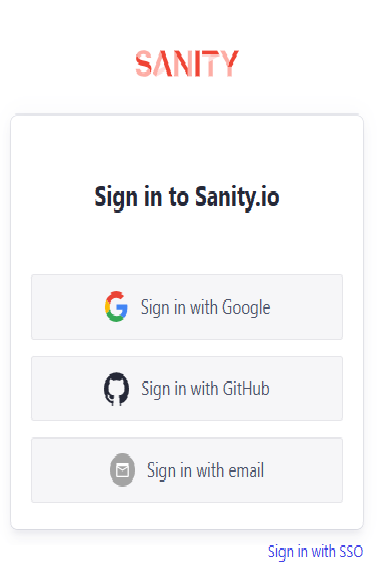
# Roll No. 407978

# Day 3 - API Integration Report - Hackathon-8 (Comforty)

## Step 01:

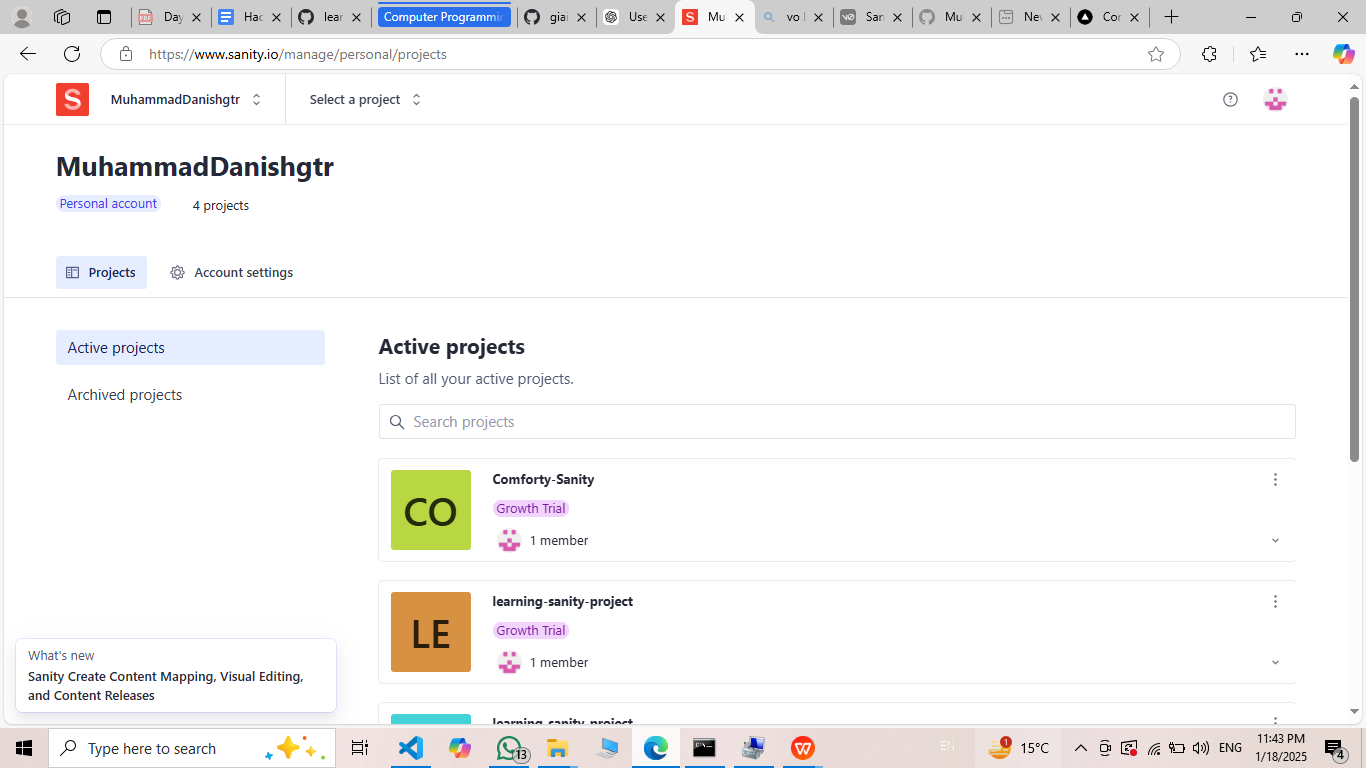
Make Account on sanity.io and sign-in through git-hub verification.



