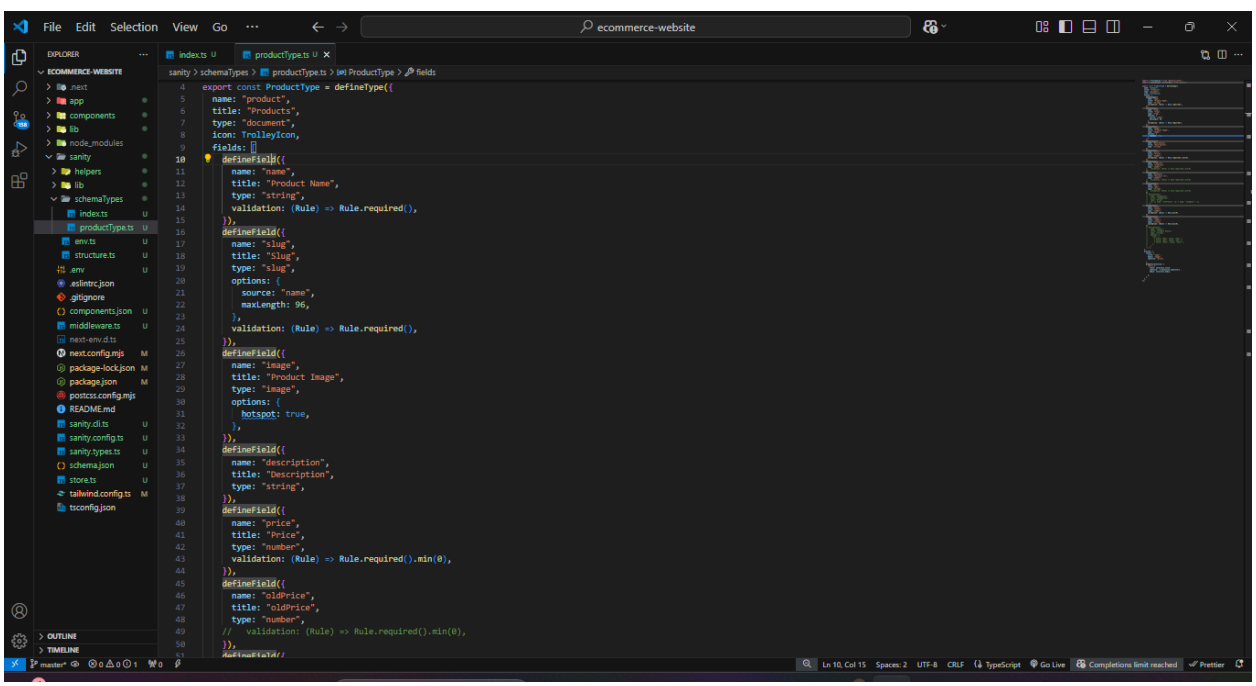
Day 3: Sanity Data Rendering in Shop&Co Marketplace

This documentation outlines the work completed on Day 3 of the Shop&Co Marketplace hackathon. It focuses on integrating Sanity for direct data rendering without migration, schema creation, and displaying data using GROQ queries in a Next.js application.

Schema Definition:

The schema defines the structure of the Shop&Co content in Sanity CMS. Below is an overview of its components:



Schema Fields:

- **Title:** Stores the product name (e.g., "SKINNY FIT JEANS").
- **Slug:** A unique identifier for dynamic routing in the frontend.

- **Description:** Provides details about the product.
- **Price:** Numeric field representing the product's cost.
- **Discount:** Represents any available discount on the product.
- **Image:** Stores product images.

Custom Validation:

Validation rules are implemented to ensure data integrity:

✔ **Title** and **Price** are required fields and cannot be empty.
✔ Ensures that the **Slug** field is unique for SEO-friendly URLs.

