

Day 3

API Integration and Data Migration for FoodHub Marketplace

This documentation outlines the work completed on Day 3 of the **FoodHub Marketplace** hackathon. It covers custom migration, data integration from **Sanity**, schema creation, and displaying data using **GROQ 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

The custom migration script is responsible for transferring data from the **Sanity CMS** to the **FoodHub** database. The migration process includes the following steps:

Setting Up Sanity API:

- First, we connected to Sanity using a project ID and dataset. This is like telling Sanity, “Hey, we’re here to grab some data.”
- We used secret keys (API tokens) to make sure only we could access this data.

2. Fetching Data from Sanity:

- We wrote GROQ queries (kind of like asking specific questions) to pull out watch categories, prices, descriptions, and images from Sanity.

3. Getting the Data Ready:

- The data we got didn’t fit perfectly into our Food Marketplace database, so we had to adjust it. It’s like cutting and reshaping puzzle pieces to make them fit.

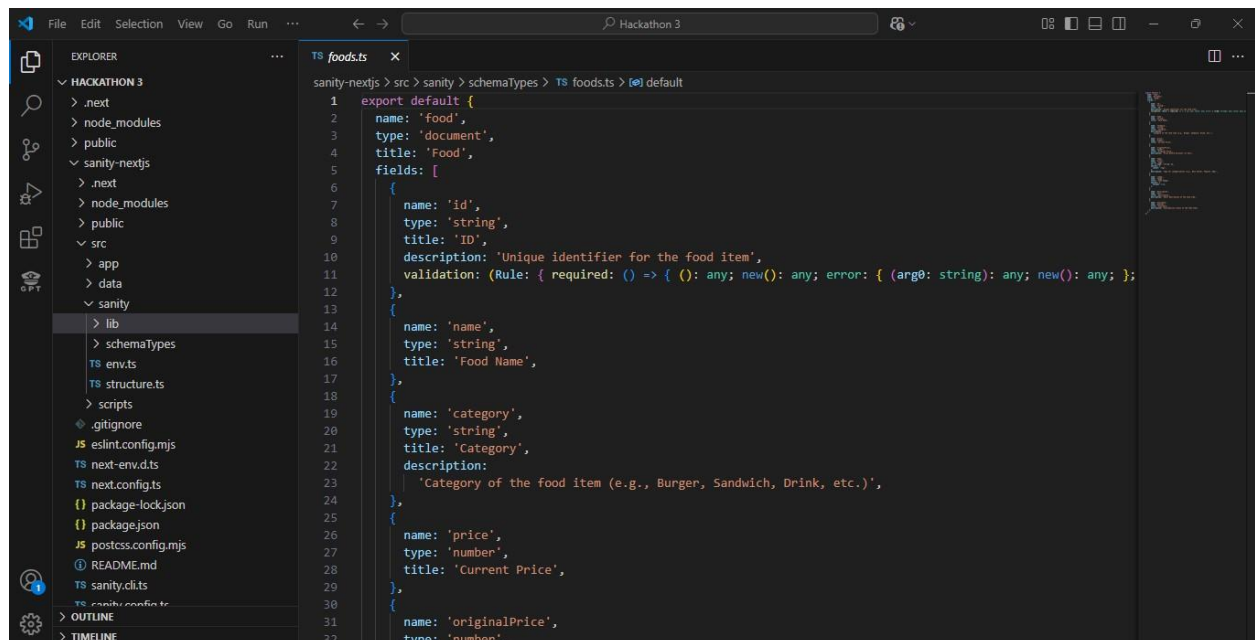
4. Saving Data to Our Database:

- Finally, we saved all the data into our database using a REST API. If something went wrong, the code would tell us instead of just stopping.