Sanity is a headless CMS (Content Management System) designed to manage structured content efficiently. It provides developers with a flexible back end to create, update, and manage content, which can be accessed through APIs for use in websites or applications.

**Step 02:**

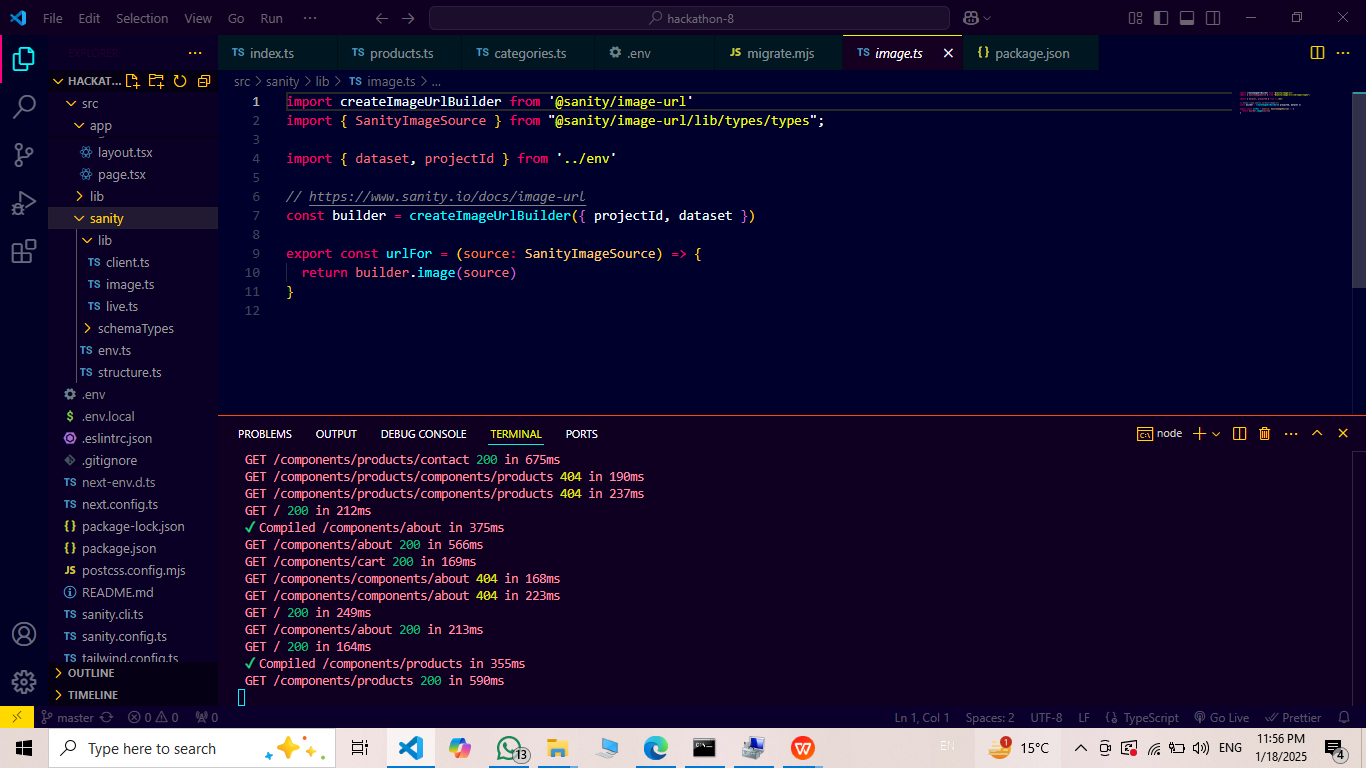
Make a project in sanity named Comforty-Sanity.



## Step 03:

Integrate sanity studio with my nextjs project named HACKATHON-8 by running the following command: npm create sanity@latest -- --template clean --create-project "Comforty-Sanity" --dataset production

This command make a sanity setup in my nextjs project and download multiple files and folder you can see in below image.



