Day 3 - API Integration Report - Hekto

API Integration Process

Product Details API

Overview

The `Product Details API` is designed to fetch specific product information based on its slug. The fetched data is used to populate product pages on the frontend dynamically.

- **Endpoint:** `/api/products/:product`
- **Method:** `GET`
- **Purpose:** Retrieve detailed information about a specific product.

Integration Steps:

1. **Define API Endpoint:** The API was defined in the `GET` method under the route `/api/products/:product`.
2. **Construct Query:** Used the GROQ query to fetch product data from Sanity CMS for the `home`, `outdoor`, and `office` schemas.
3. **API Call:** The API fetches product details such as title, price, sale price, overview, images, and tags from Sanity CMS.
4. **Error Handling:**
    - 400: Invalid or missing slug parameter.
    - 404: Product not found.
    - 500: Internal server error.

Code Snippet:

```
import { client } from "@/sanity/lib/client";
import { groq } from "next-sanity";
import { NextResponse } from "next/server";

export async function GET(req: Request, { params }: { params: { product: string } }) {
  const { product: slug } = params;

  if (!slug) {
    return NextResponse.json({ error: "Invalid or missing slug parameter" }, { status: 400 });
  }

  const query = groq`*[_type in ["home", "outdoor", "office"] && slug.current == $slug][0]{
    title,
    price,
    salePrice,
    overview,
    images[] {
      asset->{
        _id,
        url
      },
      alt
    },
    slug,
    productDetails,
    tags,
    stock,
    subCategory
  }`;

  try {
    const product = await client.fetch(query, { slug });

    if (!product) {
      return NextResponse.json({ error: "Product not found" }, { status: 404 });
    }

    return NextResponse.json({ product }, { status: 200 });
  } catch (error) {
    console.error("Error fetching product data:", error);
    return NextResponse.json({ error: "Failed to fetch product data" }, { status: 500 });
  }
}
```

Blog Details API

Overview

The `Blog Details API` retrieves specific blog post information based on the slug provided. The data is displayed on the blog detail pages.

- **Endpoint:** `/api/blogs/:slug`
- **Method:** `GET`
- **Purpose:** Retrieve detailed information about a specific blog post.

Integration Steps:

1. **Define API Endpoint:** The API was defined in the `GET` method under the route `/api/blogs/:slug`.
2. **Construct Query:** Used the GROQ query to fetch blog data such as content, images, tags, and comments from Sanity CMS.
3. **Error Handling:**
   - 404: Blog not found.
   - 500: Internal server error.

Code Snippet:

```
import { NextResponse } from "next/server";
import { client } from "@/sanity/lib/client";
import { groq } from "next-sanity";

export async function GET(req: Request, { params }: { params: { slug: string } }) {
  const { slug } = params;

  const query = groq`*[_type == "blog" && slug.current == $slug][0]{
    slug,
    title,
    overview,
    content[] {
      ...,
      _type == "image" => {
        asset-> {
          url
        },
        alt
      },
      _type == "blockquote" => {
        quote,
        author
      }
    },
    publishingDate,
    authorName,
    "authorImage": authorImage.asset->url,
    "featuredImage": featuredImage.asset->url,
    tags,
    comments[] {
      name,
      comment,
      "pic": pic.asset->url,
      postedAt
    }
  }`;

  try {
    const blog = await client.fetch(query, { slug });

    if (!blog) {
      return NextResponse.json({ error: "Blog not found." }, { status: 404 });
    }

    return NextResponse.json(blog);
  } catch (error) {
    console.error("Error fetching blog:", error);
    return NextResponse.json({ error: "Failed to fetch blog." }, { status: 500 });
  }
}
```

Adjustments Made to Schemas

- **Added Slug Field:** To ensure unique identification of products and blogs.
- **Enhanced Product Schema:** Included fields for `salePrice`, `tags`, and `subCategory` for better categorization and promotional offers.
- **Enhanced Blog Schema:** Added `comments` and `content` fields to store detailed information, including rich text and images.

