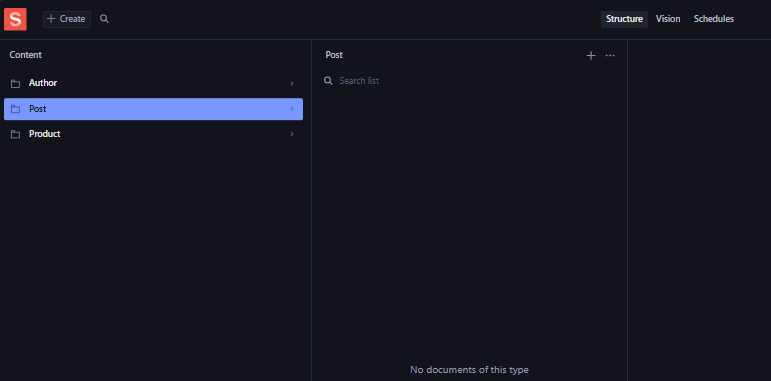
**DAY 3 - API INTEGRATION AND DATA MIGRATION**

**Step 01**

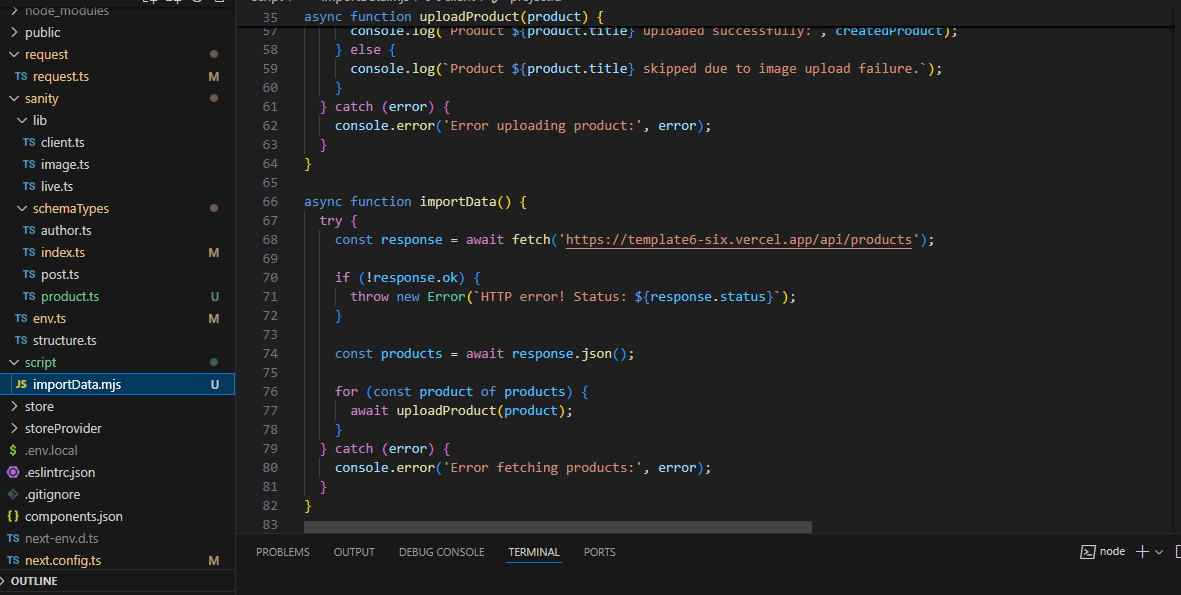
Sanity is a headless CMS (Content Management System) designed to manage structured content efficiently. It provides developers with a flexible backend to create, update, and manage content, which can be accessed through APIs for use in websites or applications. I have created a project in Sanity named **"Product"**, where all the configurations and schemas will revolve around managing product-related data. This setup will allow me to define custom fields, such as title, description, price, and images, and organize the content for seamless integration with a frontend or e-commerce platform.

**Step 02**

Next, I utilized an API URL provided by my instructor. This API serves as the data source for fetching product-related information. By integrating this API into my project, I can retrieve dynamic data such as product names, descriptions, prices, and other details, which can then be displayed or processed in the application. This step is crucial for connecting the backend content or external services with the frontend to create a seamless and functional user experience.

