity/schemaTypes folder.



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

export const productSchema = defineType({
  name: "products",
  title: "Products",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Product Title",
      type: "string",
    },
    {
      name: "price",
      title: "Price",
      type: "number",
    },
    {
      title: "Price without Discount",
      name: "priceWithoutDiscount",
      type: "number",
    },
    {
      name: "badge",
      title: "Badge",
      type: "string",
    },
    {
      name: "image",
      title: "Product Image",
      type: "image",
```

Data Migration Script



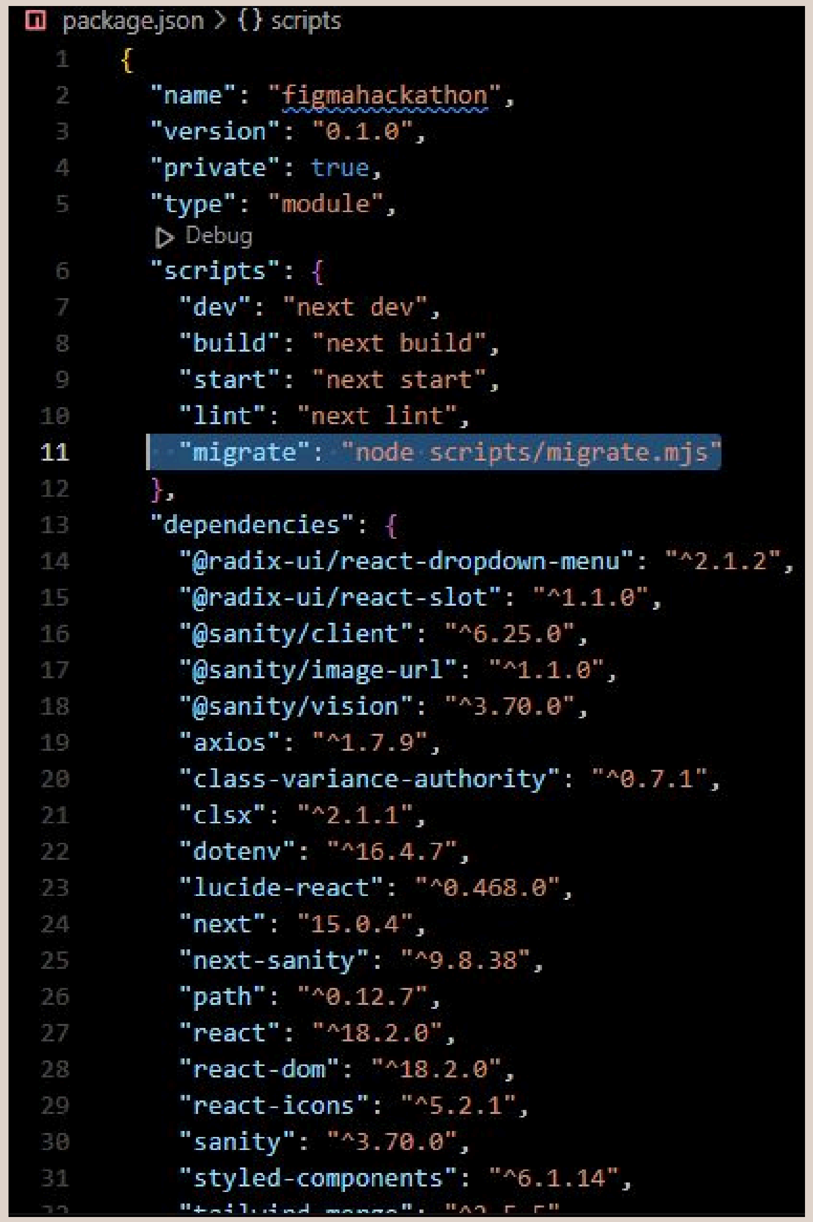
scripts > migrate.mjs > importData > response

```javascript

// migrate.mjs (file) in script folder

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

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: '.env.local' });

