Marketplace Hackathon 2024

Day-03

prepared By:

Arjumand Afreen Tabinda

Project Documentation Day 3: Integrating Sanity API with Next.js

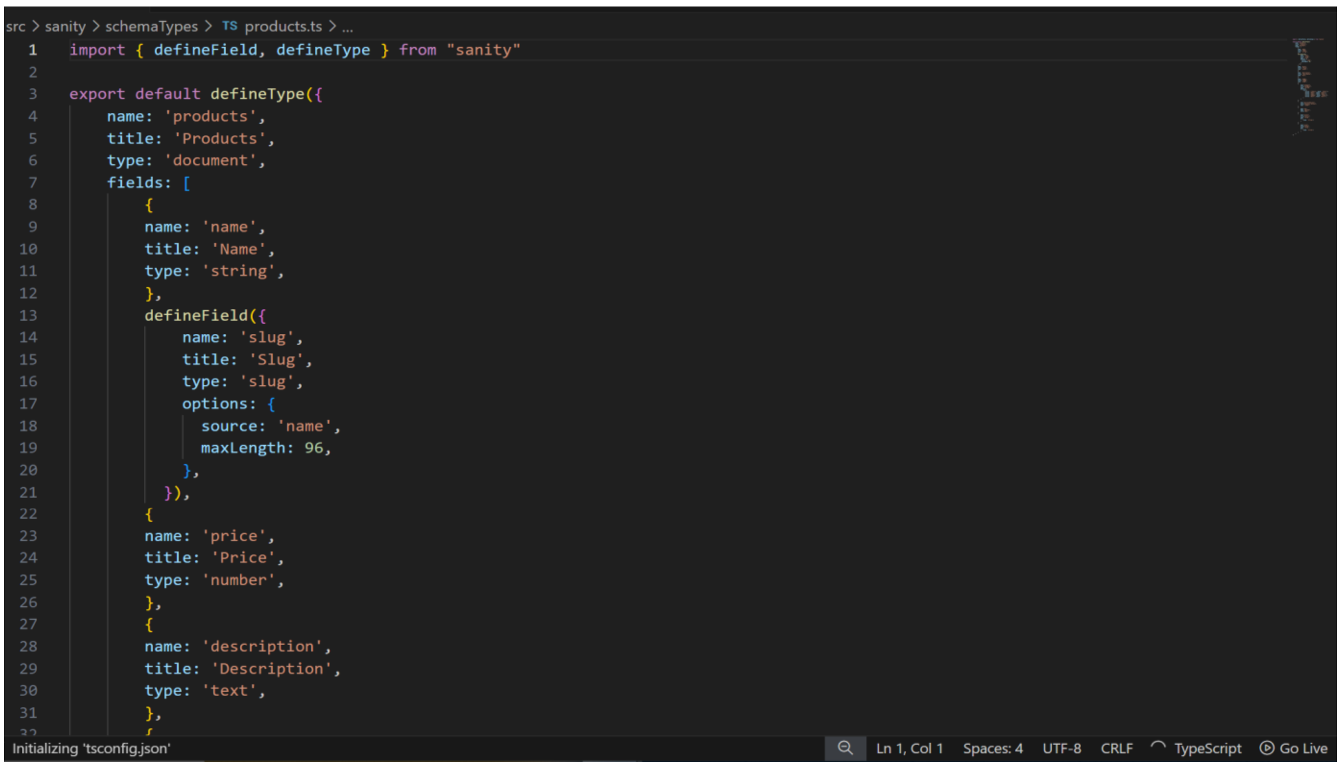
Step 1: Installing Next.js
To start, I set up Next.js in my project directory by running:
npx create-next-app@latest
This initialized a new Next.js project in the selected folder, providing a base for building the application.

Step 2: Installing Sanity
Next, I installed Sanity CMS to handle my content management by executing:
npm install @sanity/client
Sanity CMS is a headless CMS that lets me manage and retrieve my data. I used it to manage products, categories, and other essential information for the project.

Step 3: Setting Up Environment Variables
I created an .env file at the root of my project to securely store environment variables. Inside this file, I defined the following variables:
NEXT_PUBLIC_SANITY_PROJECT_ID=your-project-id
NEXT_PUBLIC_SANITY_DATASET=your-dataset-name
SANITY_API_TOKEN=your-api-token
These variables were used to configure the connection to the Sanity API.