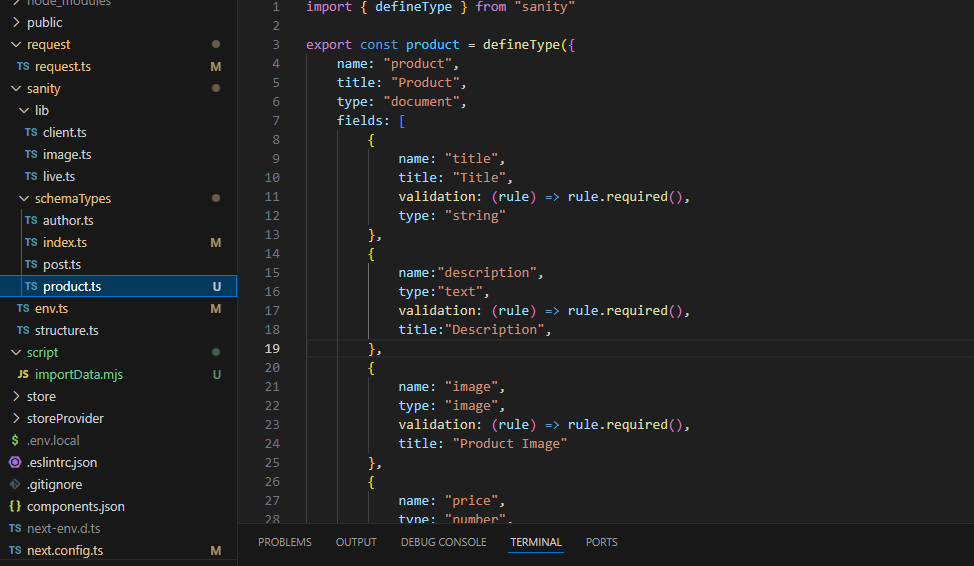


**Step 03**



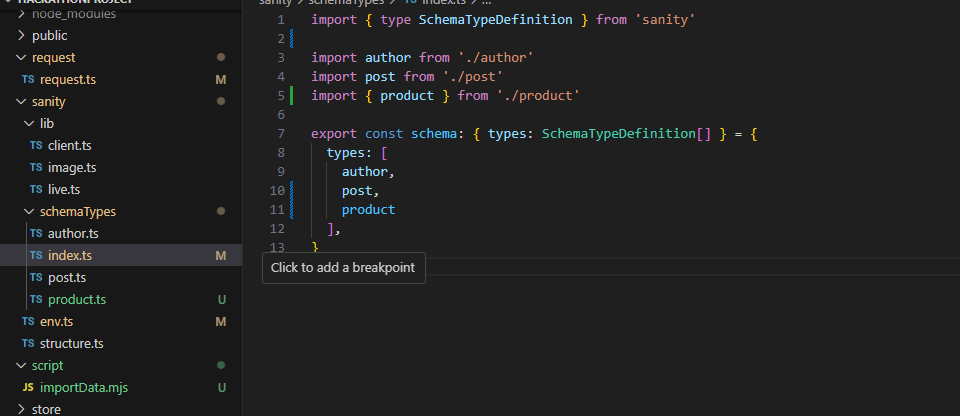
Next, I created a file named **importData.mjs** where I pasted the migration script. This script is responsible for transferring or importing data into the Sanity project. In this file, I included the API URL provided by my instructor to fetch the data from the external source. The migration script ensures that the product-related information, such as names, descriptions, prices, and images, is properly imported and synchronized into the Sanity project’s dataset. This allows the data to be accessible and managed within the Sanity Studio for further usage in the application.

