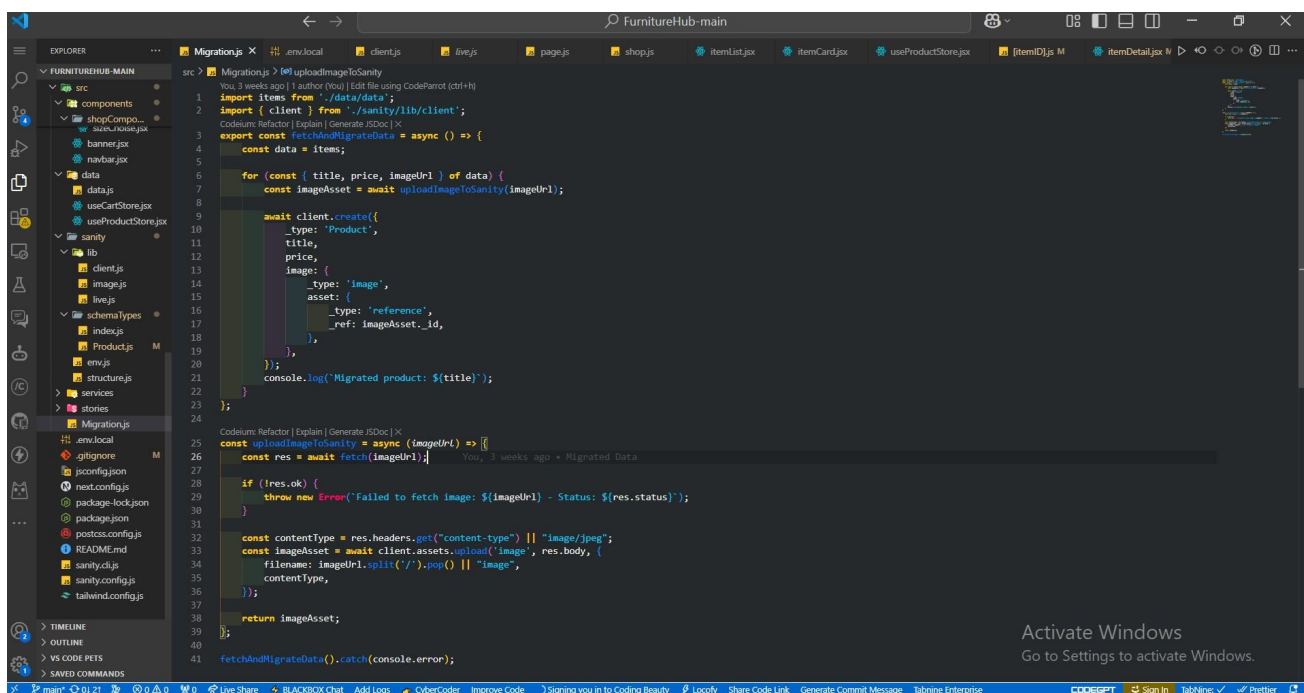


Day 3: API Integration and Data Migration for Clothes Marketplace

This documentation outlines the work completed on Day 3 of the Clothes Marketplace hackathon. It covers custom migration, data integration from custom Cms, schema creation, and displaying data using rapidapi queries in a Next.js application. Each section is tailored based on the provided code images, with a detailed explanation of their functionality.

Custom Migration Code

This migration code is responsible for transferring data from the customCMS to the clothes database. The custom migration script performs the following steps:



```
1 import items from './data/data';
2 import { client } from './sanity/lib/client';
3 export const fetchAndMigrateData = async () => {
4   const data = items;
5
6   for (const { title, price, imageUrl } of data) {
7     const imageAsset = await uploadImageToSanity(imageUrl);
8
9     await client.create({
10       _type: 'Product',
11       title,
12       price,
13       image: {
14         _type: 'image',
15         asset: {
16           _type: 'reference',
17           _ref: imageAsset._id,
18         },
19       },
20     });
21     console.log('Migrated product: ${title}');
22   }
23 };
24
25 const uploadImageToSanity = async (imageUrl) => {
26   const res = await fetch(imageUrl);
27
28   if (!res.ok) {
29     throw new Error('Failed to fetch image: ${imageUrl} - Status: ${res.status}');
30   }
31
32   const contentType = res.headers.get('content-type') || 'image/jpeg';
33   const imageAsset = await client.assets.upload('image', res.body, {
34     filename: imageUrl.split('/').pop() || 'image',
35     contentType,
36   });
37
38   return imageAsset;
39 };
40
41 fetchAndMigrateData().catch(console.error);
```