Step 4: Generating API Token
To authenticate with the Sanity API, I generated an API token via the Sanity dashboard. Afterward, I added this token to the .env file under the SANITY_API_TOKEN variable.

Step 5: Defining Schema Types
I then set up the schema types in the Sanity dashboard to define the structure of my data. This process involved creating schema definitions for products, categories, discounts, etc. Each schema included fields such as name, price, description, image, and other required attributes.


src > sanity > schemaTypes > TS products.ts > ...

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

export default defineType({
    name: 'products',
    title: 'Products',
    type: 'document',
    fields: [
        {
        name: 'name',
        title: 'Name',
        type: 'string',
        },
        defineField({
            name: 'slug',
            title: 'Slug',
            type: 'slug',
            options: {
              source: 'name',
              maxLength: 96,
            },
          }),
        {
        name: 'price',
        title: 'Price',
        type: 'number',
        },
        {
        name: 'description',
        title: 'Description',
        type: 'text',
        },
```

Initializing 'tsconfig.json' Ln 1, Col 1 Spaces: 4 UTF-8 CRLF TypeScript Go Live

Note: *I made sure to verify the provided API and schemas to confirm that they aligned correctly with the expected structure for the project.*
Step 6: Verifying the API Endpoint
Since I was unable to verify the data during the upload process, I used tools like Postman to validate the Sanity API endpoint. I sent a GET request to the Sanity API to check whether the data was returned as expected. This confirmed that the API was

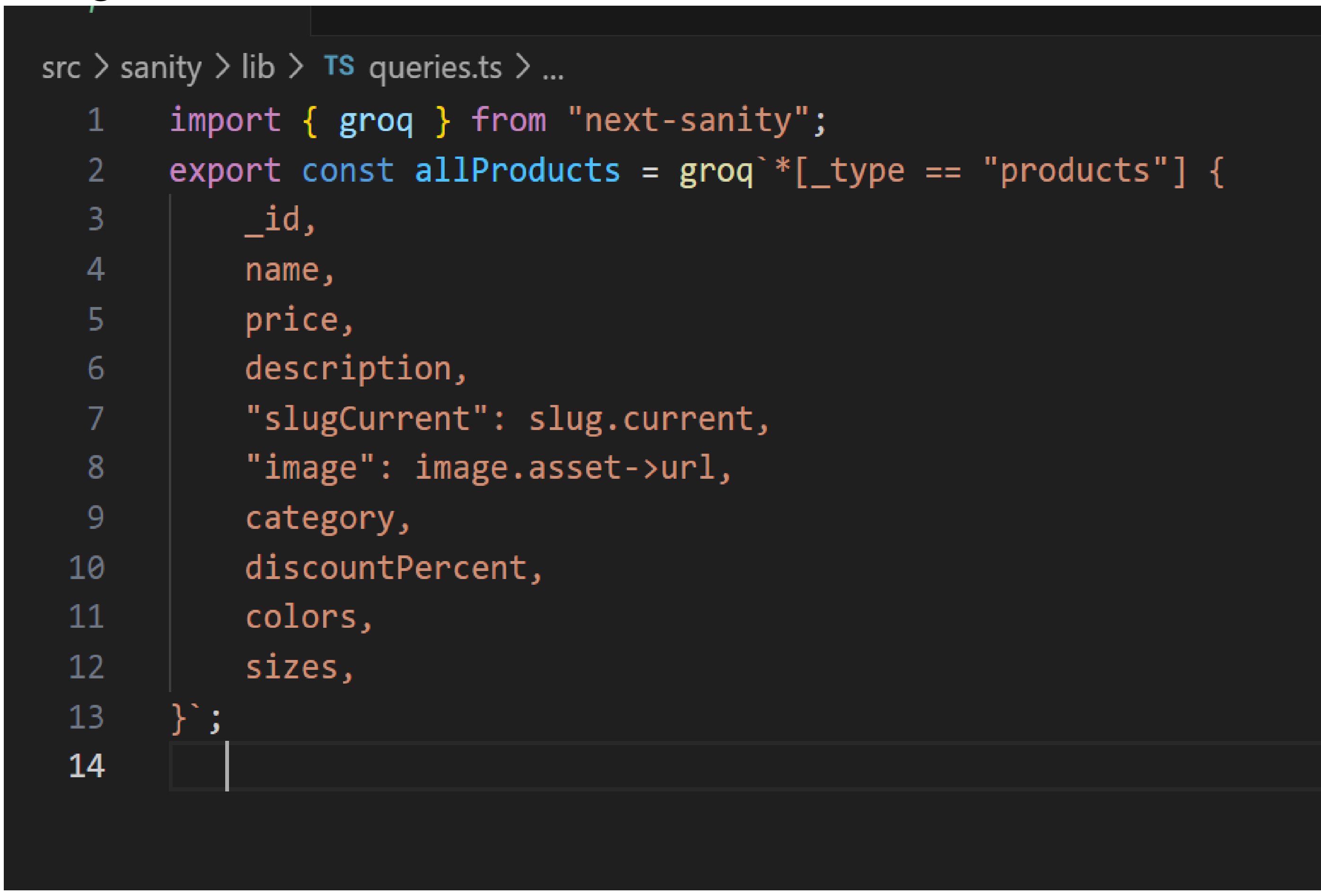
functioning correctly and the data was accessible for use in my project.

Step 7: Setting Up Fetch and Queries Files

To maintain a clean code structure, I created two files: fetch.ts and queries.ts.

- ***fetch.ts:*** *This file contained the function responsible for retrieving data from the Sanity API.*

queries.ts: *This file contained the essential queries for retrieving data, such as fetching products and categories.*

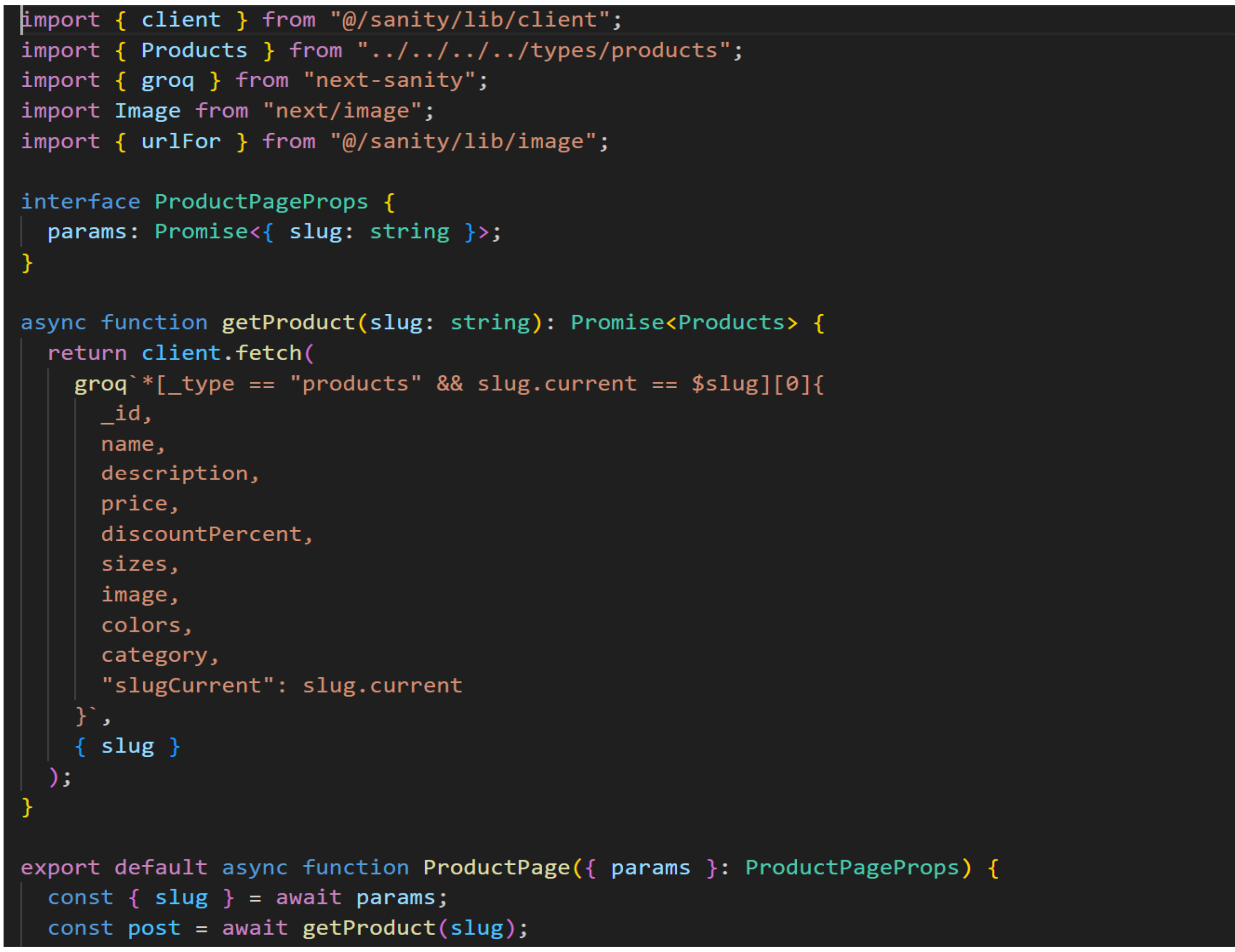


src > sanity > lib > TS queries.ts > ...

