

Day 3

API Integration and Data Migration for Nike app Template 3

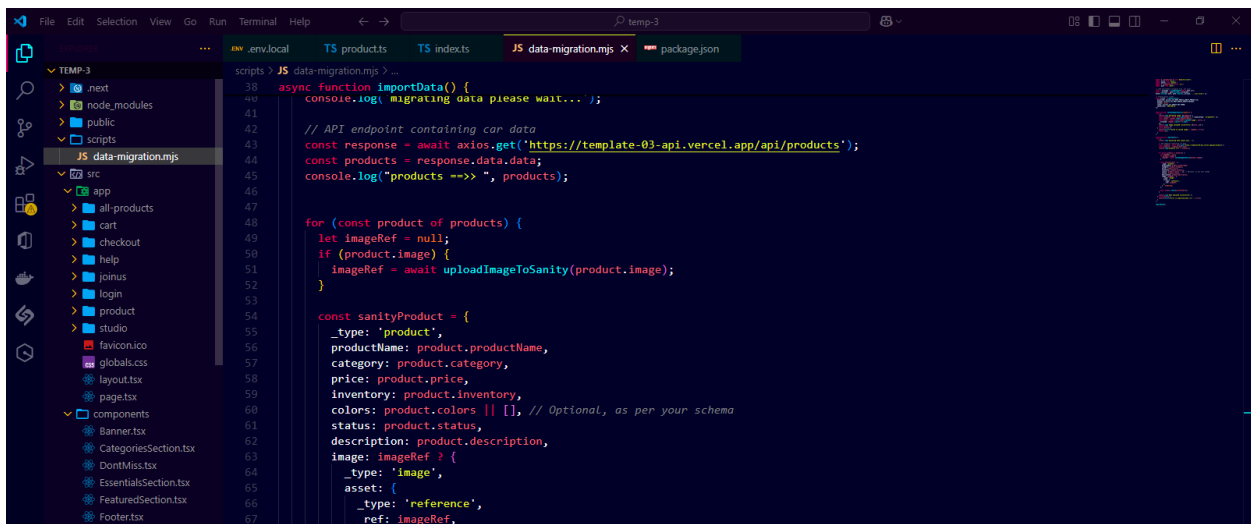
This documentation outlines the work completed on Day 3 of the API Integration and Data Migration for Nike app Template 3 hackathon. It covers custom migration code, data integration from Sanity, schema creation, and displaying data using queries in a Next.js application. Each section is tailored based on the provided code images, with detailed explanations of their functionality.

Template Choice

For this project, I initially opted for Template 6 but later switched to Template 3 due to its simplicity and ease of use.

1. Custom Migration Code

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



```
38 async function importData() {
39   console.log('migrating data please wait...');
40
41   // API endpoint containing car data
42   const response = await axios.get('https://template-03-api.vercel.app/api/products');
43   const products = response.data.data;
44   console.log('products --> ', products);
45
46   for (const product of products) {
47     let imageRef = null;
48     if (product.image) {
49       imageRef = await uploadImageToSanity(product.image);
50     }
51
52     const sanityProduct = {
53       _type: 'product',
54       productName: product.productName,
55       category: product.category,
56       price: product.price,
57       inventory: product.inventory,
58       colors: product.colors || [], // Optional, as per your schema
59       status: product.status,
60       description: product.description,
61       image: imageRef ? {
62         _type: 'image',
63         asset: {
64           _type: 'reference',
65           _ref: imageRef,
66         }
67       } : null
68     };
69   }
70 }
```

