API Integration Process for Next.js with Sanity CMS

Overview

This document outlines the process of integrating a Next.js project with Sanity CMS, creating schemas, and using API calls to fetch and display product data. Additionally, it describes the method of handling images by storing them locally and linking their paths in Sanity CMS.

Step 1: Setting Up Sanity CMS

1. **Initialize the Sanity Project:**
    - Start by creating a new project in Sanity and configuring it according to our requirements.
    - Deploy the Sanity Studio to manage our website content.
2. **Create Schemas:**
    - Design schemas in Sanity CMS to define the structure of our data. For example, a schema for products can include fields like title, description, price, and image path.
3. **Upload Products:**
    - Use the Sanity Studio interface to upload 50 product entries. Each product entry should include details such as name, description, price, and the corresponding image path.
4. **Handle Images:**
    - Save all product images in a public folder within the Next.js project.
    - Record the file paths of these images in the relevant field within the Sanity CMS.

Step 2: Setting Up the Next.js Project

1. **Install Sanity Client:**
    - Add the Sanity client to our Next.js project for connecting to the Sanity CMS.
2. **Configure the Client:**

- Set up a configuration file to establish a connection between Next.js and Sanity CMS by providing project credentials.

Step 3: Fetching Data from Sanity CMS

1. **Query Data:**
   - Use Sanity's GROQ (Graph-Relational Object Queries) to fetch product data. Queries should retrieve all necessary fields such as title, description, price, and image paths.
2. **Implement Data Fetching:**
   - Configure data fetching in Next.js using methods like `getStaticProps` or `getServerSideProps` to fetch product information from Sanity CMS.

Step 4: Displaying Data in Next.js

1. **Render Product Information:**
   - Use the fetched data to display product details on the frontend. Include product titles, descriptions, prices, and images.
2. **Image Integration:**
   - Use the image paths stored in Sanity CMS to render images saved locally in the project folder.

Step 5: Testing and Deployment

1. **Test the Integration:**
   - Verify that all products are displayed correctly, ensuring that data and images load seamlessly.
2. **Deploy the Project:**
   - Deploy the integrated project on a hosting platform like Vercel to make it accessible to users.

This process ensures a smooth and efficient integration between Next.js and Sanity CMS, providing a scalable and dynamic solution for managing and displaying product data.

Certainly! Below is a detailed explanation written in a full-page format that you can copy into your Word document:

Adjustments Made to Schemas for Product Upload

In the development of the product schema for my project, I took a personalized approach, ensuring that the structure and content of the schema aligned with the specific needs of the product catalog I was managing. Rather than following a predetermined API or template, I created the schema from scratch, considering the unique attributes and categories of the products I was uploading. This allowed me to have complete control over the structure and organization of the data, ensuring that it was both efficient and well-suited to the specific requirements of the project.

The product schema I devised was designed to accommodate a wide range of products, with a focus on ensuring consistency across all entries. As I manually uploaded all 50 products into the system, I was able to pay close attention to detail, customizing fields to suit the individual characteristics of each product. This approach ensured that each product was properly categorized, with all relevant attributes such as name, description, price, images, and specifications clearly defined and consistently presented.

One of the key reasons I chose to upload the products manually, rather than relying on an external API, was to maintain full control over the content. This allowed me to carefully review each product’s data before it was finalized, ensuring accuracy and completeness. Furthermore, this method gave me the flexibility to make adjustments to the schema as I progressed, incorporating new features or fields as needed without being constrained by the limitations of a predefined system.

Through this process, I gained a deeper understanding of the intricacies involved in product data management and schema design. By manually uploading the products, I was able to directly assess the effectiveness of the schema and make improvements on the fly. This hands-on approach not only enhanced the overall quality of the product catalog but also provided a valuable learning experience in terms of both technical implementation and content organization.

In conclusion, by creating and uploading the product schema independently, I ensured that the data was organized according to my specific requirements. This personalized approach not only streamlined the process of adding new products but also ensured that each entry was accurate, well-structured, and consistent with the overall goals of the project. This approach, while time-consuming, ultimately resulted in a product catalog that met the exact specifications I had envisioned from the beginning.