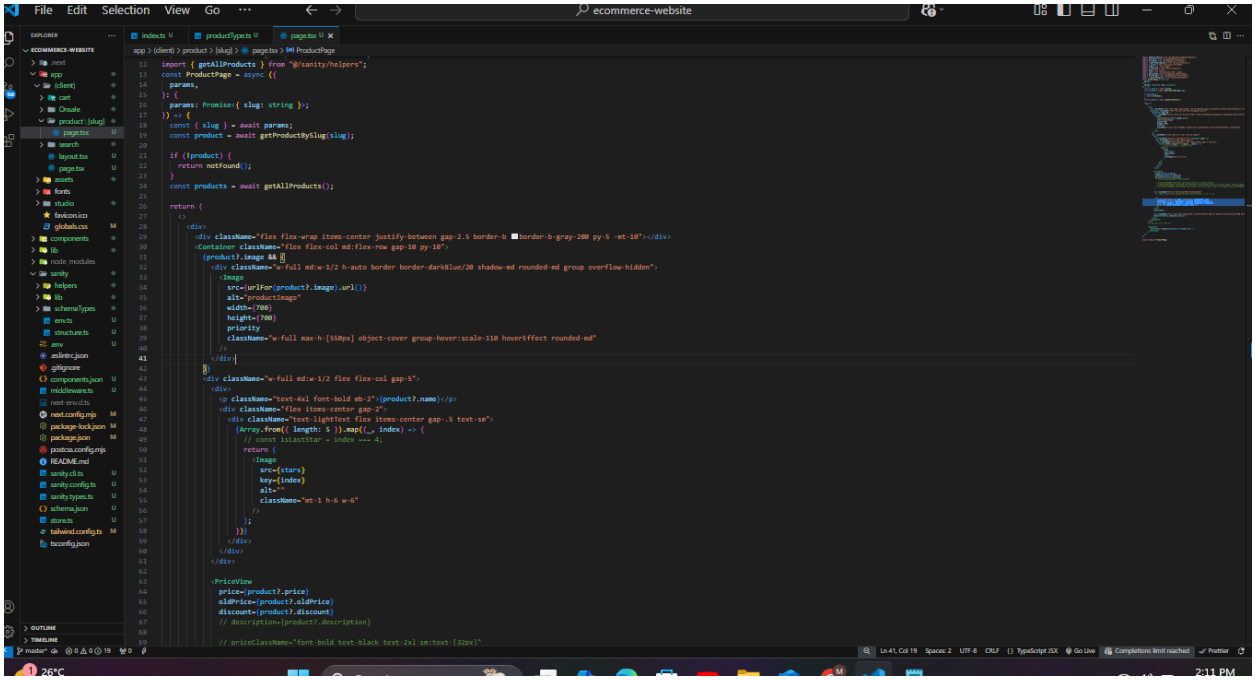
Optimized GROQ Query Access:

To maintain a **clean and scalable** code structure, a dedicated **helper folder** was created within Sanity. This folder includes:

- `queries.ts` → Contains reusable **GROQ queries** for fetching data.
- `index.ts` → Handles structured data fetching and query exports.

Product Page Code Overview:

This code represents the **Product Page** for an e-commerce store. Users can view **detailed information** about a clothing item, and the product data is dynamically fetched from the database.



Fetching Data:

- Fetches product details using `getProductBySlug(slug)`.
- Retrieves **all products** using `getAllProducts()` for **related item suggestions**.

Data Handling & Props:

Instead of **static props**, product data (**name, image, price, etc.**) is dynamically retrieved based on the **slug** in the URL.

Design & Styling:

✔ Built with **Next.js** and **Tailwind CSS**.
✔ Fully **responsive** layout.
✔ The **Container** component structures the layout efficiently.

Dynamic Features:

Product Information

- **Product Image:** Displayed via Next.js **Image** component using `urlFor()`.
- **Ratings:** Implements a **five-star rating system** using an imported `stars` image.
- **Price Display:** The `PriceView` component shows **discounted prices** dynamically.
- **Description:** Currently a placeholder, but ready for **dynamic content rendering**.
- **Add to Cart:** `AddToCartButton` component allows users to **add items to the cart**.

Related Products:

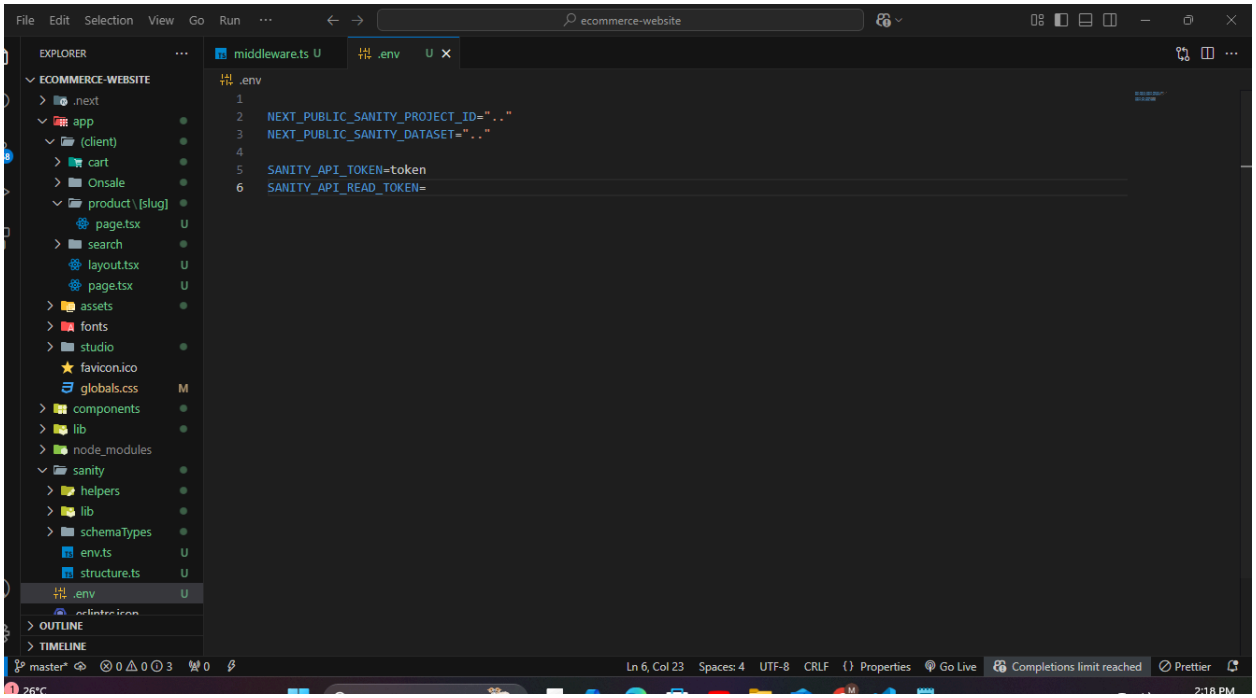
- The `FourProduct` component fetches and displays **recommended items** at the bottom of the page.

Component Reusability:

Several components are **reused** across the platform:

✔ `AddToCartButton`
✔ `PriceView`
✔ `FourProduct`

Environment Variables & API Security:



To secure **API configurations**, sensitive keys are stored in the **.env** file.

Sanity Configuration

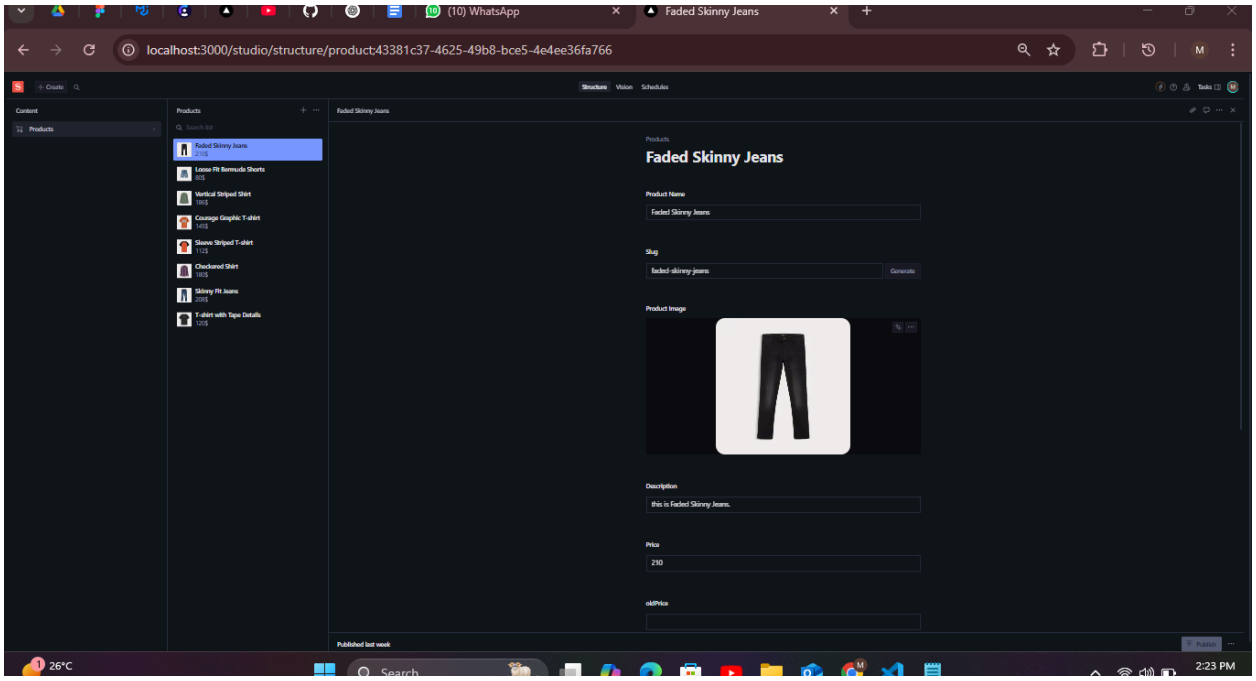
- **NEXT_PUBLIC_SANITY_PROJECT_ID** → Unique identifier for the Sanity project.
- **NEXT_PUBLIC_SANITY_DATASET** → Defines the dataset (e.g., **production**, **development**).