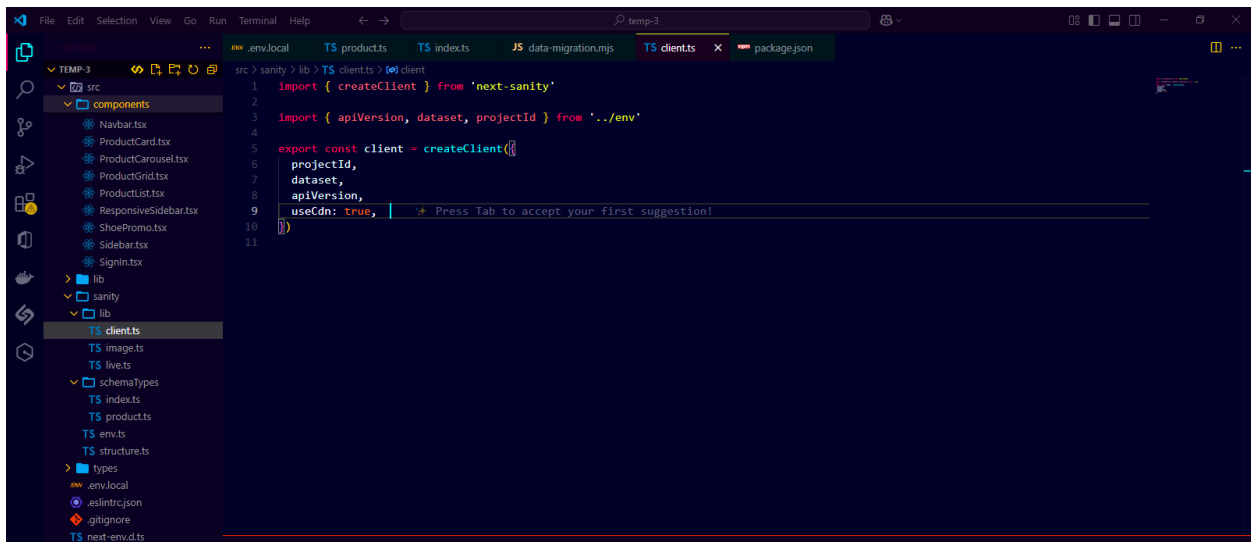
Breakdown of Steps

- **API Setup:**
 - Connects to Sanity's API using the project ID and dataset.
 - Authenticates using an API token sourced from environment variables for security.
- **Fetching Data:**
 - Asynchronously retrieves data from an external API and handles image uploads using the uploadImageToSanity function.
- **Mapping and Formatting:**
 - Data is structured to be compatible with the Project schema.

- **Error Handling:**
 - Logs any errors that occur during migration to ensure the process doesn't crash.

2. Client Page Code

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

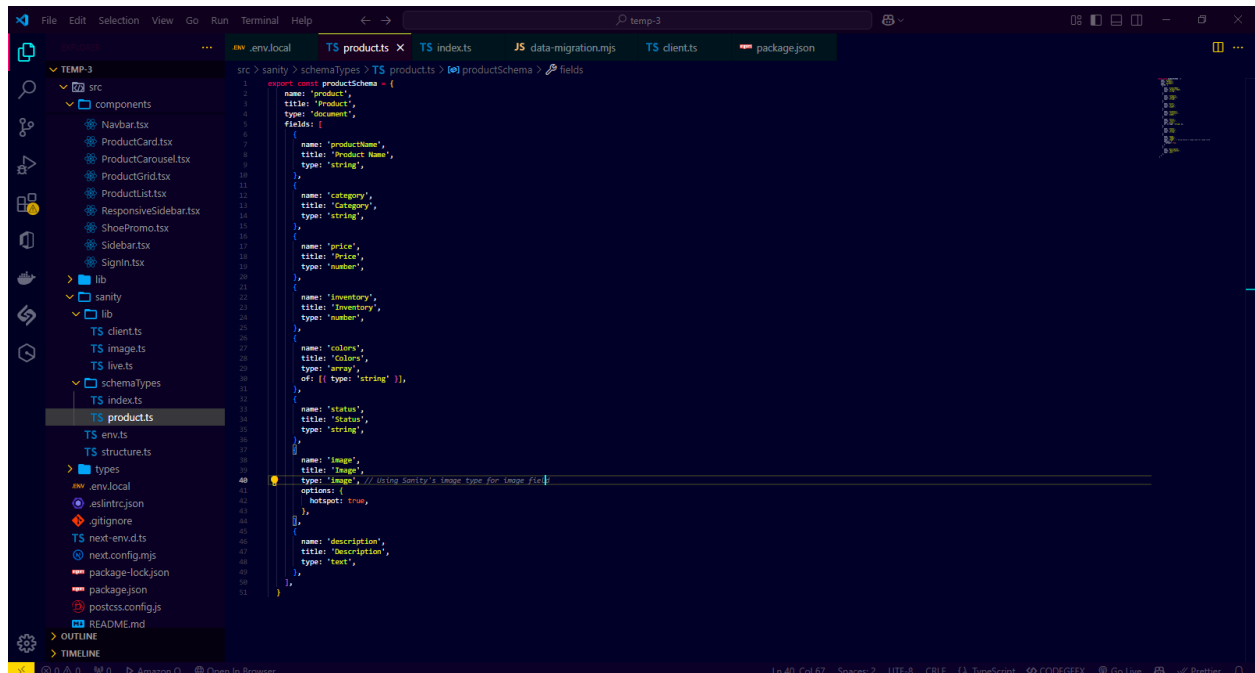


```
src > sanity > lib > TS clients > TS client
1 import { createClient } from 'next-sanity'
2
3 import { apiVersion, dataset, projectId } from '../env'
4
5 export const client = createClient({
6   projectId,
7   dataset,
8   apiVersion,
9   useCdn: true,
10 })
11
```