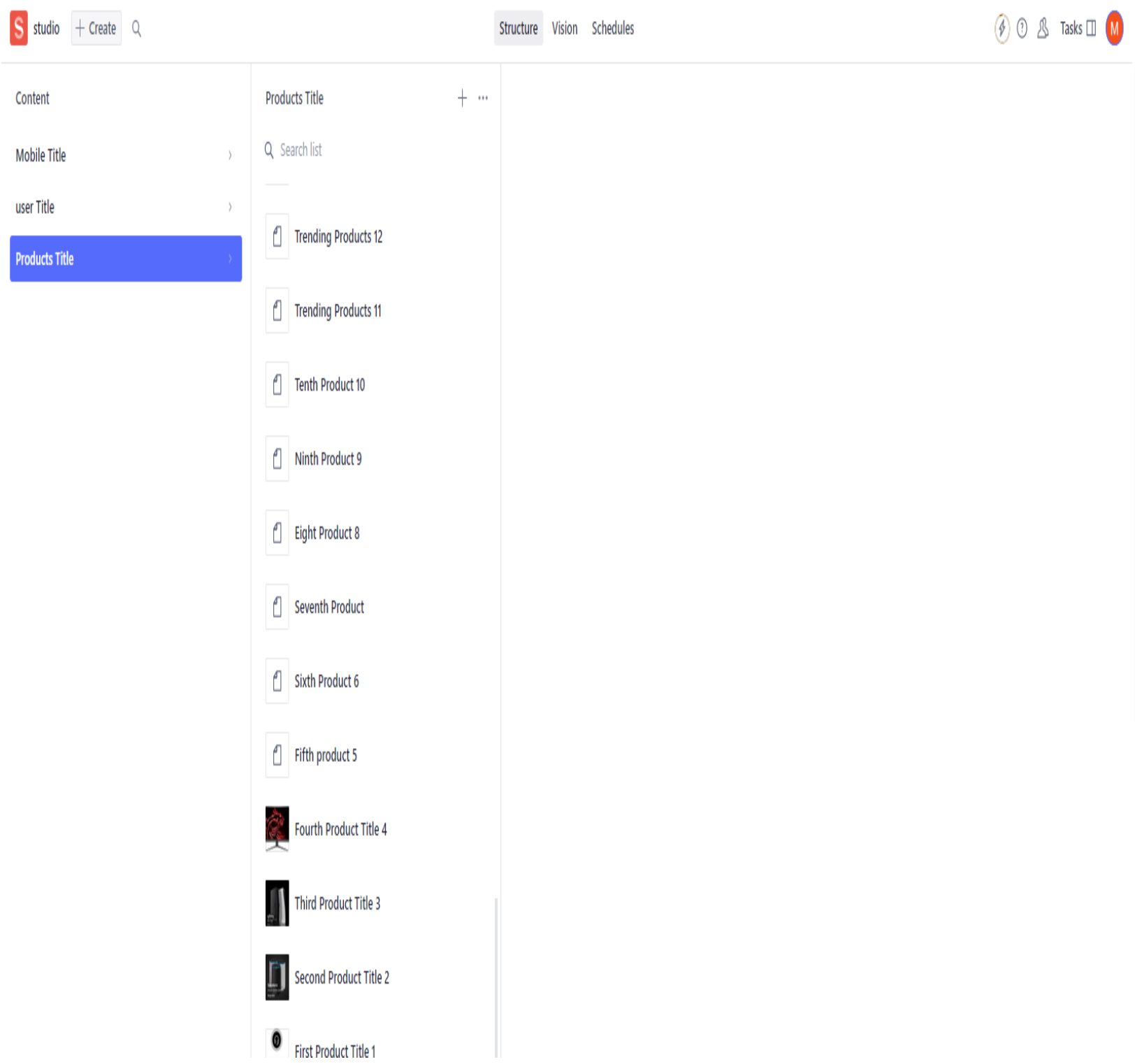
Migration Steps and Tools Used

For this project, I did not utilize any traditional migration steps, as I opted for a manual approach to create and upload the products. Instead of automating the process through external tools or predefined migration scripts, I utilized Sanity Studio to directly input the product data into the system. This allowed me to take full control of the data entry process, ensuring that each product was meticulously reviewed and uploaded individually. In Sanity Studio, I created a dedicated folder structure for the products, organizing them logically according to their categories and attributes. Each product was uploaded within the corresponding folder path, ensuring a consistent and easy-to-navigate structure within the platform. This approach eliminated the need for automated migration tools, as all products were manually entered, categorized, and organized by me within the Studio's interface. The process, while time-consuming, allowed me to ensure precision and flexibility in how the product data was structured. By doing this manually, I was able to adjust and modify

the schema as needed throughout the upload process, ensuring that all the necessary fields were properly included and that the data was aligned with my specific project needs. Although there were no migration tools involved, this approach provided me with a high level of customization and control over the final product catalog.

