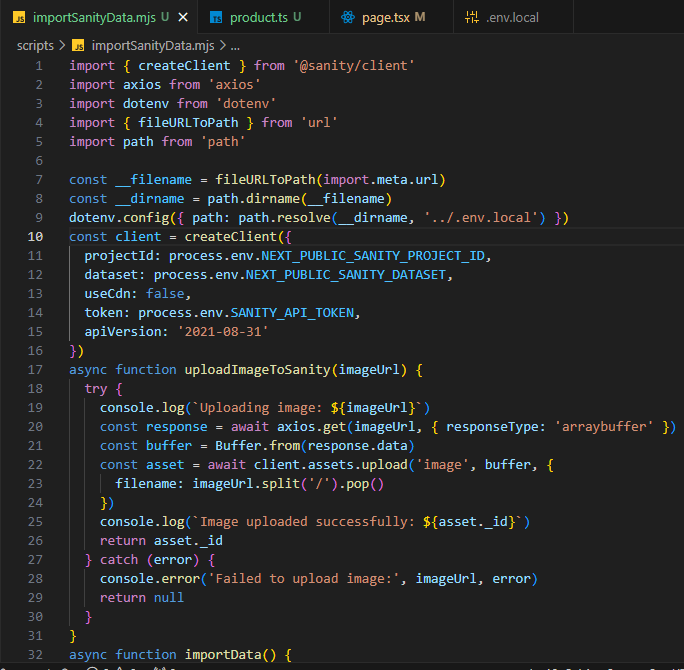
# **Day 3 - API Integration Report – Pharmaceutical Q-Commerce**

This report covers Day 3 of the Pharmaceutical Q-Commerce project, focusing on

1. **Custom Migration:**
   * Migrated data from the old system to the new one.
2. **Data Integration:**
   * Integrated pharmaceutical product data via an external API.
3. **Schema Creation:**
   * Created a schema to structure the API data.
4. **Displaying Data in Next.js:**
   * Used GROQ queries to fetch and display product data in the Next.js app.

# **Migration Code:**

1. **Insert Data into Sanity:**
   * Fetch product data from the API.
   * Map the data to match the Sanity schema.
   * Insert the data into Sanity CMS.
2. **Fetch Data from Sanity for Page:**
   * Use GROQ queries to fetch the inserted product data from Sanity.
   * Display the data on your Q-Commerce page.



# **Page Code**

This is the client-side code for rendering the Pharmaceutical Q-Commerce data in a Next.js page. Here’s how the code works:



• **GROQ Query to Fetch Data:**

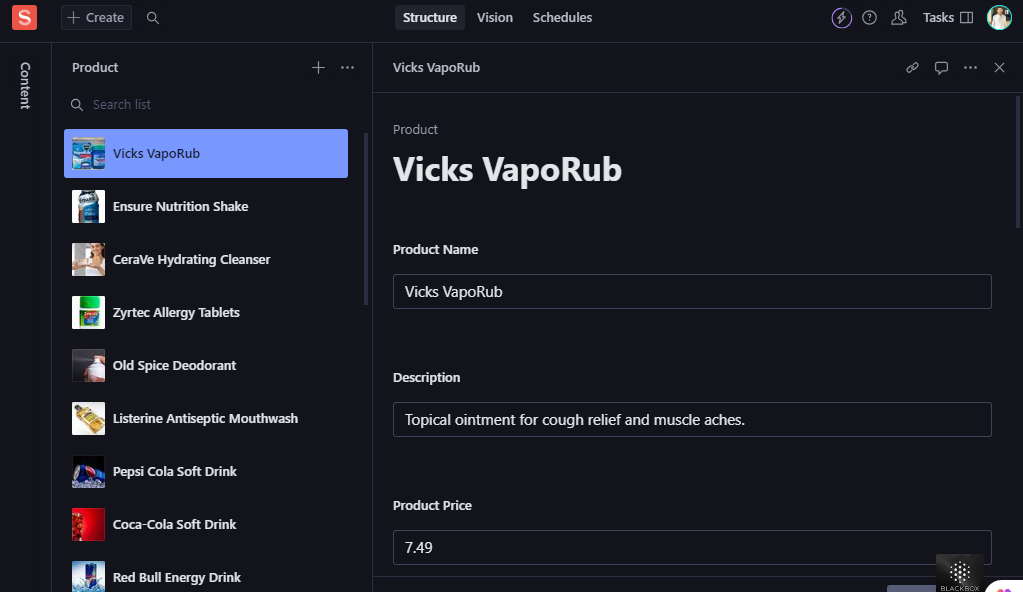
The g function uses a GROQ query to fetch data from Sanity during server-side rendering (SSR). This ensures that data is pre-fetched and injected into the page before the user sees it.

# **Schema Code**



The schema defines the structure of the Pharmaceutical Q-Commerce content in the Sanity CMS.

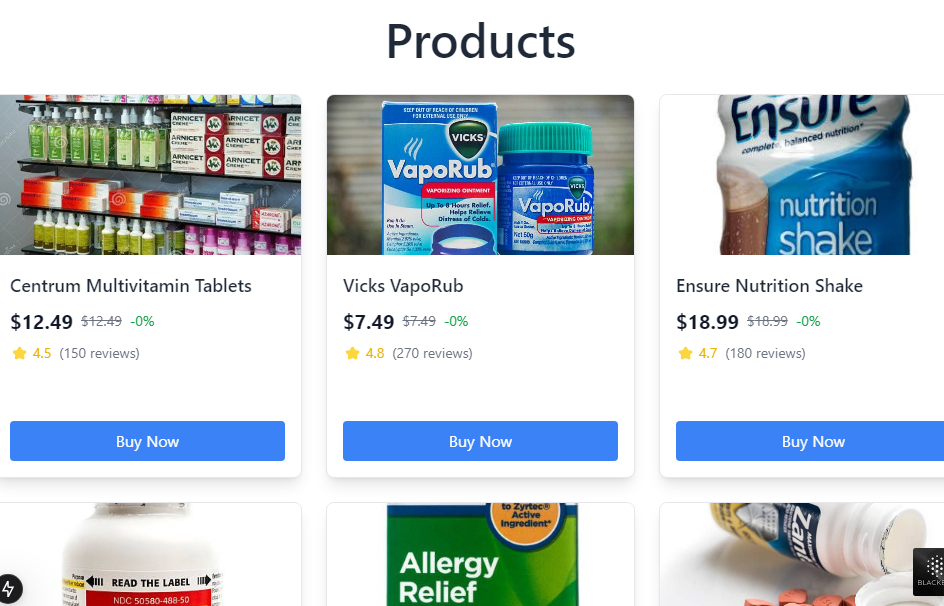
# **Sanity Product Schema**



The Sanity product schema defines the structure for storing furniture product data in the Sanity Content Management System (CMS).

**Data successfully displayed in the frontend.**

The product data was successfully fetched from Sanity CMS using GROQ queries and displayed on the frontend.



# **Conclusion**

Day 3 of the hackathon focused on setting up the backend for the Pharmaceutical Q-Commerce platform and integrating the data. The following were accomplished:

1. Custom migration code transferred data from the API to Sanity CMS.
2. A structured schema was created to ensure data consistency.
3. The frontend fetched and displayed the data dynamically using GROQ queries.
4. Product cards were designed to showcase pharmaceutical items attractively and responsively.
5. Environment variables were used to securely handle API configurations.