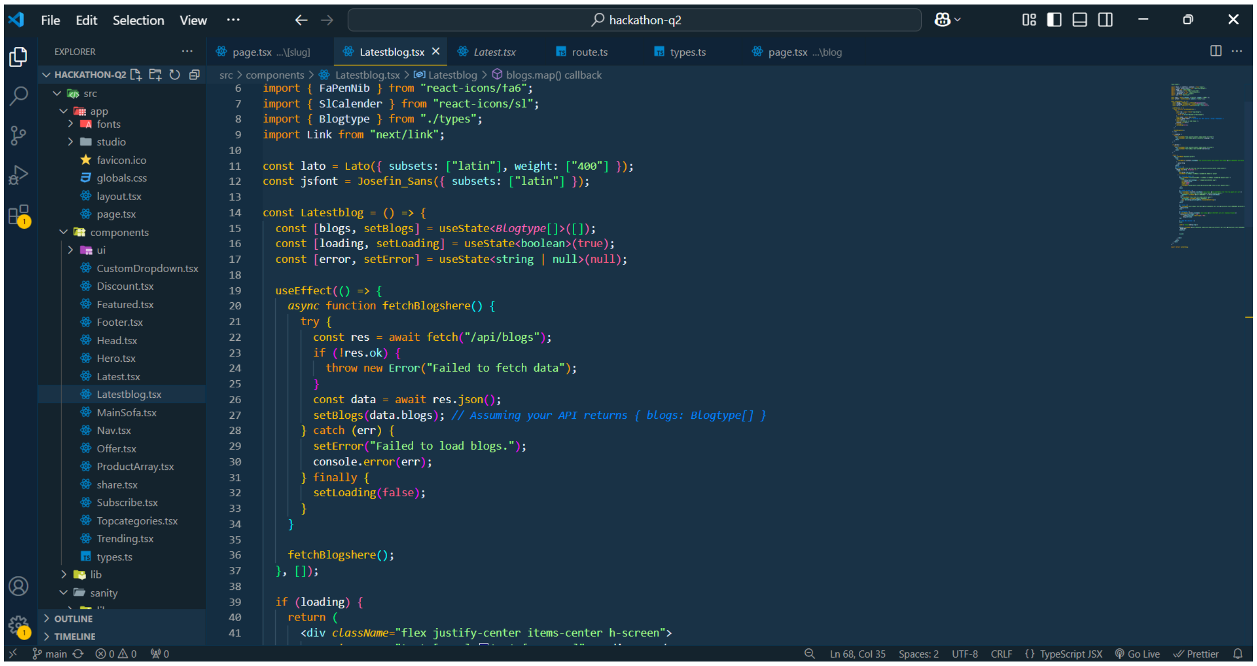
Migration Steps and Tools Used

1. **Migration Tool:** Used Sanity CLI for exporting and importing data during schema updates.
2. **Steps:**
    - Exported existing data using the `sanity dataset export` command.
    - Modified schemas in the `schemas` folder.
    - Reimported updated data using the `sanity dataset import` command.
3. **Testing:** Verified the APIs against both existing and updated schema data.