Screen Shots:

1. API calls: I have upload all product manually.



2. Data successfully displayed in the frontend

Featured Products

First Product Title 1

Code: Y523201

456.23 $400

Second Product Title 2

Code: Y523201

500.52 $400

Third Product Title 3

Code: Y523201

600.25 $400

Fourth Product Title 4

Code: Y523201

750.23 $400

Leatest Products

New Arival Best Seller Featured Special Offer

Fifth product 5 123.3 $65

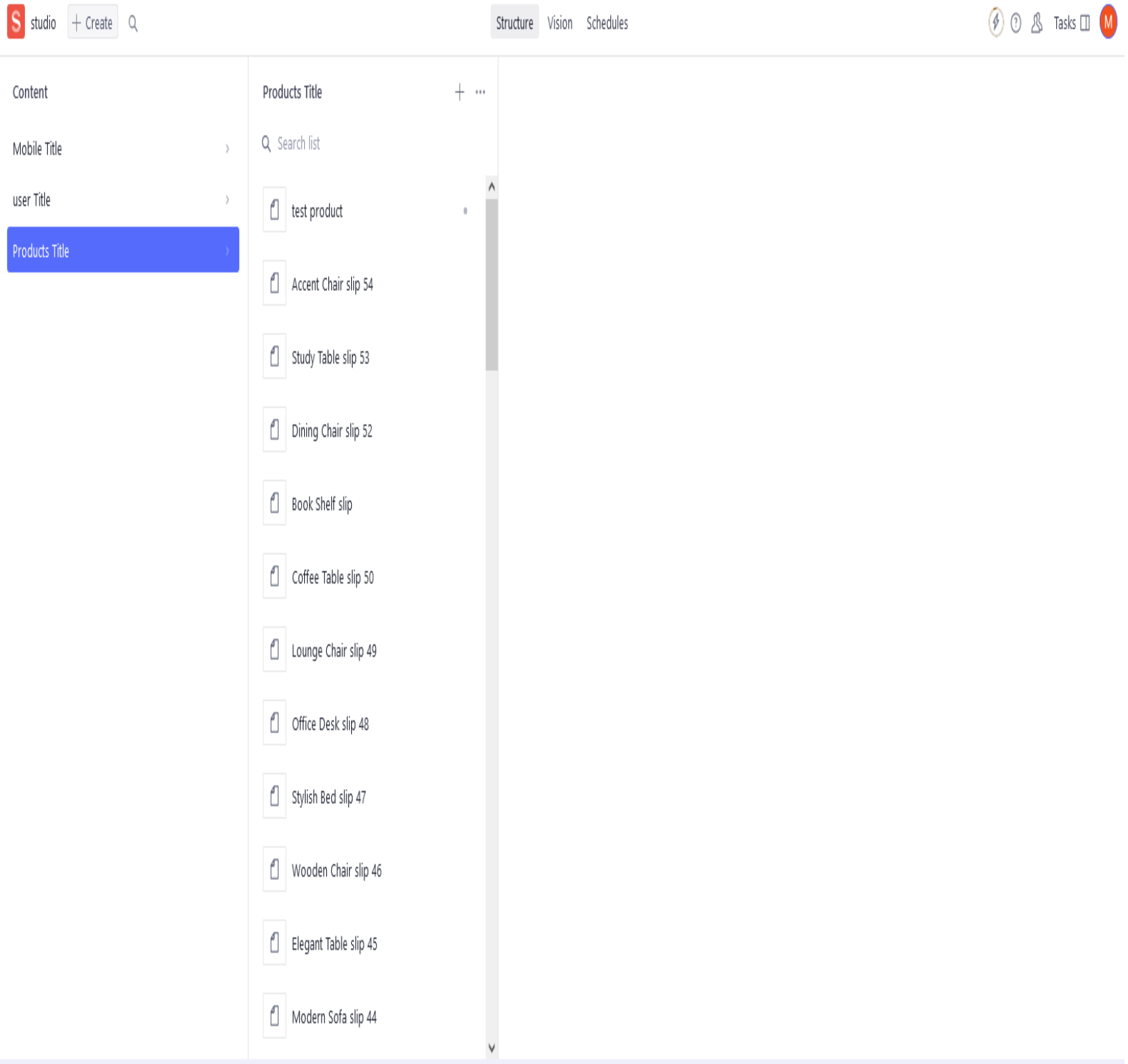
Sixth Product 6 652.3 $65

Seventh Product 7895.2 $65

Eight Product 8 5468.6 $65

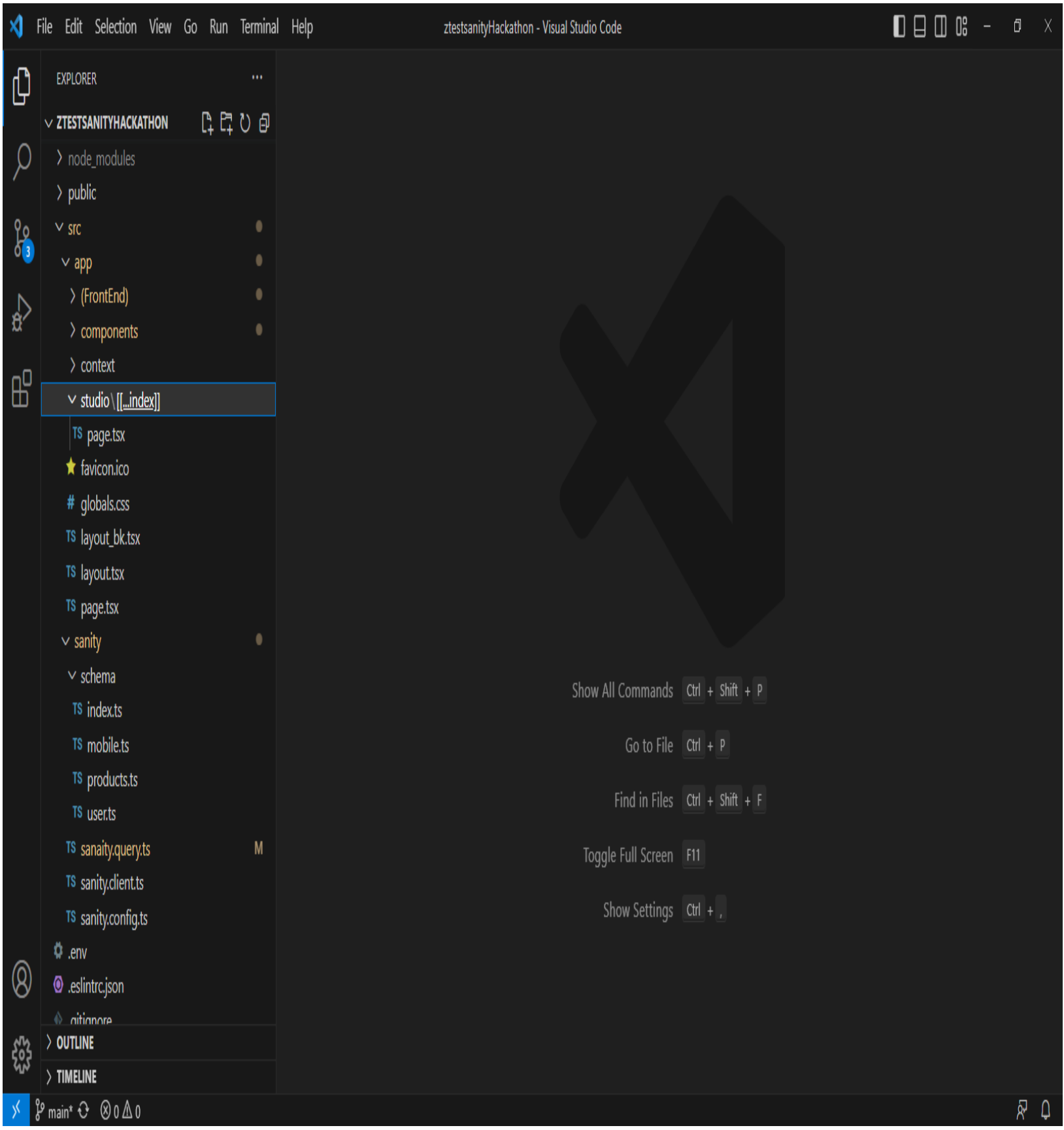
Ninth Product 9 159.2 $65