## Step 04:

Inside sanity folder there is a sub folder named schema type which is made automatic during sanity integration and Create two new files and add the following code:

1)products.ts

[/src/sanity/schemaTypes/products.ts](https://github.com/Hamzah-syed/giaic-hackathon-template-08/blob/master/src/sanity/schemaTypes/products.ts)

2)categories.ts

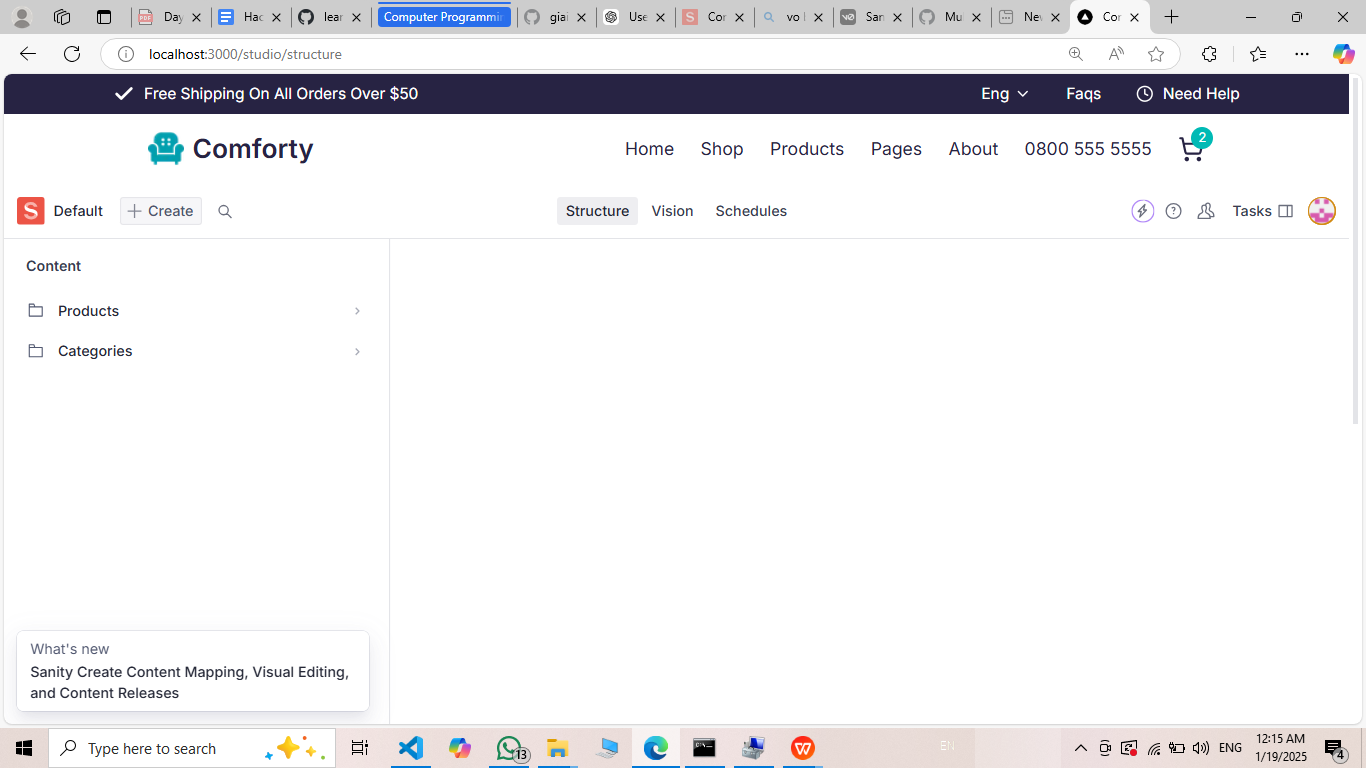
[/src/sanity/schemaTypes/categories.ts](https://github.com/Hamzah-syed/giaic-hackathon-template-08/blob/master/src/sanity/schemaTypes/categories.ts)

#### And then Importing above Schemas in index.ts

#### [src/sanity/schemaTypes/index.ts](https://github.com/Hamzah-syed/giaic-hackathon-template-08/blob/master/src/sanity/schemaTypes/index.ts)