**Step 04**



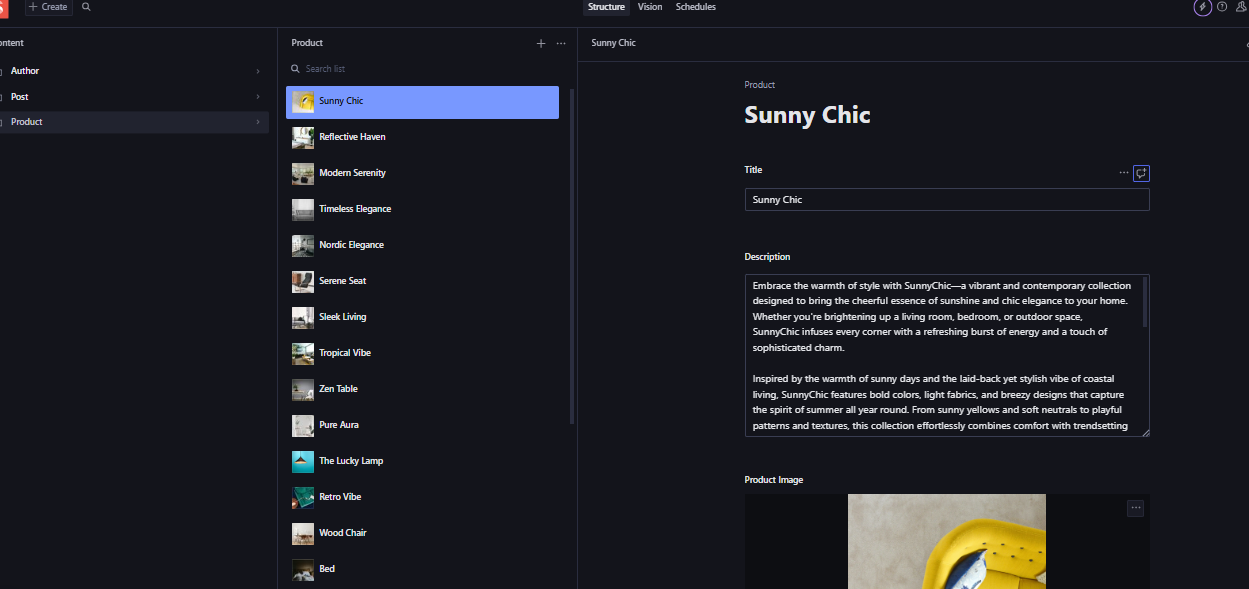
Next, I navigated to the Sanity project folder and within the **schemas** folder, I created a file named **product.ts**. In this file, I defined the schema for the product document. The schema outlines the structure of the product data, specifying fields such as title, description, price, and image. This schema ensures that the content for each product is properly structured and validated within the Sanity CMS. By defining this schema, I set up a template for how product information will be managed and stored in the database, making it easier to retrieve and display within the application.

**Step 05**



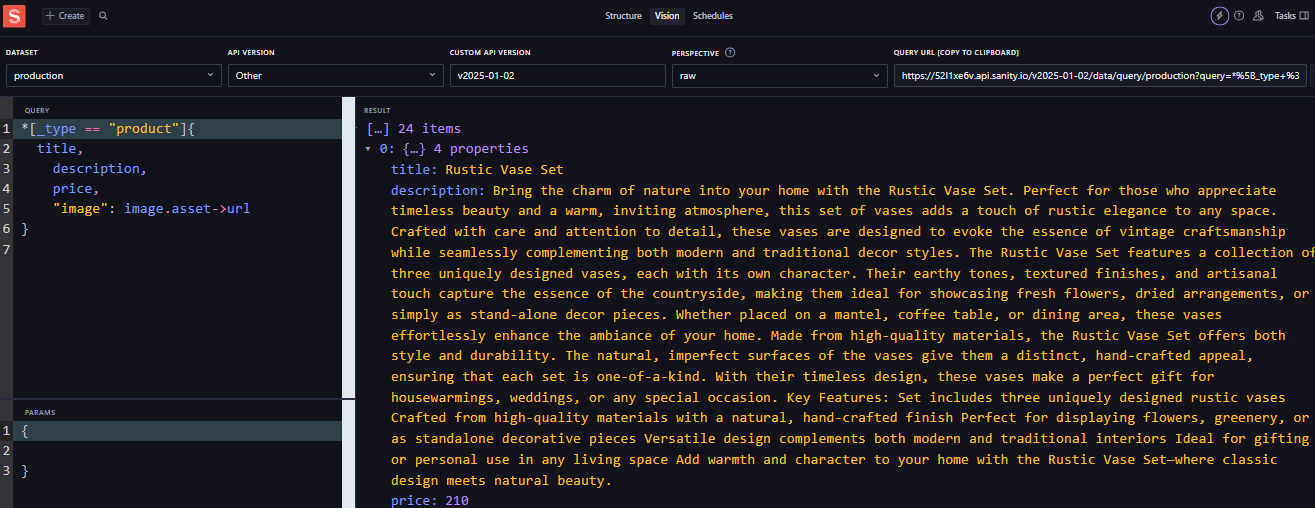
Next, I added the **product schema** to the **index.ts** file in the Sanity schema folder. This step involves importing the product schema into the index.ts file and then including it in the array of schemas that Sanity uses to manage and display content. By doing this, I ensure that the product schema is registered within the Sanity project, allowing it to be available in the Sanity Studio for content creation and management. This integration makes it possible to use the defined product schema throughout the project for managing product data efficiently.