API Security:

- **SANITY_API_TOKEN** → A secure token used for API authentication (**never expose it to the frontend**).

Security Best Practices:

✔ Environment variables are stored securely and accessed via process.env.
✔ Credentials are **never hardcoded** in the frontend.



Schema Breakdown:

1 Title Field

- **Type: String**
- **Purpose: Stores the product name (e.g., "Harmony Modular Sectional").**
- **Validation: Ensures the field cannot be empty, as it is crucial for identifying products.**

2 Slug Field

- **Type: Slug**
- **Purpose: Creates a unique, SEO-friendly URL identifier (e.g., "skinny-fit-jeans").**

3️⃣ Image Field

- Type: Image
- Purpose: Stores a single product image.

4️⃣ Description Field

- Type: Text
- Purpose: Provides a detailed product description.

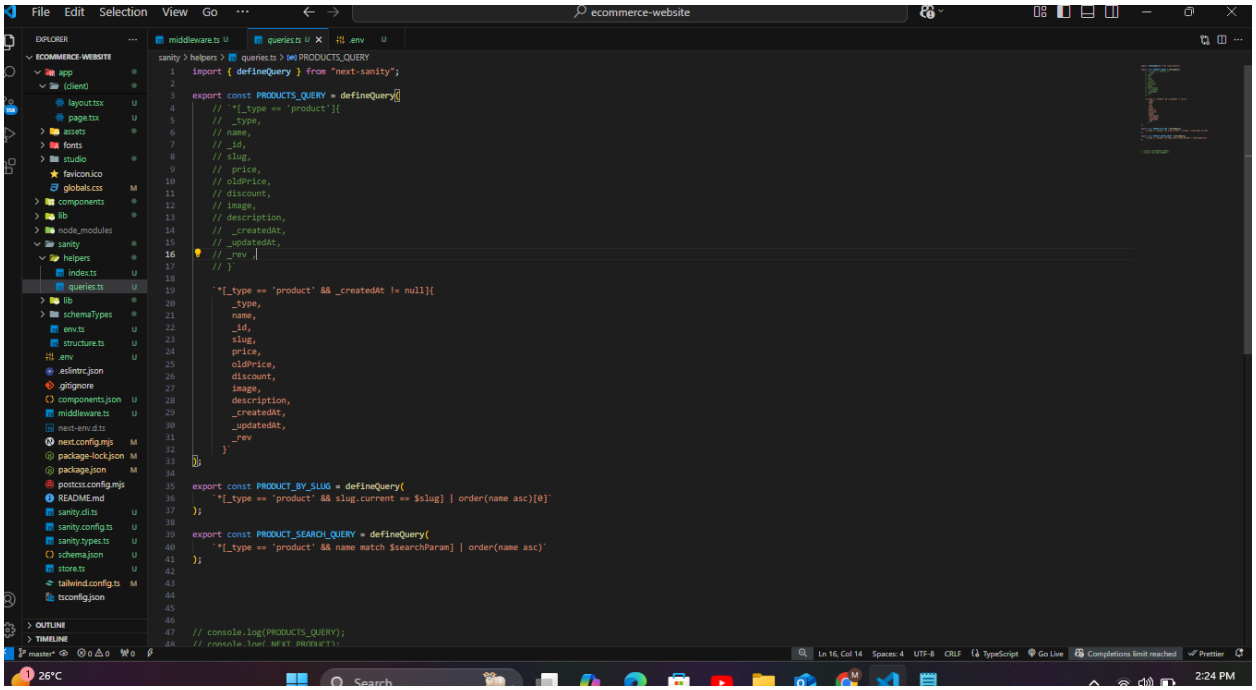
5️⃣ Price Field

- Type: Number
- Purpose: Represents the cost of the product.

Scalability & Frontend Integration:

The schema is designed for scalability:

✔ Additional fields (e.g., stock levels, dimensions, materials) can be added without breaking the existing structure.
✔ Each field is accessible via GROQ queries, ensuring seamless data retrieval.



Summary of Day 3 Progress

The Day 3 milestone focused on backend setup and Sanity CMS integration. Key accomplishments include:

✔ Structured Schema Development: Ensured consistent data storage for products.
✔ GROQ Query Implementation: Used to dynamically fetch product data.
✔ Product Page Rendering: Dynamically displayed product details, prices, and related items.
✔ Environment Variables & Security: Implemented safe API configurations.
✔ Reusable UI Components: Created a scalable component-based architecture for better maintainability.