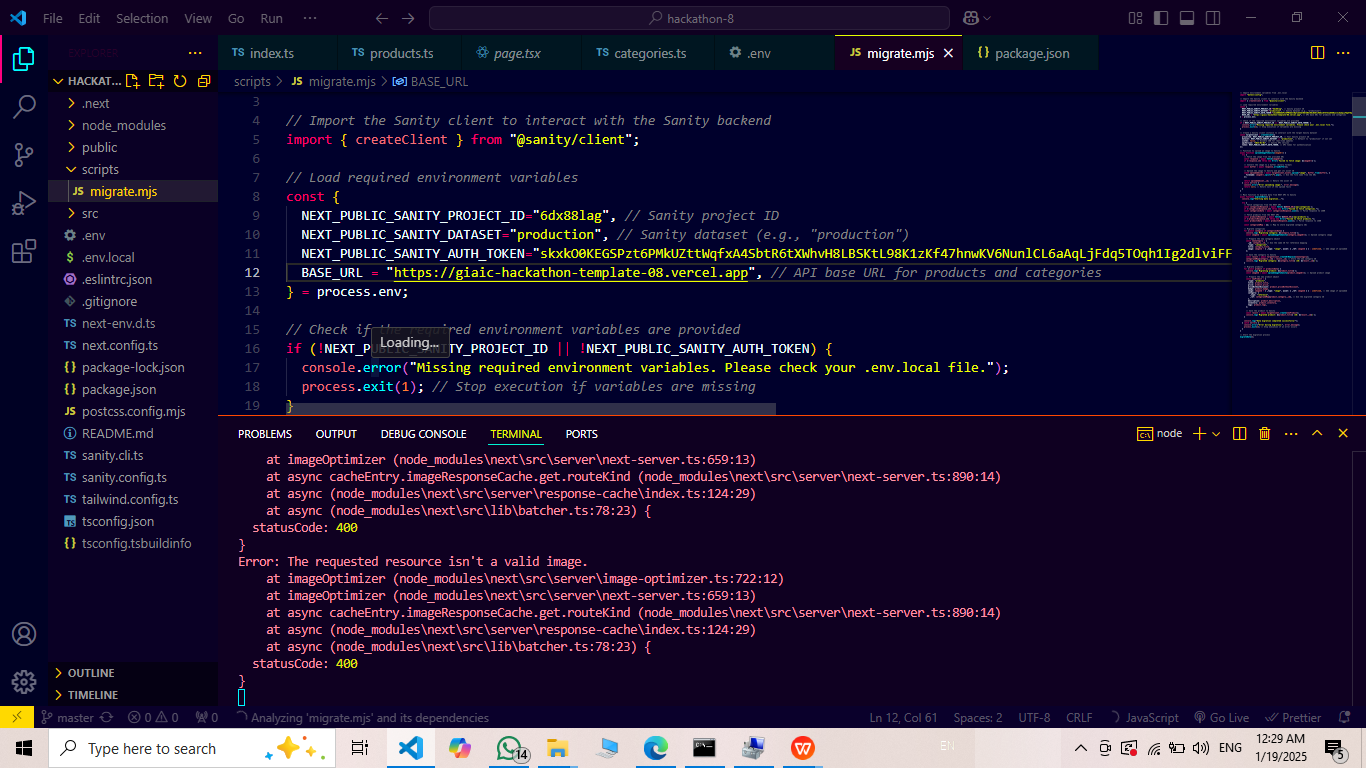
**Step 05:**

Run the command in cmd “npm run dev” and when server run in browser local host:3000 I add /studio in url to access my studio :



**Step 06:**

Next, I created a file named **migrate.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 07:**