```
import { groq } from "next-sanity";
export const allProducts = groq`*[_type == "products"] {
    _id,
    name,
    price,
    description,
    "slugCurrent": slug.current,
    "image": image.asset->url,
    category,
    discountPercent,
    colors,
    sizes,
}`;

```

Step 8: Creating the Scripts Folder

I set up a scripts folder within my project. Inside this folder, I created the importData.mjs file, where I configured the provided migration data. This file facilitated the import of all the data into Sanity.



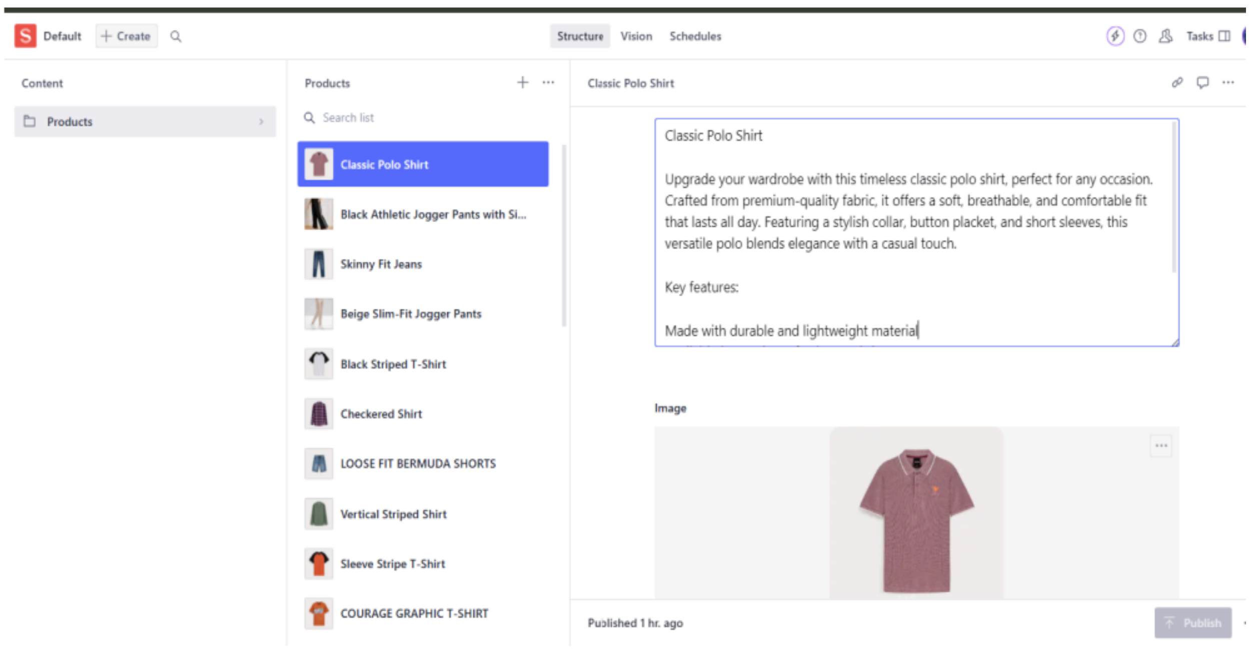
```
import { client } from "@/sanity/lib/client";
import { Products } from "../../../../types/products";
import { groq } from "next-sanity";
import Image from "next/image";
import { urlFor } from "@/sanity/lib/image";