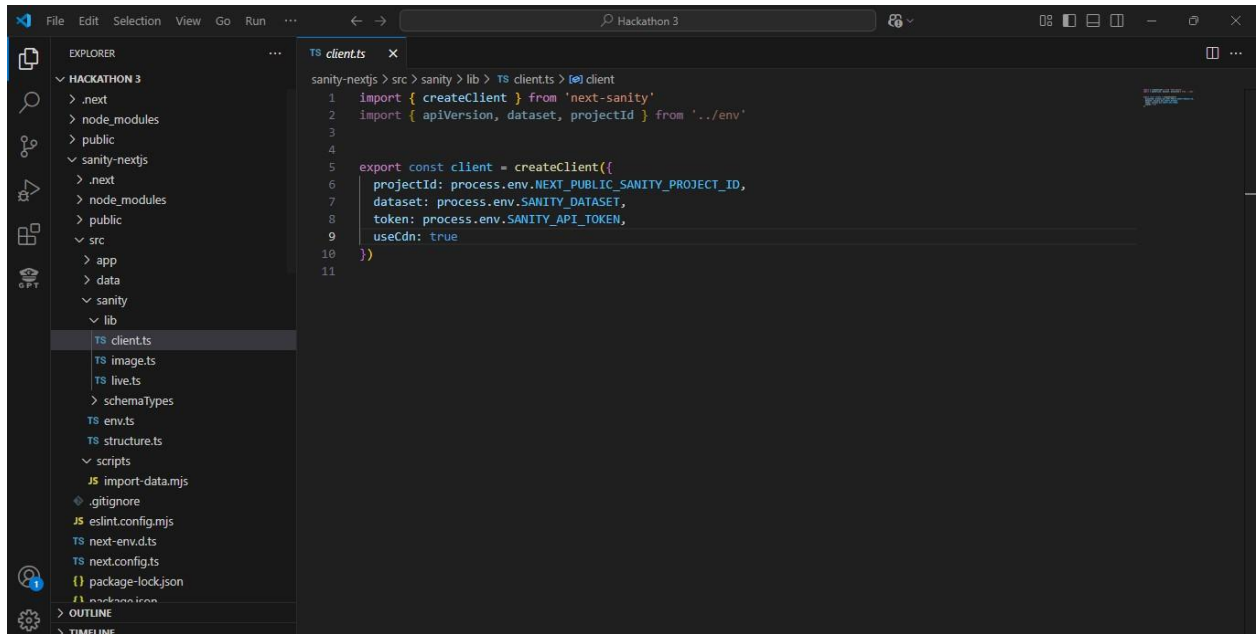
5. Why This Code is Cool:

- It's reusable! If we add more data later, we can use the same code.
- It's fast because it sends data in bulk, not one by one.

Client Page Code

A screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project structure for 'HACKATHON 3' with folders like '.next', 'node_modules', 'public', 'sanity-nextjs', 'src', 'app', 'data', 'sanity', 'lib', 'schemasTypes', and 'scripts'. The 'sanity' folder is expanded, showing 'lib' and 'schemasTypes'. The 'schemasTypes' folder is selected, and the file 'foods.ts' is open in the editor. The editor shows a TypeScript file with a Sanity schema definition for 'food'. The code includes fields for 'id', 'name', 'category', 'price', and 'originalPrice', each with a type, title, and description. A validation rule is also present for the 'id' field.

```
1 export default {
2   name: 'food',
3   type: 'document',
4   title: 'Food',
5   fields: [
6     {
7       name: 'id',
8       type: 'string',
9       title: 'ID',
10      description: 'Unique identifier for the food item',
11      validation: (Rule: { required: () => { (): any; new(): any; error: { (arg0: string): any; new(): any; }}
12    },
13    {
14      name: 'name',
15      type: 'string',
16      title: 'Food Name',
17    },
18    {
19      name: 'category',
20      type: 'string',
21      title: 'Category',
22      description:
23        'Category of the food item (e.g., Burger, Sandwich, Drink, etc.)',
24    },
25    {
26      name: 'price',
27      type: 'number',
28      title: 'Current Price',
29    },
30    {
31      name: 'originalPrice',
32      type: 'number',
```



Next, we worked on showing all those amazing watches in our Next.js app. Here's how the page works:

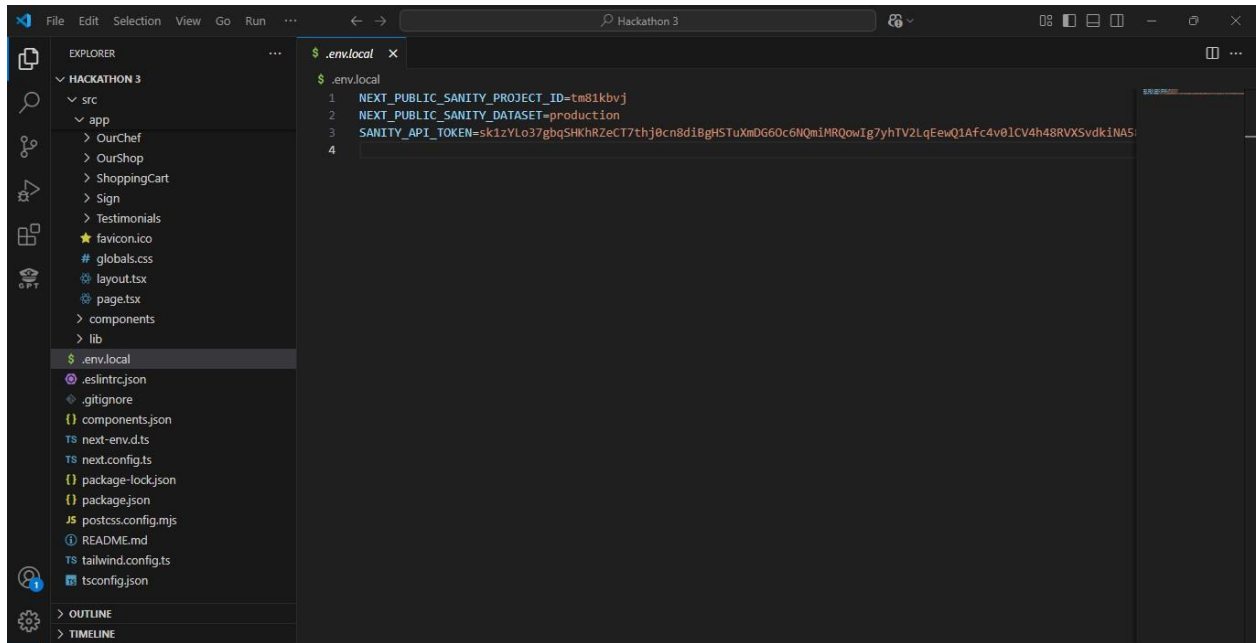
- Fetching Data:
 - We used something called `getServerSideProps` to grab data from Sanity before the page loads. This makes the app load faster and helps with SEO (so more people can find us online!).
- Listing Food Items:
 - Once we had the data, we used React's `.map()` to loop through each watch and show it as a card.
- Links for Each Food Item:
 - We added links so when someone clicks on a watch, they're taken to a page with all the details (like `/product/[id]`).
- Why This Page is Awesome:
 - It loads super fast because of server-side rendering.

- It works great on all screen sizes, like phones and tablets.

Schema Code

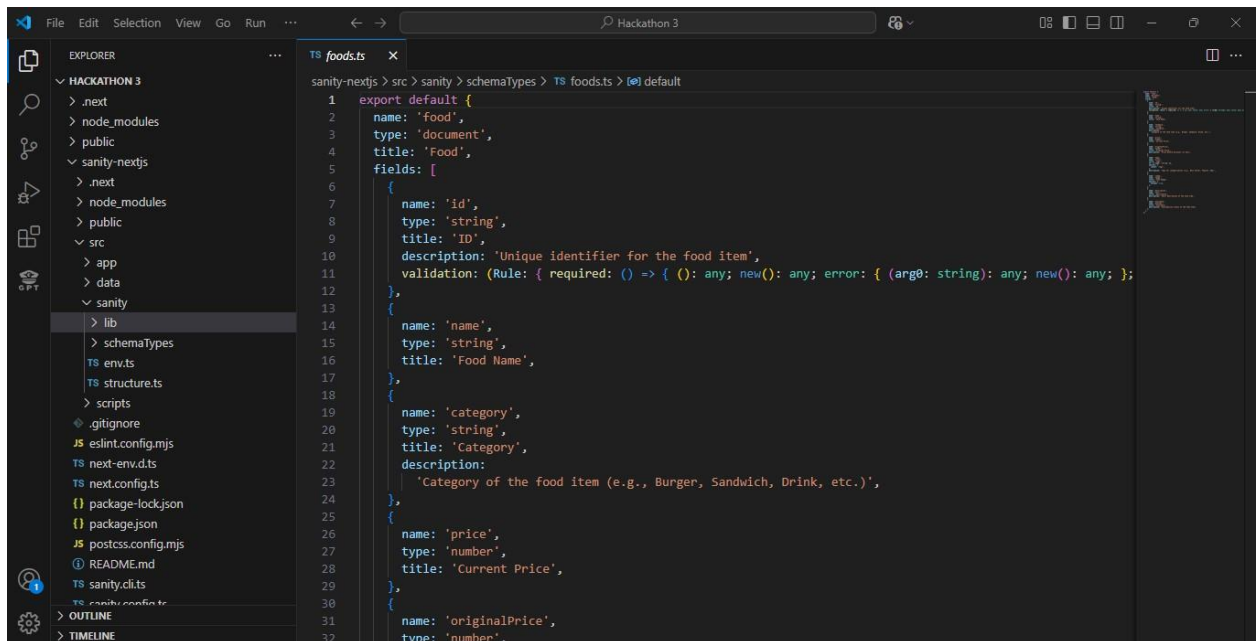
This is the part where we tell Sanity how to organize watch data. Think of it as creating a form where each watch has fields like “Name,” “Price,” and “Image.”