Screenshots

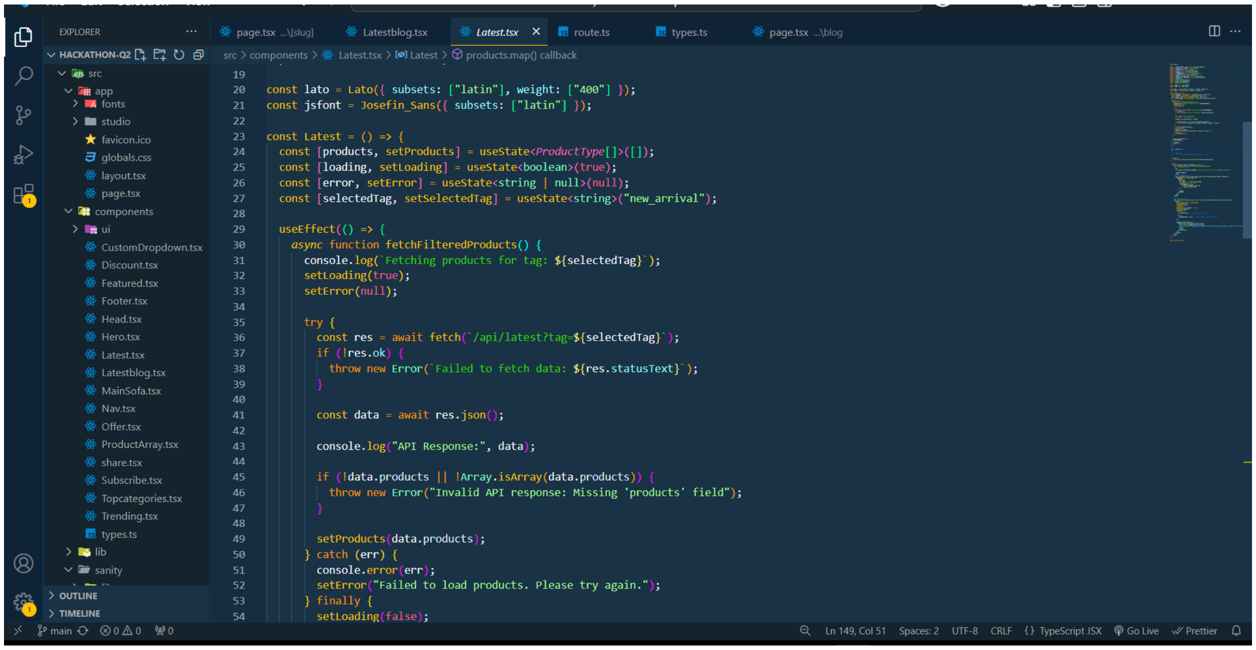
1. **API Calls:** (Attach screenshots of Postman or browser console showing API responses.)

Api Call for blogs :



```tsx
import { FaPenNib } from "react-icons/fa6";
import { SlCalender } from "react-icons/sl";
import { Blogtype } from "./types";
import Link from "next/link";

const lato = Lato({ subsets: ["latin"], weight: ["400"] });
const jsfont = Josefin_Sans({ subsets: ["latin"] });

const Latestblog = () => {
  const [blogs, setBlogs] = useState<Blogtype[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    async function fetchBlogshere() {
      try {
        const res = await fetch("/api/blogs");
        if (!res.ok) {
          throw new Error("Failed to fetch data");
        }
        const data = await res.json();
        setBlogs(data.blogs); // Assuming your API returns { blogs: Blogtype[] }
      } catch (err) {
        setError("Failed to load blogs.");
        console.error(err);
      } finally {
        setLoading(false);
      }
    }

    fetchBlogshere();
  }, []);

  if (loading) {
    return (
      <div className="flex justify-center items-center h-screen">
```