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

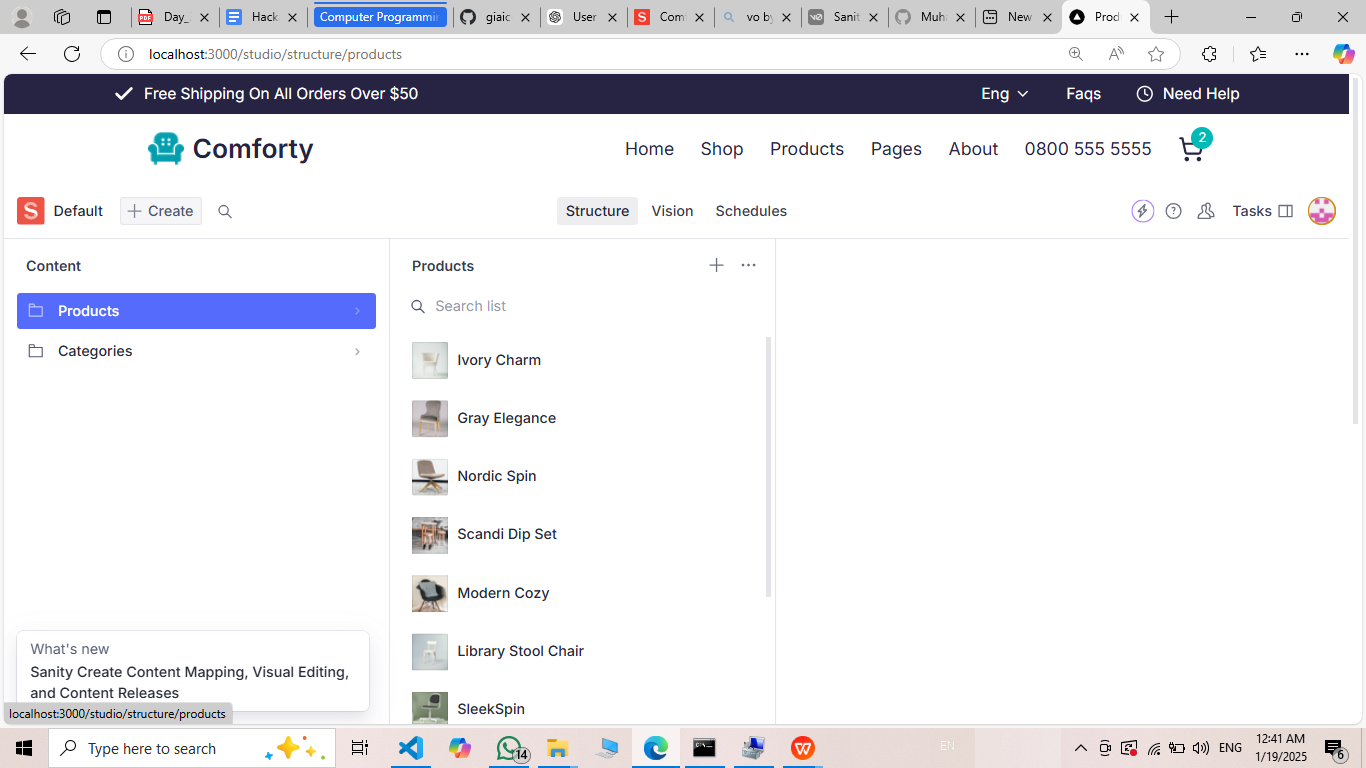
• **SANITY\_PROJECT\_ID**: The unique project ID for the Sanity project, which links the application to the correct dataset.

• **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.

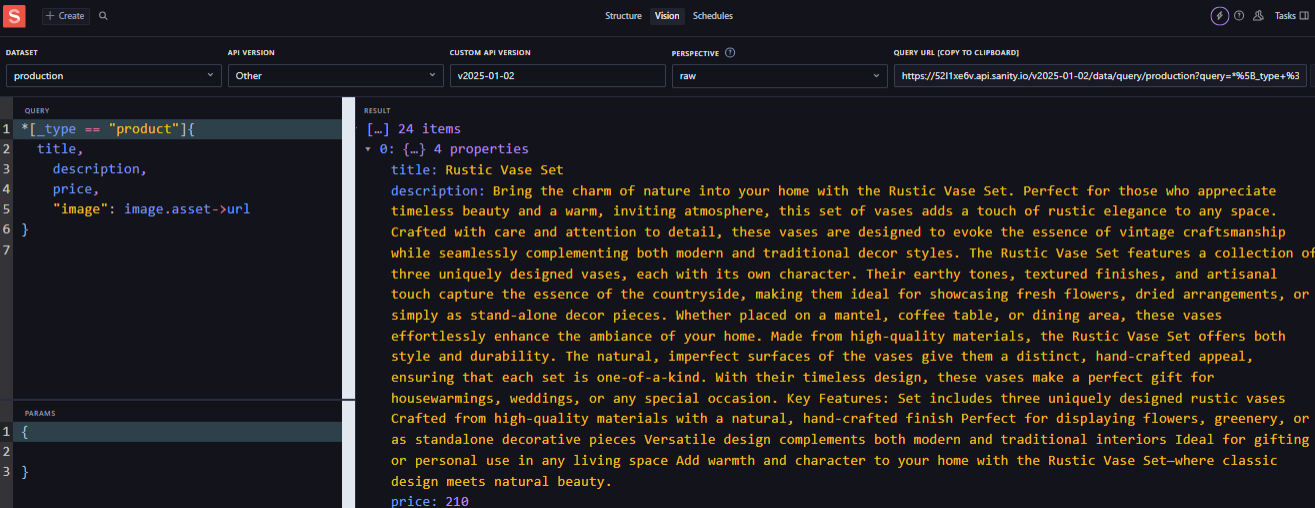
**Step 08:**

Finally run the command in CMD of my project “npm run migrate” it push all my data to Sanity studio:



**Step 09:**

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 10:**

Next, I created a page where I needed to fetch the data from the API. To do this, I used the **fetch f**unction 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.** 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.