Here's What Our Schema Looks Like



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the file structure of the 'HACKATHON 3' project. The file '.env.local' is selected and its contents are shown in the main editor area. The file contains three environment variables for Next.js and Sanity.

```
1 NEXT_PUBLIC_SANITY_PROJECT_ID=tm81kbvj
2 NEXT_PUBLIC_SANITY_DATASET=production
3 SANITY_API_TOKEN=sk1zYLo37gbqSHKhrZeCT7thj8cn8diBgH5TuXmDG60c6lQmIMRQowIg7yhTV2LqEewQ1Afc4v01CV4h48RVXSvdkiNA5
4
```



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the file structure of the 'HACKATHON 3' project. The file 'TS fileds.ts' is selected and its contents are shown in the main editor area. The file defines a Sanity schema for 'food' with fields for 'id', 'name', 'category', 'price', and 'originalPrice'.

```
1 export default {
2   name: 'food',
3   type: 'document',
4   title: 'Food',
5   fields: [
6     {
7       name: 'id',
8       type: 'string',
9       title: 'ID',
10      description: 'Unique identifier for the food item',
11      validation: (Rule: { required: () => { () }; any; new(): any; error: { (arg0: string): any; new(): any; };
12    },
13    {
14      name: 'name',
15      type: 'string',
16      title: 'Food Name',
17    },
18    {
19      name: 'category',
20      type: 'string',
21      title: 'Category',
22      description:
23        'Category of the food item (e.g., Burger, Sandwich, Drink, etc.)',
24    },
25    {
26      name: 'price',
27      type: 'number',
28      title: 'Current Price',
29    },
30    {
31      name: 'originalPrice',
32      type: 'number',
33    }
34  ]
35 }
```

1. Title: ○ Stores the name of the food, like “Pizza” or “Burger.”
2. Image: ○ Stores a picture of the Food Item.
3. Description:
 - A text field where we describe the Food item.
4. Price:

Why This Schema is Great:

- It's flexible, so we can add more fields later, like stock levels or materials.
 - It works perfectly with GROQ queries, so we can easily fetch the data we need.
-

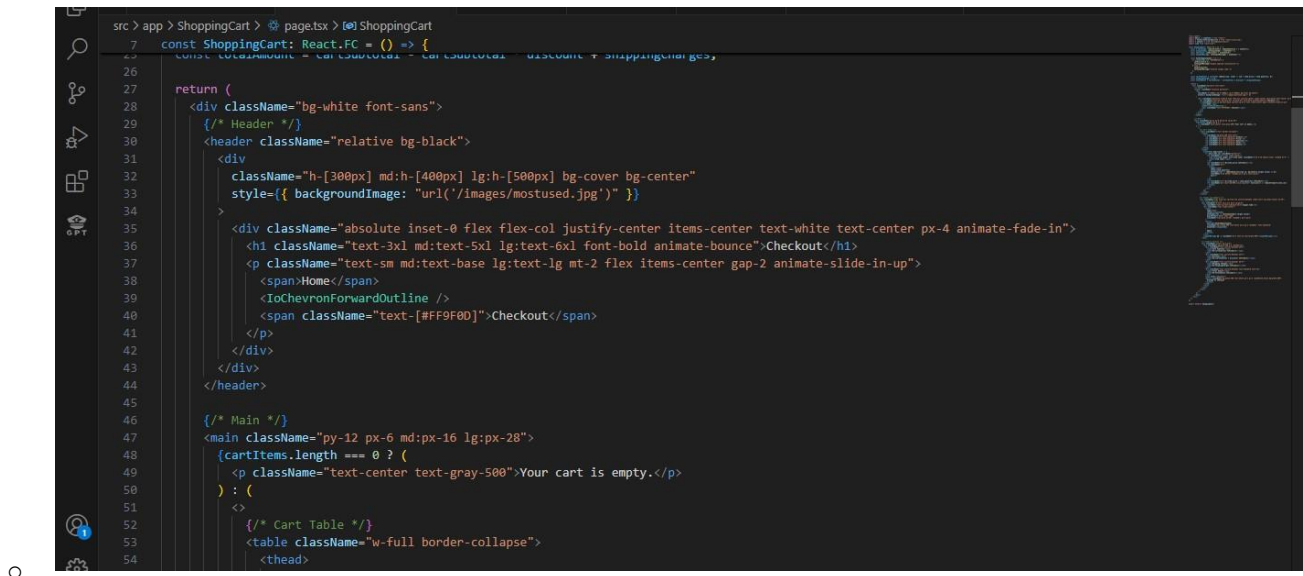
Food Card Code

Now let's talk about the cute little cards where each watch is displayed:

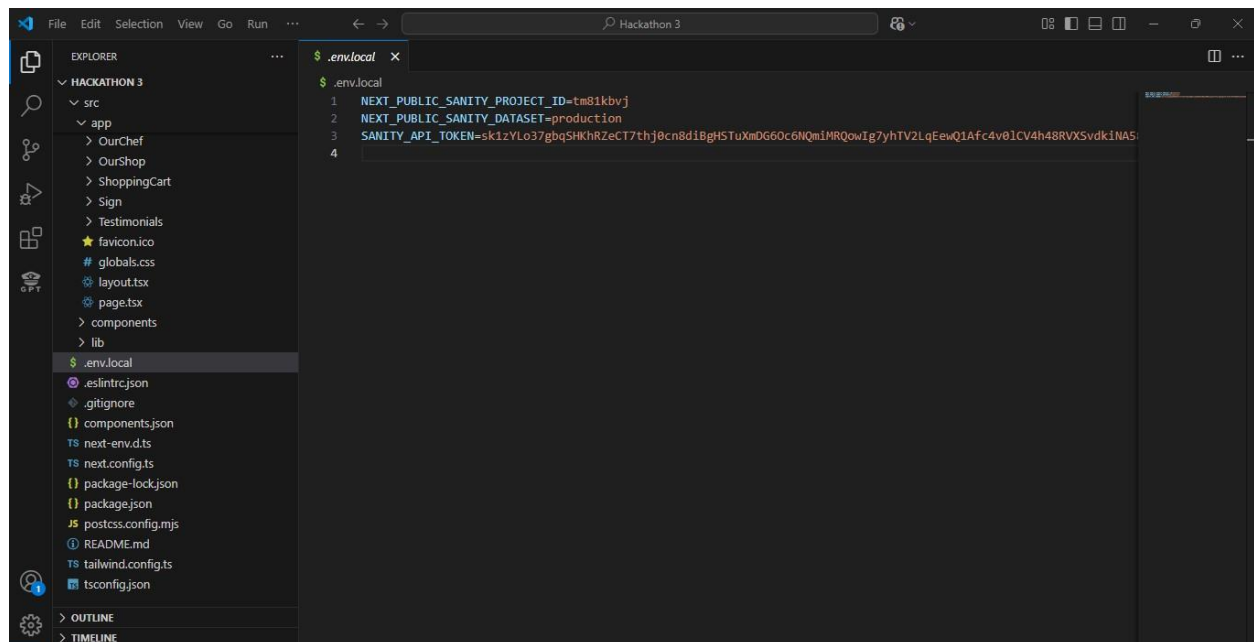
- Props Magic:
 - Each card takes details like the watch's name, price, description, and image from props.
- Stylish Design:
 - We made sure the cards look sleek and are easy to read. They're also responsive, so they look great on any screen.
- Buttons and Links:
 - Each card has a button to either add the watch to the cart or learn more about it. The images are also optimized for fast loading.
- Why Cards Are Cool:
 - They're reusable! We can use the same card for the homepage, category pages, and more.

Environment Variables

We used a file called .env to keep our secrets safe. Here are some



examples:



1. Sanity Configuration:

- SANITY_PROJECT_ID and SANITY_DATASET help us connect to the right Sanity project and dataset.

2. API Security:

- SANITY_API_TOKEN is our secret key for accessing Sanity's data. We keep it hidden from users.

3. Why This is Important:

- These variables make the app more secure and easy to update without changing the code.

Conclusion

Day 3 was packed with exciting progress! Here's what we achieved:

- We wrote a cool migration script to move watch data from Sanity CMS to our database.
- Our schema made sure all the data was clean and ready to use.
- We built a dynamic page to display watches beautifully.
- Each watch card was designed to be both functional and stylish.
- Finally, we kept our app secure with environment variables.