Tenth Product 10 456.6 $65

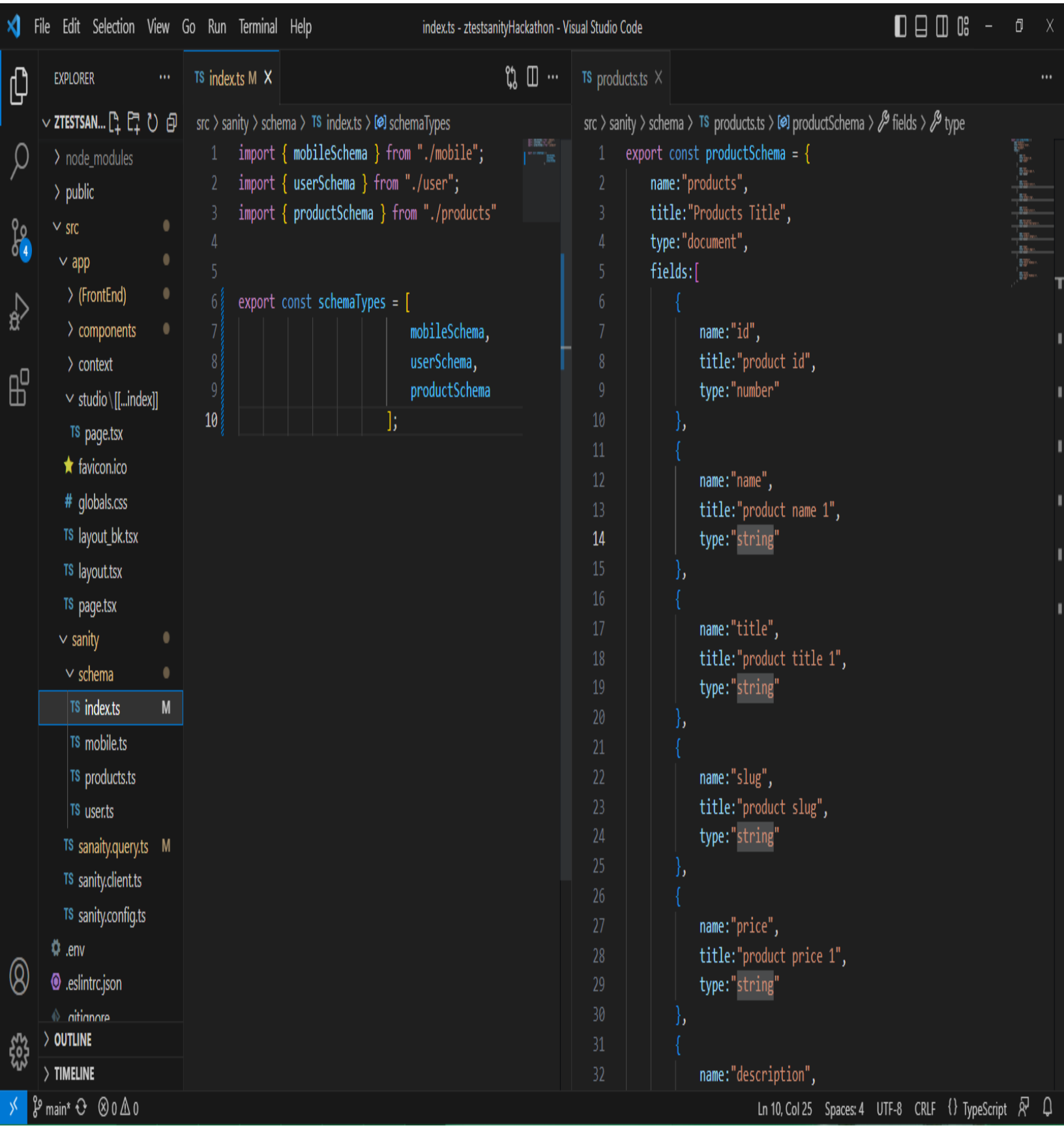


3. Populated Sanity CMS fields

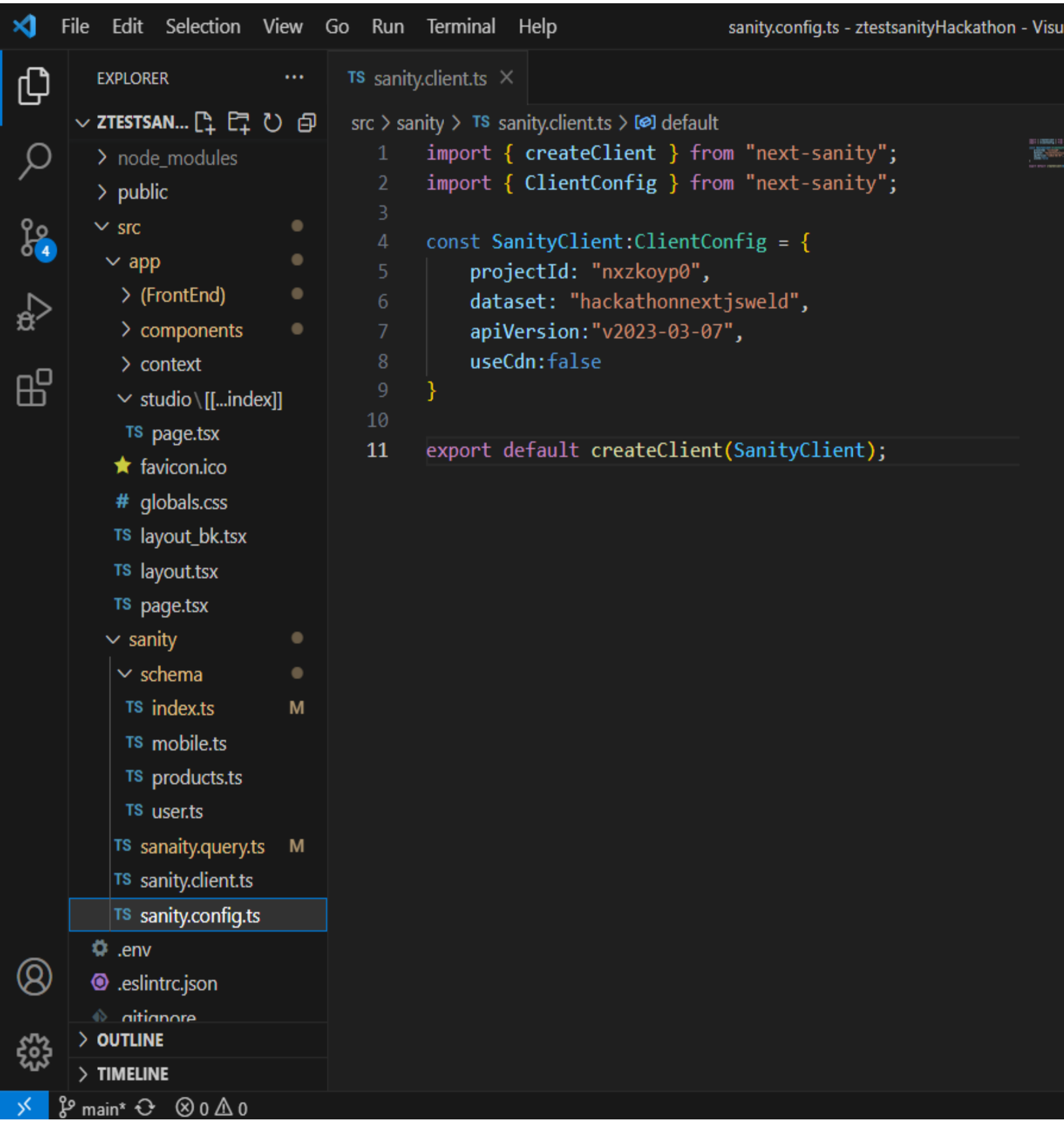
4. Code snippets for API integration and migration scripts