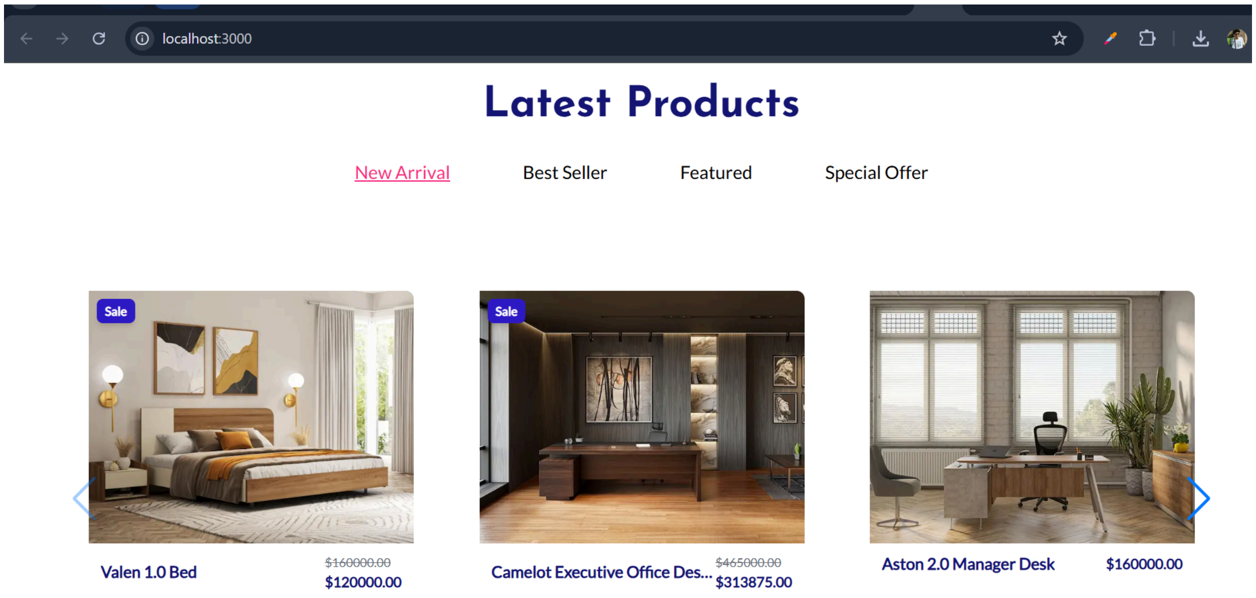
API call for Products(Selected products):



```tsx

const lato = Lato({ subsets: ["latin"], weight: ["400"] });
const jsfont = Josefin_Sans({ subsets: ["latin"] });

const Latest = () => {
  const [products, setProducts] = useState<ProductType[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);
  const [selectedTag, setSelectedTag] = useState<string>("new_arrival");

  useEffect(() => {
    async function fetchFilteredProducts() {
      console.log(`Fetching products for tag: ${selectedTag}`);
      setLoading(true);
      setError(null);

      try {
        const res = await fetch(`/api/latest?tag=${selectedTag}`);
        if (!res.ok) {
          throw new Error(`Failed to fetch data: ${res.statusText}`);
        }

        const data = await res.json();

        console.log("API Response:", data);

        if (!data.products || !Array.isArray(data.products)) {
          throw new Error("Invalid API response: Missing 'products' field");
        }

        setProducts(data.products);
      } catch (err) {
        console.error(err);
        setError("Failed to load products. Please try again.");
      } finally {
        setLoading(false);
```

2. **Frontend Display:** (Attach screenshots showing product and blog data displayed on the frontend.)

Products on the Website:



Latest Products

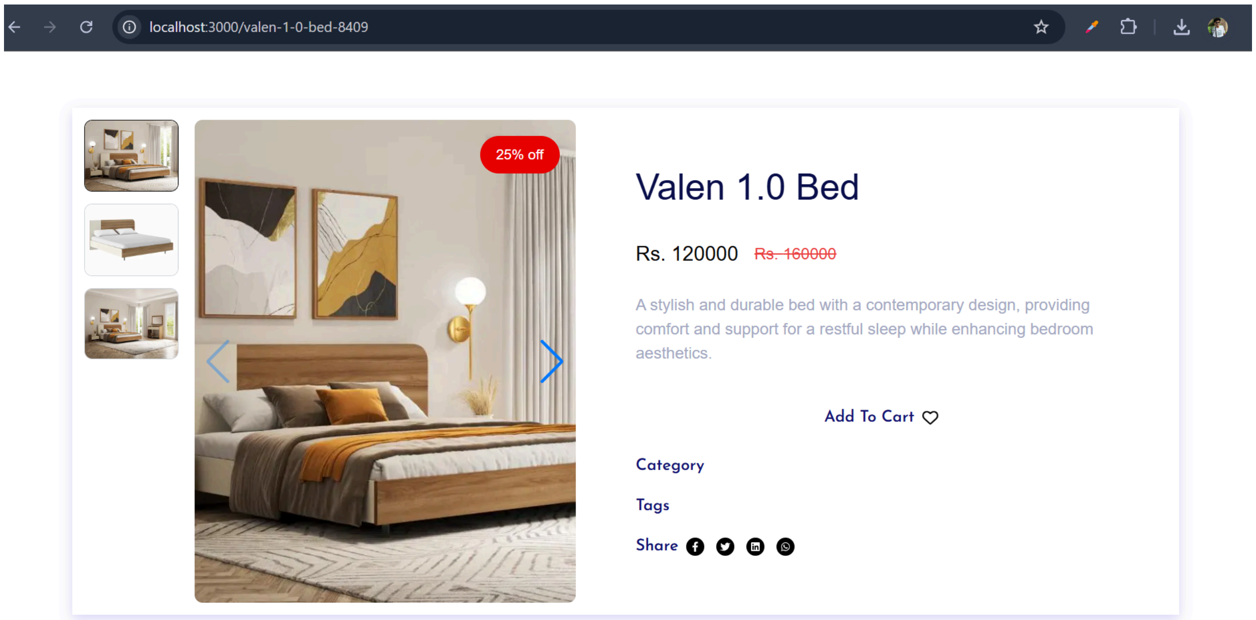
New Arrival | Best Seller | Featured | Special Offer