const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  token: process.env.SANITY_API_TOKEN,
  useCdn: false,
  apiVersion: '2025-01-15',
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading Image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(),
    });
    console.log(`Image Uploaded Successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
```

To automate the migration of data from the API to Sanity, created a script named import-data.mjs in the scripts folder. The script fetched the data from the provided API and imported it into Sanity CMS.

Running the Migration Script

Added a new script to package.json to simplify the execution of the migration:



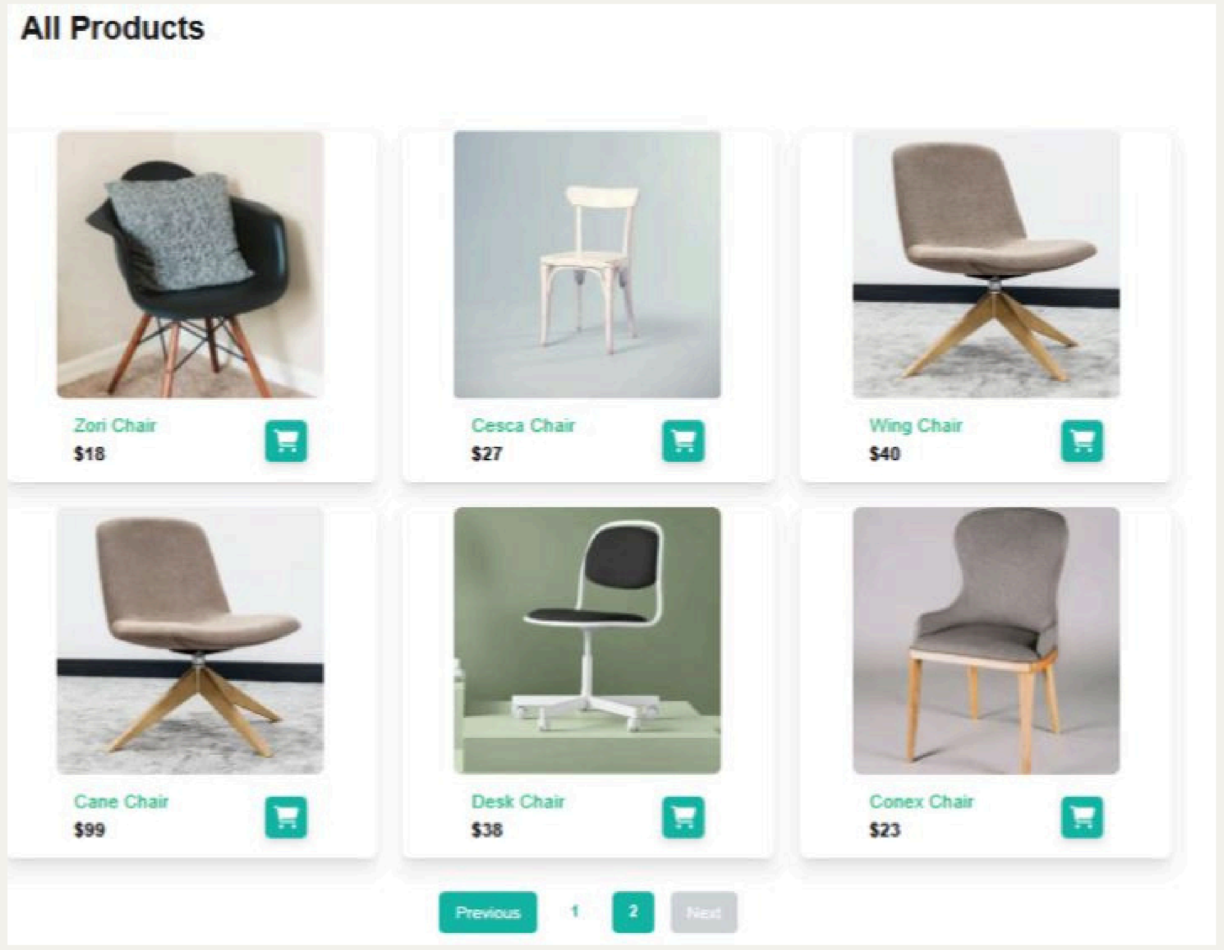
package.json > {} scripts