interface ProductPageProps {
  params: Promise<{ slug: string }>;
}

async function getProduct(slug: string): Promise<Products> {
  return client.fetch(
    groq`*[_type == "products" && slug.current == $slug][0]{
      _id,
      name,
      description,
      price,
      discountPercent,
      sizes,
      image,
      colors,
      category,
      "slugCurrent": slug.current
    }`,
    { slug }
  );
}

export default async function ProductPage({ params }: ProductPageProps) {
  const { slug } = await params;
  const post = await getProduct(slug);
```

I also specified the necessary commands in package.json to easily execute the scripts:

"scripts": {
  "import-data": "node scripts/importData.mjs"
}

Step 9: Importing Data into Sanity

I ran the script defined in the importData.mjs file to import all the provided migration data into Sanity. This ensured that the data was ready for use in my Next.js application. The data was successfully imported into Sanity.



Classic Polo Shirt
Black Athletic Jogger Pants with Si...
Skinny Fit Jeans
Beige Slim-Fit Jogger Pants
Black Striped T-Shirt
Checkered Shirt
LOOSE FIT BERMUDA SHORTS
Vertical Striped Shirt
Sleeve Stripe T-Shirt
COURAGE GRAPHIC T-SHIRT

Classic Polo Shirt

Upgrade your wardrobe with this timeless classic polo shirt, perfect for any occasion. Crafted from premium-quality fabric, it offers a soft, breathable, and comfortable fit that lasts all day. Featuring a stylish collar, button placket, and short sleeves, this versatile polo blends elegance with a casual touch.

Key features:

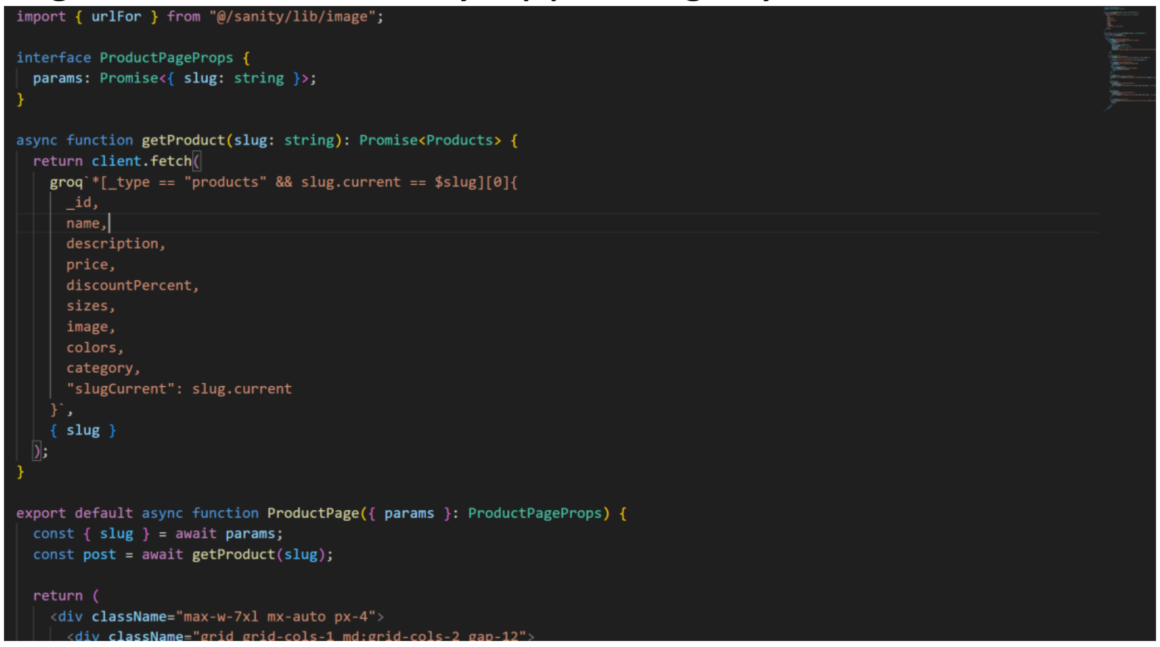
Made with durable and lightweight material

Published 1 hr. ago

Step 10: Retrieving Data in Next.js

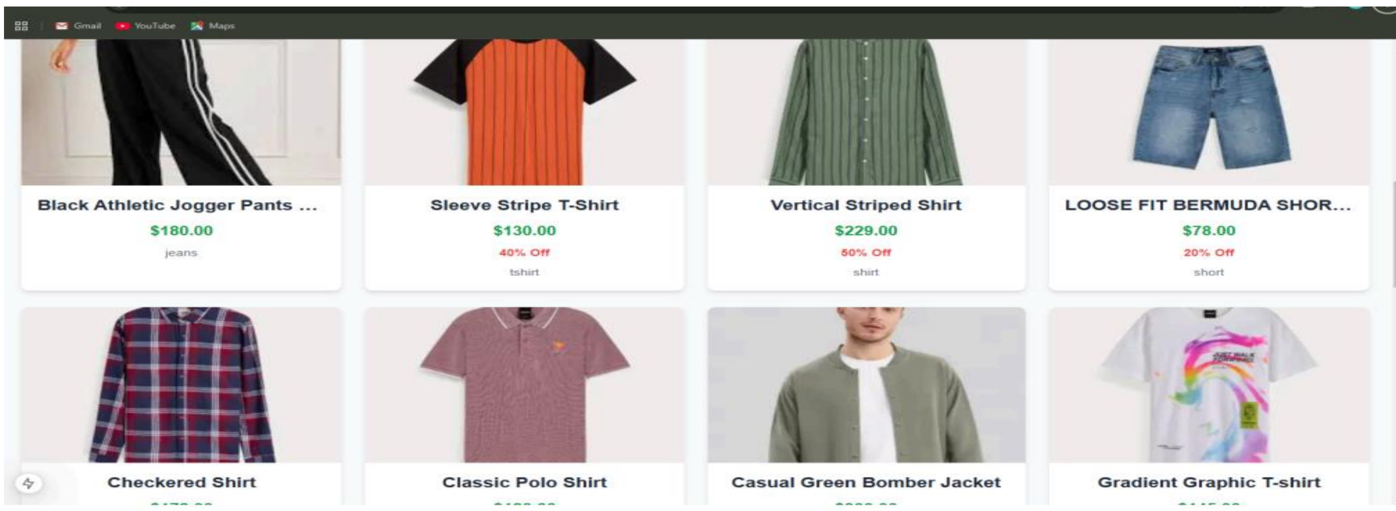