File Edit Selection View Go Run Terminal Help
ztestsanityHackathon - Visual Studio Code
EXPLORER
ZTESTSANITYHACKATHON
node_modules
public
src
app
(FrontEnd)
components
context
studio\[[...index]]
page.tsx
favicon.ico
globals.css
layout_bk.tsx
layout.tsx
page.tsx
sanity
schema
index.ts
mobile.ts
products.ts
user.ts
sanaity.query.ts
M
sanity.client.ts
sanity.config.ts
.env
.eslintrc.json
.gitignore
OUTLINE
TIMELINE
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,
main*
0 0

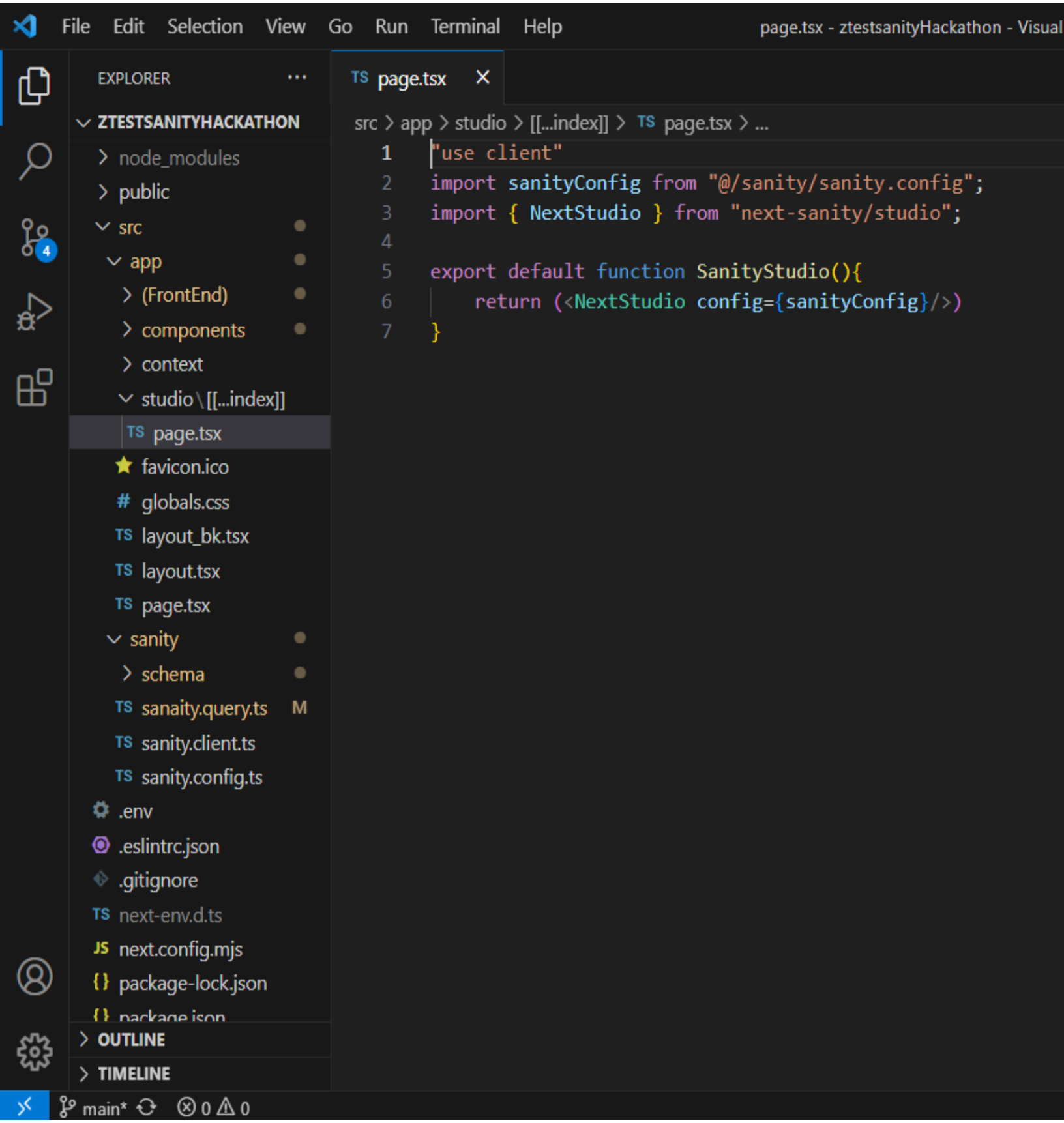
Schema/index.ts and products.ts


File Edit Selection View Go Run Terminal Help
index.ts - ztestsanityHackathon - Visual Studio Code
EXPLORER
ZTESTSAN...
node_modules
public
src
app
(FrontEnd)
components
context
studio \ [[...index]]
page.tsx
favicon.ico
globals.css
layout_bk.tsx
layout.tsx
page.tsx
sanity
schema
index.ts M
mobile.ts
products.ts
user.ts
sanaity.query.ts M
sanity.client.ts
sanity.config.ts
.env
.eslintrc.json
.gitignore
OUTLINE
TIMELINE
index.ts M X
src > sanity > schema > index.ts > schemaTypes
import { mobileSchema } from "./mobile";
import { userSchema } from "./user";
import { productSchema } from "./products"
export const schemaTypes = [
mobileSchema,
userSchema,
productSchema
];
products.ts X
src > sanity > schema > products.ts > productSchema > fields > type
export const productSchema = {
name:"products",
title:"Products Title",
type:"document",
fields:[
{
name:"id",
title:"product id",
type:"number"
},
{
name:"name",
title:"product name 1",
type:"string"
},
{
name:"title",
title:"product title 1",
type:"string"
},
{
name:"slug",
title:"product slug",
type:"string"
},
{
name:"price",
title:"product price 1",
type:"string"
},
{
name:"description",
main*
Ln 10, Col 25
Spaces: 4
UTF-8
CRLF
TypeScript