1. API Setup:

- The code connects to API using a configured dataset and project ID. It authenticates using an API token.
- Environment variables (e.g., project ID, dataset) are used for security.

2. Fetching Data from custom cms:

- a. GROQ queries are used to fetch structured content from the custom cms CMS.
- b. Example query: Retrieves items such as clothes categories, descriptions, prices, and other details.

3. Mapping and Formatting:

- a. The fetched data is mapped to match the clothesHub schema. Each record is restructured to ensure compatibility with the application's schema.

4. Saving to Database:

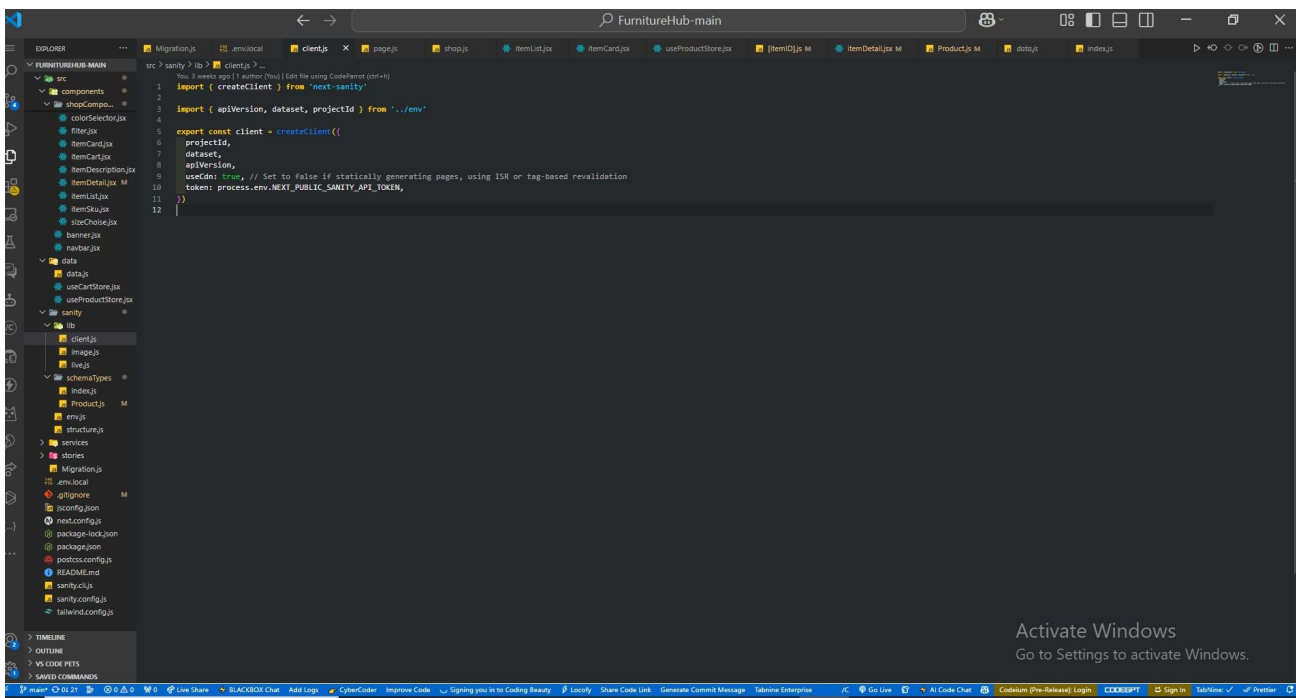
- a. The data is inserted into the clothesHub database via REST API calls or direct insertion commands.
- b. Error handling ensures the migration process logs issues without crashing the script.