```
{
  "name": "figmahackathon",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  ▷ Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "migrate": "node scripts/migrate.mjs"
  },
  "dependencies": {
    "@radix-ui/react-dropdown-menu": "^2.1.2",
    "@radix-ui/react-slot": "^1.1.0",
    "@sanity/client": "^6.25.0",
    "@sanity/image-url": "^1.1.0",
    "@sanity/vision": "^3.70.0",
    "axios": "^1.7.9",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "dotenv": "^16.4.7",
    "lucide-react": "^0.468.0",
    "next": "15.0.4",
    "next-sanity": "^9.8.38",
    "path": "^0.12.7",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^5.2.1",
    "sanity": "^3.70.0",
    "styled-components": "^6.1.14",
```

Then ran the script using: npm run migrate

Featured and Categories Data Fetching:

- Products are categorized (e.g., featured, new arrivals, etc.) and can be ltered dynamically using category-based APIs.
- Category Filtering: Users can view products based on categories (e.g., Men’s Fashion, Electronics), or they can browse specic featured products that highlight top-selling items.
- This system streamlines product discovery, allowing users to quickly nd what they are looking for.



All Products

Zori Chair $18

Cesca Chair $27

Wing Chair $40

Cane Chair $99

Desk Chair $38

Conex Chair $23

What We Learned from This Project

Dynamic Data Handling:

We gained valuable skills in working with dynamic data fetching and routing, making our application exible and responsive to data changes.

User Experience Enhancements:

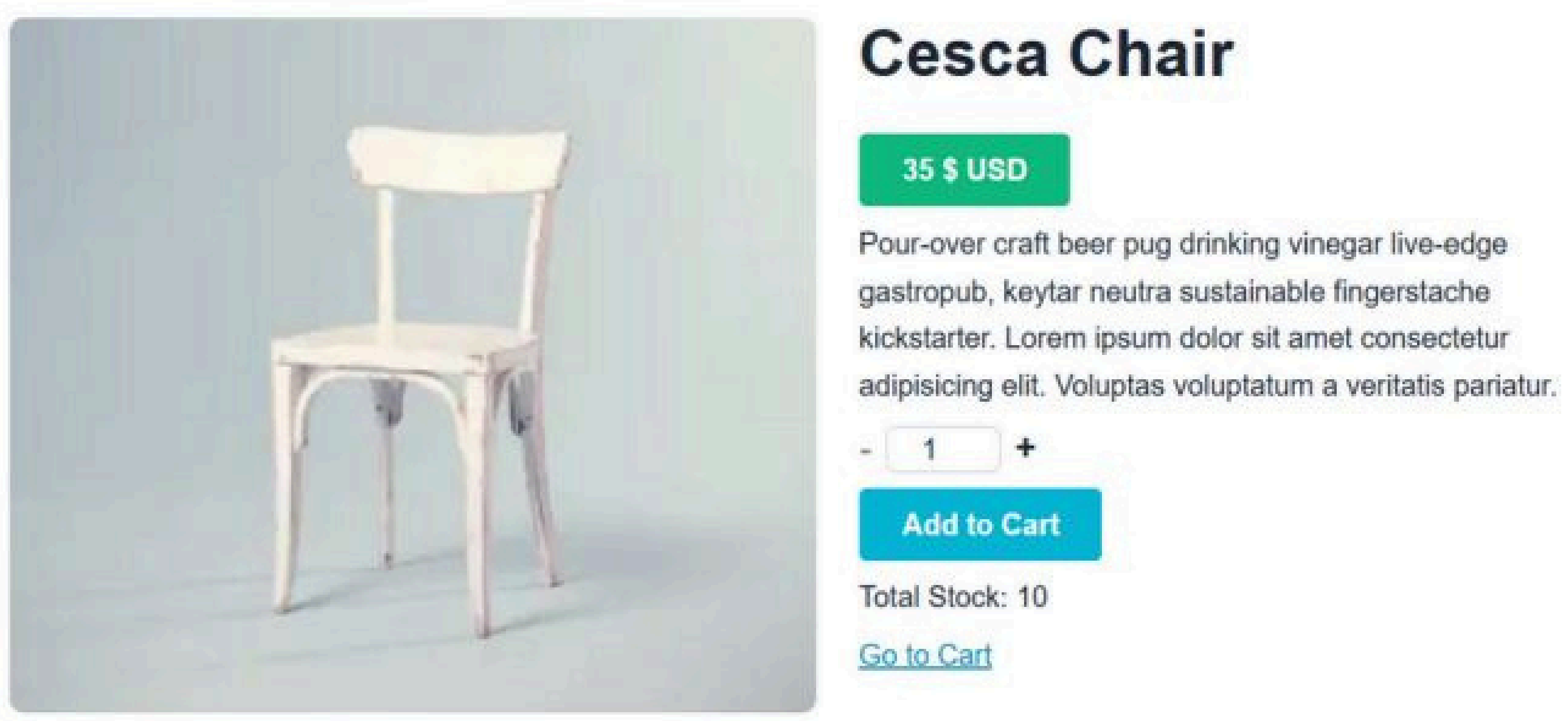
By adding features like search, lters, pagination, and related products, we learned how to improve the usability and overall experience for customers.

Frontend Optimization:

We learned how to implement pagination and lazy loading to optimize the performance of the frontend, making sure pages load faster and handle large datasets eciently.

Responsive Web Design:

We explored how to build responsive web pages using Tailwind CSS, ensuring that the website looks great on any device, which is crucial for modern web development.


Cesca Chair

35 $ USD

Pour-over craft beer pug drinking vinegar live-edge gastropub, keytar neutra sustainable fingerstache kickstarter. Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptas voluptatum a veritatis pariatur.

Total Stock: 10

Go to Cart

Conclusion

In conclusion, the Comforty website oers a dynamic and responsive e-commerce platform, featuring an attractive selection of chairs, stools, and sofas. By integrating product listings, category lters, and pagination, it provides an intuitive shopping experience. The website's design ensures accessibility on all devices, and its modular components are built for future scalability, creating a seamless and efficient platform for users to explore and purchase stylish furniture.