Sanity/sanity.client.ts and sanity/sanity.config.ts


File Edit Selection View Go Run Terminal Help
sanity.config.ts - ztestsanityHackathon - Visu
EXPLORER
ZTESTSAN...
node_modules
public
src
app
(FrontEnd)
components
context
studio \ [[...index]]
page.tsx
favicon.ico
globals.css
layout_bk.tsx
layout.tsx
page.tsx
sanity
schema
index.ts M
mobile.ts
products.ts
user.ts
sanaity.query.ts M
sanity.client.ts
sanity.config.ts
.env
.eslintrc.json
.gitignore
OUTLINE
TIMELINE
sanity.client.ts
src > sanity > sanity.client.ts > default
1 import { createClient } from "next-sanity";
2 import { ClientConfig } from "next-sanity";
3
4 const SanityClient:ClientConfig = {
5 projectId: "nxzkoyp0",
6 dataset: "hackathonnextjsweld",
7 apiVersion:"v2023-03-07",
8 useCdn:false
9 }
10
11 export default createClient(SanityClient);
main*
0 0

Sanity/[[…index]]/page.tsx


File Edit Selection View Go Run Terminal Help
page.tsx - ztestsanityHackathon - Visual
EXPLORER
ZTESTSANITYHACKATHON
node_modules
public
src
app
(FrontEnd)
components
context
studio\[[...index]]
page.tsx
favicon.ico
globals.css
layout_bk.tsx
layout.tsx
page.tsx
sanity
schema
sanaity.query.ts M
sanity.client.ts
sanity.config.ts
.env
.eslintrc.json
.gitignore
next-env.d.ts
next.config.mjs
package-lock.json
OUTLINE
TIMELINE
page.tsx
src > app > studio > [[...index]] > page.tsx > ...
1 "use client"
2 import sanityConfig from "@/sanity/sanity.config";
3 import { NextStudio } from "next-sanity/studio";
4
5 export default function SanityStudio(){
6 return (<NextStudio config={sanityConfig}/>)
7 }
main*
0 0

Day Checklist:

Self-Validation Checklist:

API Understanding: ✓

Schema Validation: ✓

Data Migration: ✓

API Integration in Next.js: ✓

Submission Preparation ✓

========== X X X X X X ==========