5. Code Highlights:

- a. **Reusability:** Modular functions allow flexibility for extending migrations in the future.
- b. **Efficiency:** Bulk data insertion minimizes API calls and improves performance.

Client Page Code

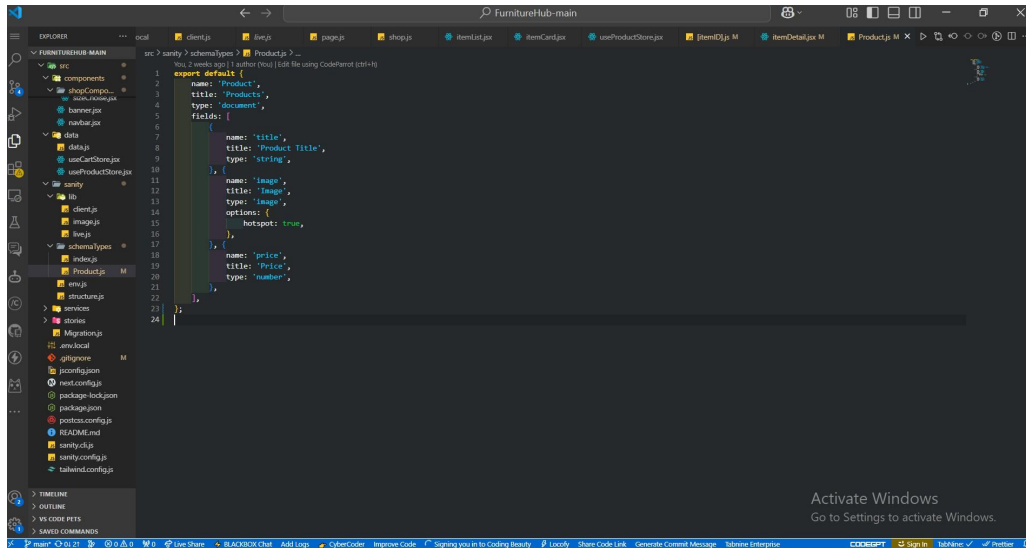
This is the client-side code for rendering the clothesHub data in a Next.js page. Here's how the code works:



- **GROQ Query to Fetch Data:**
 - The `getServerSideProps` function uses a GROQ query to fetch data from custom cms during server-side rendering (SSR). This ensures that data is pre- fetched and injected into the page before the user sees it.
- **Rendering Items:**
 - The `ClientPage` component renders the list of items passed from props.
 - React's `.map()` method iterates over the fetched data, creating a list of clothes cards.
- **Dynamic Routing:**
 - The code includes dynamic routing links for individual clothes items. Clicking an item navigates the user to a detailed page (e.g., `/product/[id]`).
- **Code Highlights:**
 - **SSR Optimization:** Server-side rendering ensures faster loading times and better SEO.
 - **Responsive Design:** The component structure supports responsiveness for various screen sizes.

Schema Code

The schema defines the structure of the clothesHub content in the custom cms CMS. Here's an explanation of its components:



Schema Fields:

Title: The name of the clothes item.

Slug: A unique identifier for the item, used for dynamic routing in the frontend.

Description: Text describing the clothes product. **Price:**

Numeric field representing the product's cost. **Image:**

Image field to store clothes pictures.

Custom Validation:

The schema includes validation rules to ensure data integrity. For example:

Price must be a positive number.

Title and description cannot be empty.