Valen 1.0 Bed ~~$160000.00~~ $120000.00

Camelot Executive Office Des... ~~$465000.00~~ $313875.00

Aston 2.0 Manager Desk $160000.00

Product detail on website:



Valen 1.0 Bed

Rs. 120000 ~~Rs. 160000~~

A stylish and durable bed with a contemporary design, providing comfort and support for a restful sleep while enhancing bedroom aesthetics.

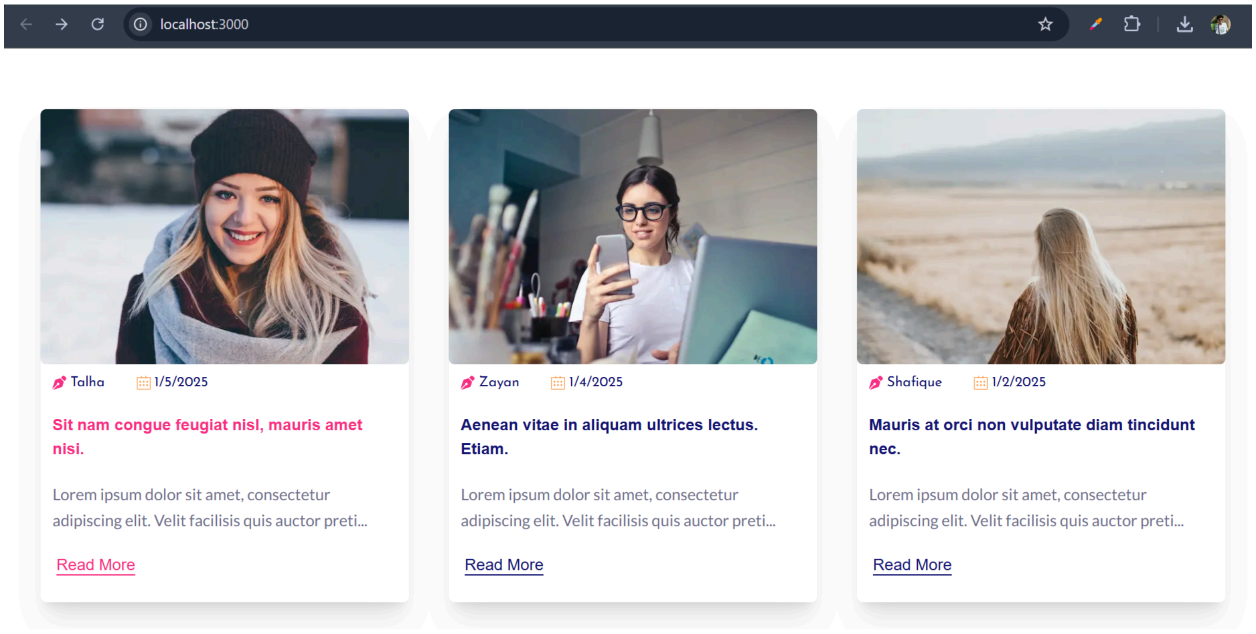
Add To Cart

Category

Tags

Share

Blogs on the Website:



Talha 1/5/2025

Sit nam congue feugiat nisl, mauris amet nisi.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit facilisis quis auctor preti...

Read More

Zayan 1/4/2025

Aenean vitae in aliquam ultrices lectus. Etiam.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit facilisis quis auctor preti...