- **GROQ Query to Fetch Data:**
 - The `getServerSideProps` function uses a GROQ query to fetch data from Sanity during server-side rendering (SSR). This ensures that data is prefetched 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 furniture cards.
- **Dynamic Routing:**
 - The code includes dynamic routing links for individual furniture 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.

3. Schema Code

The schema defines the structure of the content in Sanity.



Key Aspects

- **Schema Fields:** Includes fields for product name, category, price, inventory, colors, status, and image.
- **Validation:** Rules to ensure all necessary fields are filled correctly, such as requiring a non-empty product name.

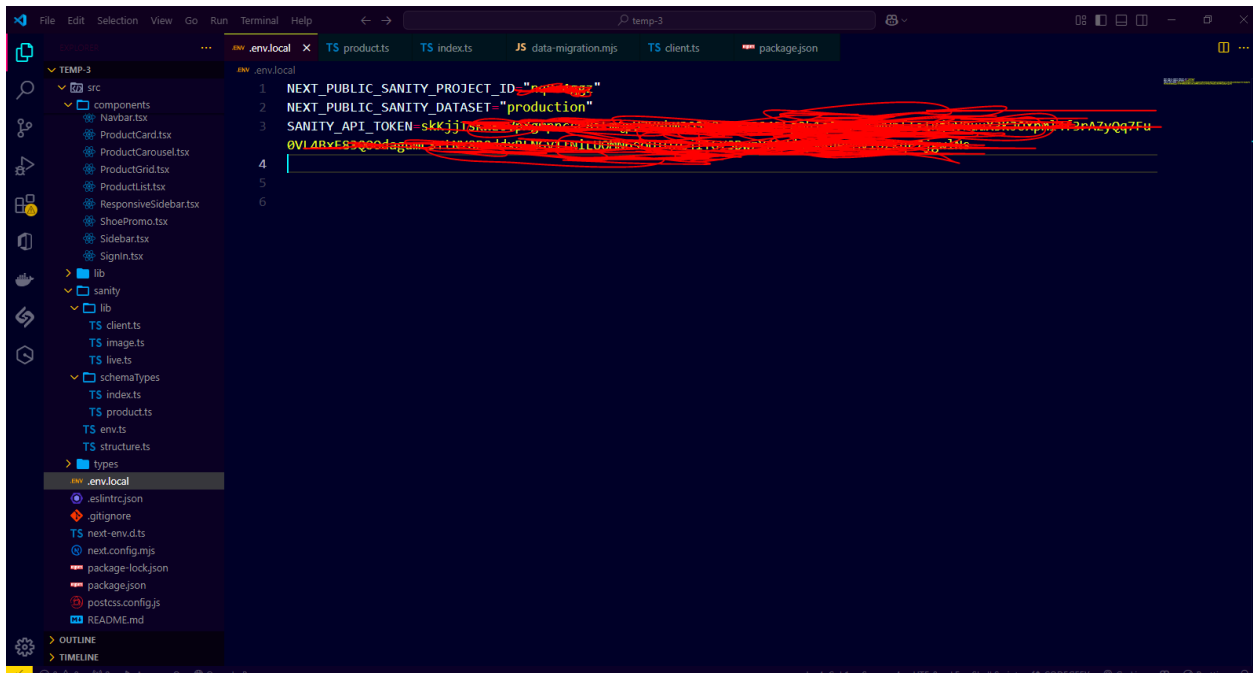
4. Sanity Studio Fields

In the Sanity Studio, product entries can be managed easily. Each product form includes:

- **Product Name:** A string field to name the product.
- **Category:** A dropdown or input field for categorizing the product (e.g., "Women's Shoes").
- **Price:** Numeric input for the product's cost.
- **Inventory:** Number field indicating the available stock.
- **Colors:** Array field to add multiple color options.
- **Status:** A dropdown to indicate if the product is available, sold out, etc.
- **Image:** An image upload area to add a product picture.