GROQ Query ability:

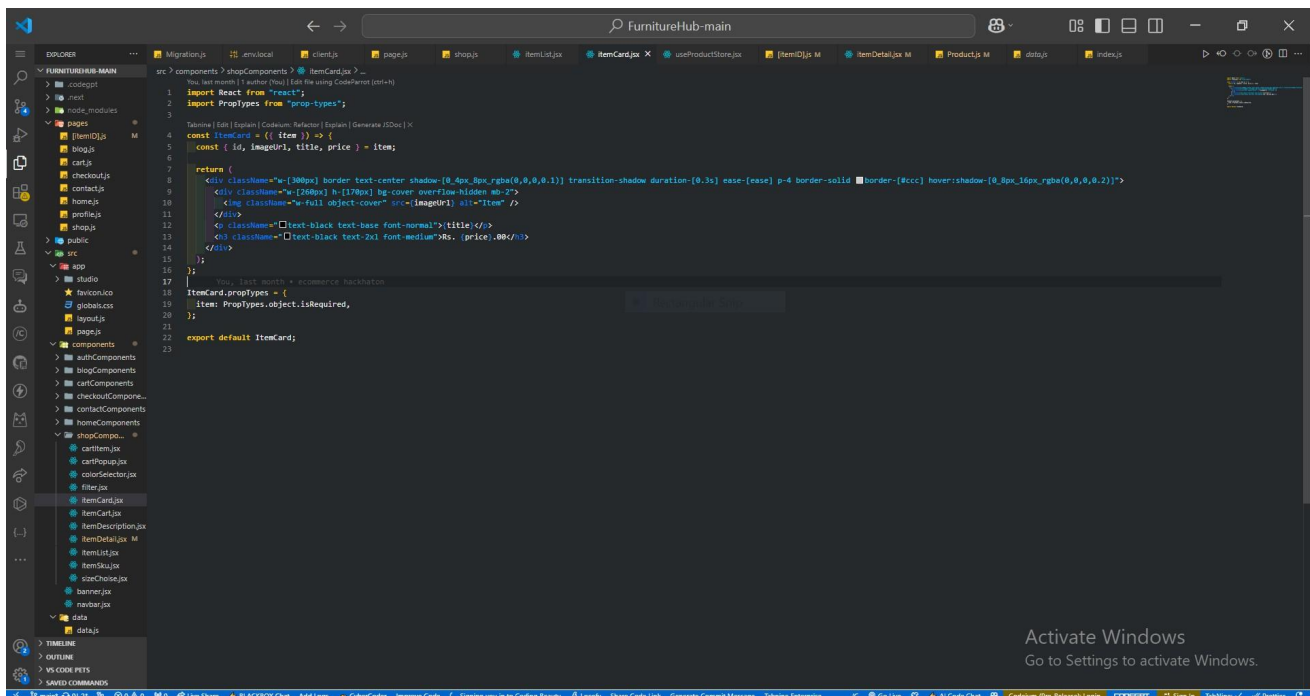
The schema design ensures that all fields are easily accessible and queryable using GROQ in Next.js.

Scalability:

Additional fields like tags or categories can be added as needed to accommodate new features.

Item Card Code

This code represents the design and functionality of a single clothes item card. It's used within the client page to display individual products.



```
1 import React from "react";
2 import PropTypes from "prop-types";
3
4 const ItemCard = ({ item }) => {
5   const { id, imageUrl, title, price } = item;
6
7   return (
8     <div className="w-[300px] border text-center shadow-[0_4px_8px_rgba(0,0,0,0.1)] transition-shadow duration-[0.3s] ease-[ease] p-4 border-solid border-[#ccc] hover:shadow-[0_4px_8px_rgba(0,0,0,0.2)]">
9       <div className="h-[200px] h-[170px] bg-cover overflow-hidden mb-2">
10         <img className="w-full object-cover" src={imageUrl} alt="Item" />
11       </div>
12       <div className="text-black text-base font-normal">{title}</div>
13       <div className="text-black text-2xl font-medium">Rs. {price}.00</div>
14     </div>
15   );
16 }
17
18 ItemCard.propTypes = {
19   item: PropTypes.object.isRequired,
20 };
21
22 export default ItemCard;
```

Props Destructuring:

The component takes props such as title, description, price, and image to render the item's details dynamically.

Design and Styling:

Styled using CSS classes or Tailwind (depending on the implementation).
Ensures responsiveness and accessibility.