After successfully importing the data into Sanity, I proceeded with presenting it in the user interface. Instead of utilizing getServerSideProps, I created type definitions for products and other data models. I

then imported the fetch.ts and queries.ts files into my component.
By utilizing the sanityFetch function (defined in the fetch.ts file), I retrieved the data and displayed it on the UI. The data was passed as props to the component, enabling me to showcase product details such as the name, price, description, and image in an organized and visually appealing layout.



```
import { urlFor } from "@/sanity/lib/image";

interface ProductPageProps {
  params: Promise<{ slug: string }>;
}

async function getProduct(slug: string): Promise<Products> {
  return client.fetch(
    groq`*[_type == "products" && slug.current == $slug][0]{
      _id,
      name,
      description,
      price,
      discountPercent,
      sizes,
      image,
      colors,
      category,
      "slugCurrent": slug.current
    }`,
    { slug }
  );
}

export default async function ProductPage({ params }: ProductPageProps) {
  const { slug } = await params;
  const post = await getProduct(slug);

  return (
    <div className="max-w-7xl mx-auto px-4">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-12">
```

Step 11: Rendering Data on the UI
Finally, I utilized Next.js to build a user interface that automatically displayed the retrieved product data. Each product was shown in a card format, featuring the name, price, description, and image.



Black Athletic Jogger Pants ...
$180.00
jeans

Sleeve Stripe T-Shirt
$130.00
40% Off
tshirt

Vertical Striped Shirt
$229.00
50% Off
shirt

LOOSE FIT BERMUDA SHOR...
$78.00
20% Off
short

Checkered Shirt

Classic Polo Shirt

Casual Green Bomber Jacket

Gradient Graphic T-shirt