Read More

Shafique 1/2/2025

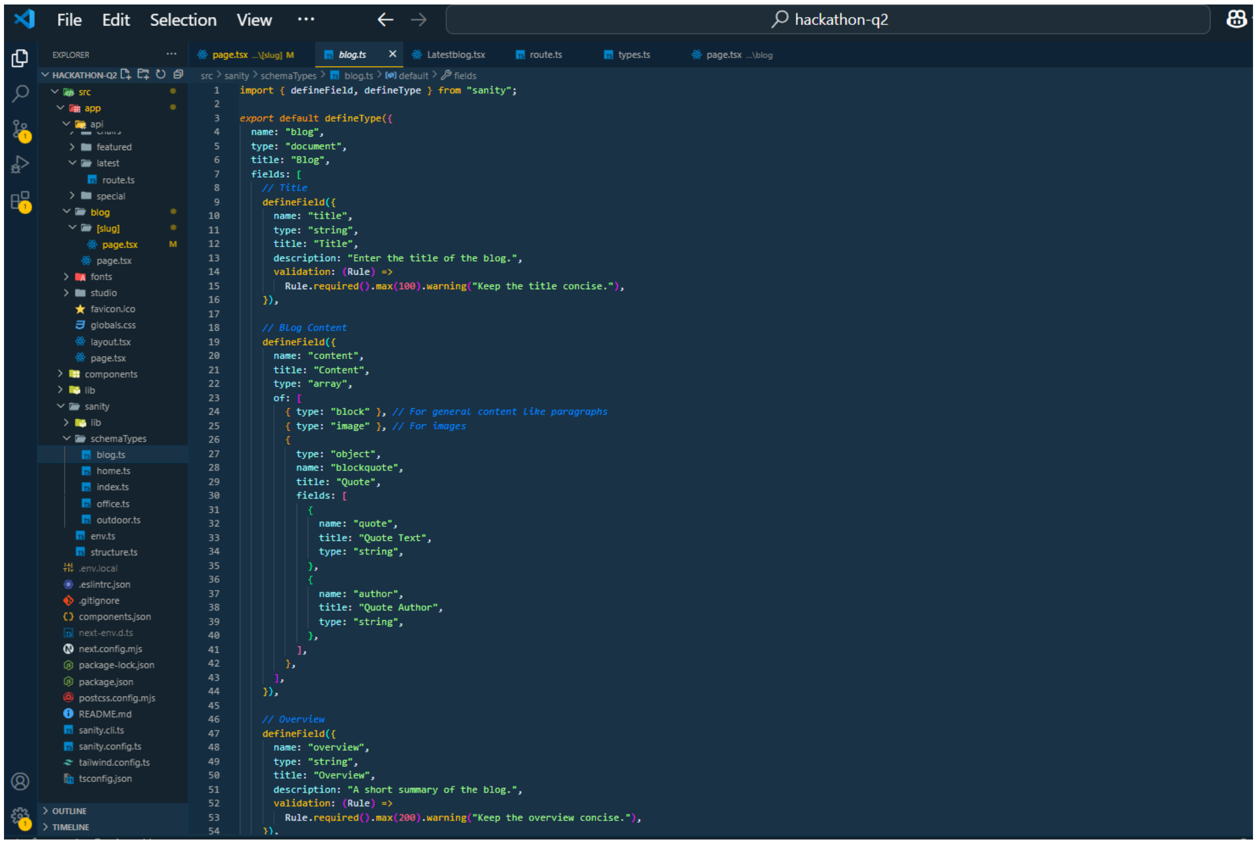
Mauris at orci non vulputate diam tincidunt nec.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit facilisis quis auctor preti...