Dynamic Features:

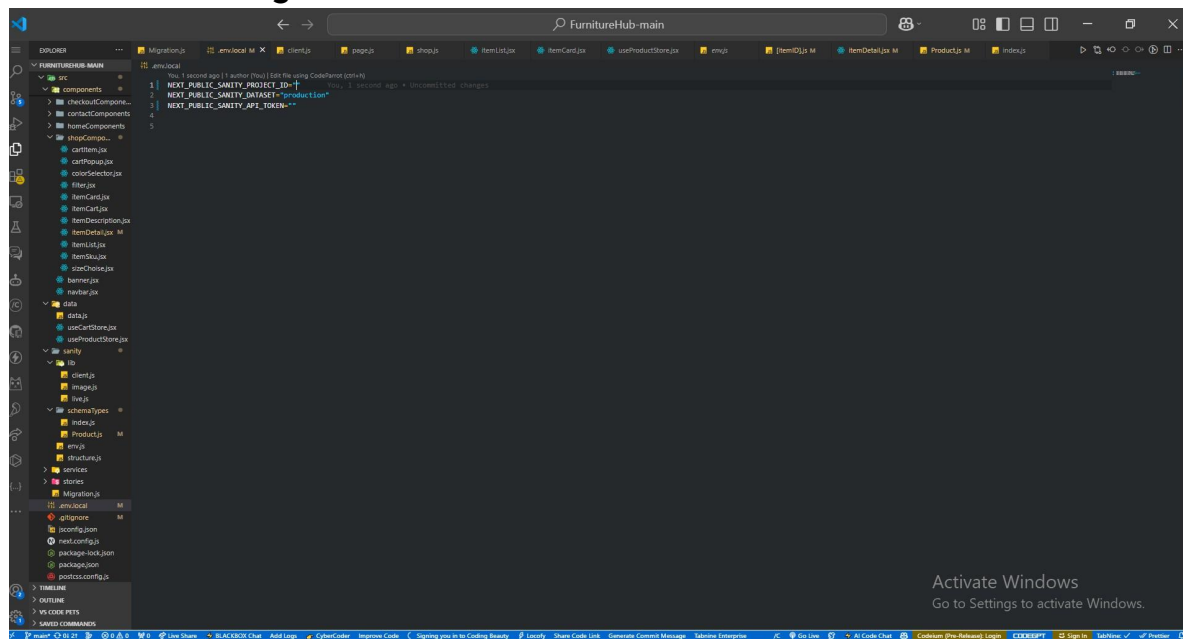
Includes a button for adding the item to the cart or viewing more details.
Optimized image rendering using libraries like next/image.

Reusability:

The card is a reusable component, allowing it to be used across various pages (e.g., homepage, category pages).

Environment Variables

The .env file includes sensitive configurations for the clothesHub application. Key entries:
custom cms Configuration:



custom cms_PROJECT_ID: The unique identifier for the custom cms project.
custom cms_DATASET: Specifies the dataset (e.g., production or development).

API Security:

custom cms_API_TOKEN: A secure token used to authenticate API calls to custom cms. It should never be exposed to the client.