**Step 06**

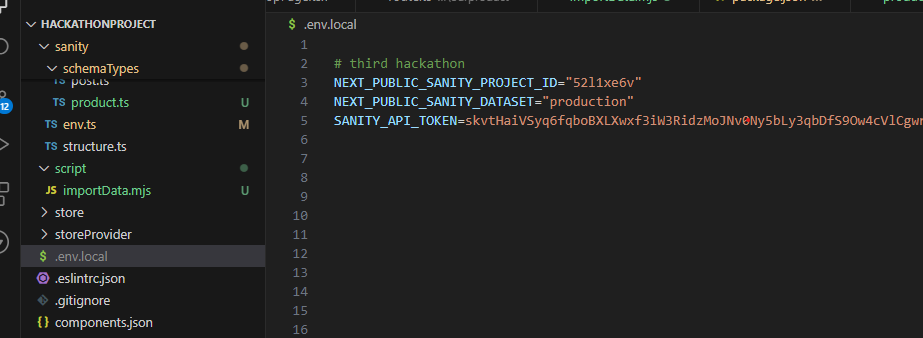


Finally, I navigated to **localhost:3000/studio**, which is the local development server for the Sanity Studio. Once there, I was able to see that all the data had been successfully fetched and imported into Sanity. The migration script had properly transferred the product information, and now it was available in the Sanity Studio for management. The products, along with their details (such as title, description, price, and image), were visible and ready to be edited or used within the application. This confirms that the integration between the API, migration script, and Sanity CMS was successful.

**Step 07**



Next, I navigated to the **Vision** section within the Sanity Studio. Vision is a powerful tool in Sanity that allows you to query and view the data in real-time. In the **Vision** interface, I defined a query for the **fields** of the product schema. This query allows me to view and filter the product data based on specific fields like title, description, price, and image. It provides a way to inspect the data structure and ensure that all fields are properly populated and accessible. By using Vision, I was able to verify that the product data was correctly fetched and organized as expected within the Sanity Studio.

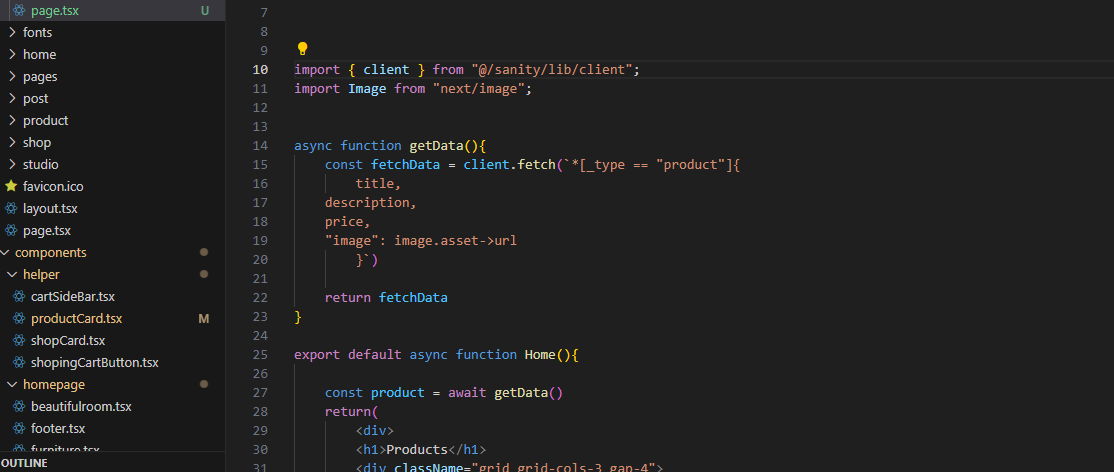
**Step 08**

Next, I navigated to the **.env.local** file and defined the necessary environment variables for the Sanity project. In this file, I added the following key details:

* **SANITY\_PROJECT\_ID**: The unique project ID for the Sanity project, which links the application to the correct dataset.
* **SANITY\_DATASET**: The dataset name, usually "production," which specifies where the data is stored.
* **SANITY\_TOKEN**: The authentication token, which provides secure access to the Sanity API for reading and writing data.

By defining these values in the .env.local file, I ensured that the application could connect to the Sanity backend securely and fetch or manipulate the product data as needed. This step is essential for maintaining the project's configuration and establishing the connection to the Sanity CMS.

**Step 09**



Next, I created a page where I needed to fetch the data from the API. To do this, I used the **fetch** function to make a request to the API and retrieve the necessary product data. By using **fetch**, I was able to send a request to the API endpoint that I had previously defined in the **.env.local** file and get the response containing the product information. This function allowed me to asynchronously fetch data and then display or process it on the page. It ensures that the product details, such as titles, descriptions, and images, are dynamically loaded and rendered from the backend into the frontend of the application.

Bottom of Form

**Step 10**