Read More

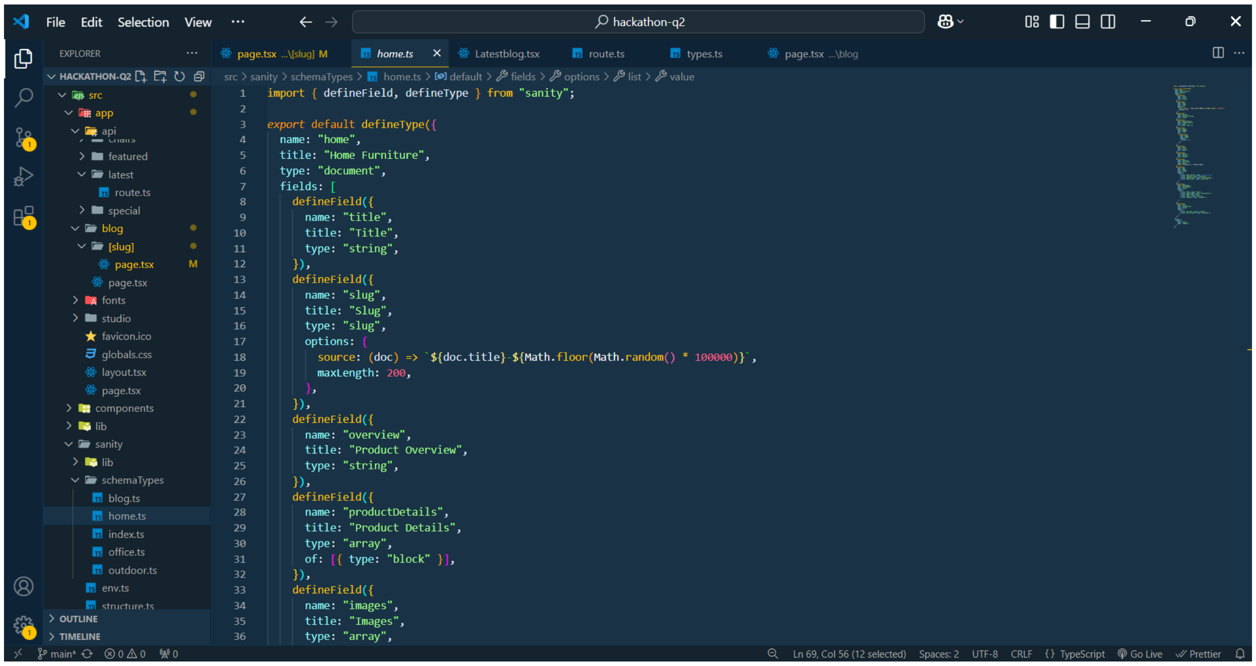
3. **Sanity CMS Fields:** (Attach screenshots of populated Sanity CMS fields, such as `title`, `overview`, and `tags`.)

Schema for blog:



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

export default defineType({
  name: "blog",
  type: "document",
  title: "Blog",
  fields: [
    // Title
    defineField({
      name: "title",
      type: "string",
      title: "Title",
      description: "Enter the title of the blog.",
      validation: (Rule) =>
        Rule.required().max(100).warning("Keep the title concise."),
    }),

    // Blog Content
    defineField({
      name: "content",
      title: "Content",
      type: "array",
      of: [
        { type: "block" }, // For general content like paragraphs
        { type: "image" }, // For images
        {
          type: "object",
          name: "blockquote",
          title: "Quote",
          fields: [
            {
              name: "quote",
              title: "Quote Text",
              type: "string",
            },
            {
              name: "author",
              title: "Quote Author",
              type: "string",
            },
          ],
        },
      ],
    }),

    // Overview
    defineField({
      name: "overview",
      type: "string",
      title: "Overview",
      description: "A short summary of the blog.",
      validation: (Rule) =>
        Rule.required().max(200).warning("Keep the overview concise."),
    }),
```

Schema for home:



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

export default defineType({
  name: "home",
  title: "Home Furniture",
  type: "document",
  fields: [
    defineField({
      name: "title",
      title: "Title",
      type: "string",
    }),
    defineField({
      name: "slug",
      title: "Slug",
      type: "slug",
      options: {
        source: (doc) => `${doc.title}-${Math.floor(Math.random() * 100000)}`,
        maxLength: 200,
      },
    }),
    defineField({
      name: "overview",
      title: "Product Overview",
      type: "string",
    }),
    defineField({
      name: "productDetails",
      title: "Product Details",
      type: "array",
      of: [{ type: "block" }],
    }),
    defineField({
      name: "images",
      title: "Images",
      type: "array",
```