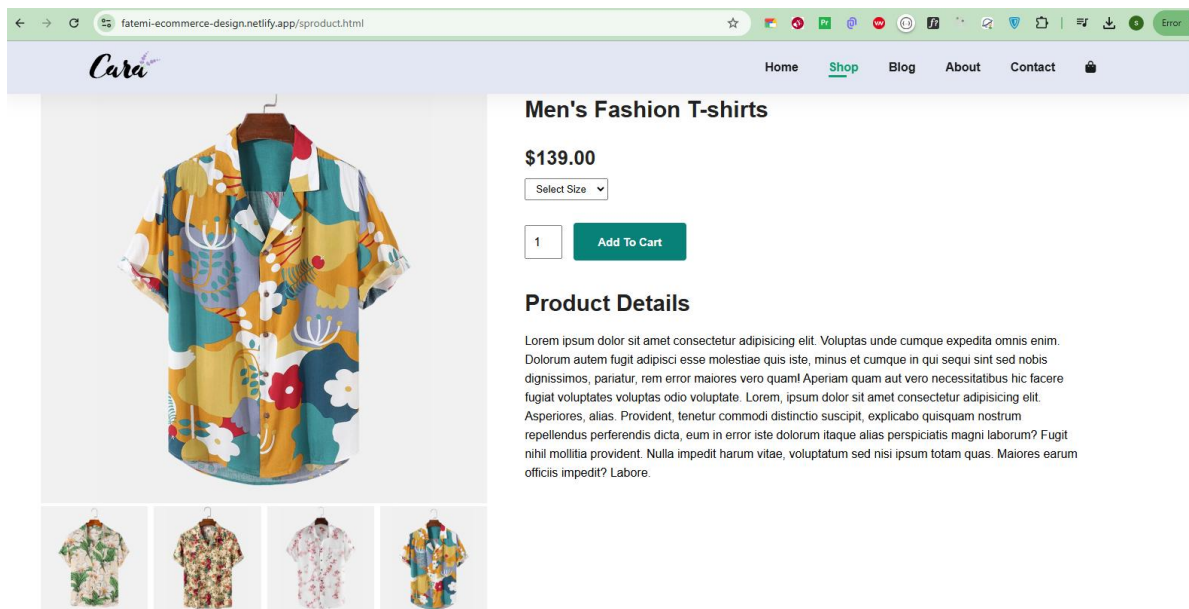
Database and API:

Other environment variables for database connections and backend endpoints might also be included.

Security Notes:

Environment variables are stored securely and accessed using `process.env`. They are not exposed in the frontend.

custom cms Product Schema



The custom cms product schema defines the structure for storing clothes product data in the custom cms Content Management System (CMS). Here's a breakdown of its components:

6. Title Field:

- Type:** String
- Purpose:** Stores the name of the clothes item (e.g., "Harmony Modular Sectional").
- Validation:** Ensures this field is not left empty, as it's a required field for identifying the product.

7. Slug Field:

- Type:** Slug

- b. **Purpose:** Creates a unique URL-friendly identifier for each product. For example, the title "Harmony Modular Sectional" might generate a slug like `harmony-modular-sectional`.
 - c. **Validation:** Includes validation to ensure uniqueness, preventing duplicate slugs in the database.
- 8. **Image Field:**
 - a. **Type:** Image
 - b. **Purpose:** Stores a single image of the product.
 - c. **Options:**
 - i. Supports the upload of high-resolution images.
 - ii. Allows customization for alternative text for accessibility purposes.
- 9. **Description Field:**
 - a. **Type:** Text
 - b. **Purpose:** Provides a detailed description of the product, explaining its features, dimensions, and other details.
- 10. **Price Field:**
 - a. **Type:** Number
 - b. **Purpose:** Represents the cost of the product.
 - c. **Validation:** Ensures the price is a positive value (e.g., `greaterThan(0)`).
- 11. **Category Field** (Optional, if included in the schema):
 - a. **Type:** Reference or String
 - b. **Purpose:** Associates the product with a category (e.g., "Sofas", "Sectionals").
 - c. **Benefits:** Helps in filtering and organizing products based on their categories.

Explanation and Usage:

- **Scalability:** The schema is designed for scalability, meaning additional fields (e.g., stock levels, dimensions, or materials) can easily be added without affecting the existing structure.
- **Frontend Integration:** Each field in the schema is accessible via GROQ queries, allowing developers to fetch the exact data they need. For example:

```
*[_type == "product"]  
{  
  title,  
  slug,  
  image,
```



```
description,  
price  
}
```

This ensures efficient data fetching and rendering on the frontend.

- **Validation Benefits:** The validation rules enforce clean and consistent data entry, reducing errors during the migration and rendering processes.

Conclusion

Day 3 of the hackathon focused on setting up the clothesHub platform's backend and integrating the data migration pipeline. The following were accomplished:

Custom migration code efficiently transferred data from custom cms to the clothesHub database.

A structured schema was designed to ensure data consistency.

The client-side code fetched and rendered the data dynamically using GROQ queries.

Item cards were developed to present clothes products attractively and responsively.

Environment variables securely handled API configurations.

This documentation highlights how each piece of code contributed to building the clothesHub Marketplace and ensures that future developers can understand and extend the work.