5. Environment Variables:

The `.env.local` file includes sensitive configurations for the Nike application.



The screenshot shows a code editor with the `.env.local` file open. The file contains the following configuration values:

```
1 NEXT_PUBLIC_SANITY_PROJECT_ID="next-apps"
2 NEXT_PUBLIC_SANITY_DATASET="production"
3 SANITY_API_TOKEN="skkjjtsn...forAZyqg7Fu"
4
5
6
```

The values for `SANITY_API_TOKEN` and the project ID are redacted with red scribbles. The editor's sidebar on the left shows the file structure, including `src`, `components`, `lib`, `sanity`, `types`, and `.env.local`.

Sanity Configuration:

SANITY_PROJECT_ID: The unique identifier for the Sanity project.

SANITY_DATASET: Specifies the dataset (e.g., production or development).

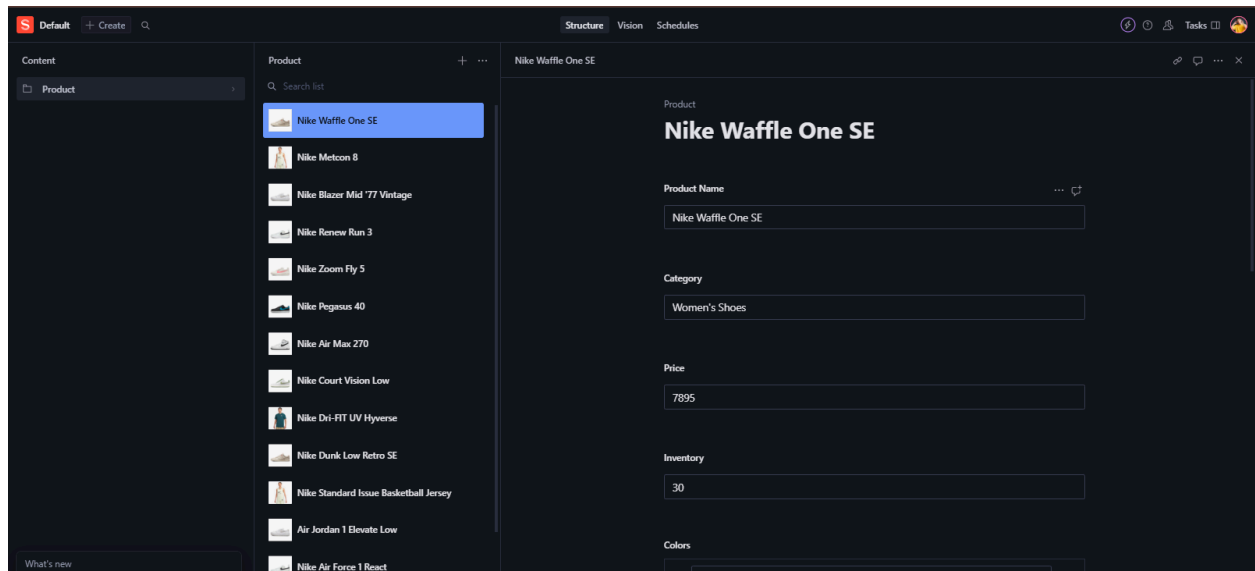
API Security:

SANITY_API_TOKEN: A secure token used to authenticate API calls to Sanity. 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.

6.Sanity Product Schema:



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

1. Title Field.
2. Category Field.
3. Price Field.
4. Inventory Field.
5. Colors Fields.
6. Status Field.
7. Image Field.
8. Description Field.

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,  
  category,  
  price,  
  inventory,  
  status,  
  image,  
  description,  
}
```

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 focused on setting up the Nike Template 3 backend and integrating the data migration pipeline. The following key tasks were accomplished:

- Successfully implemented custom migration code to transfer data from Sanity.
- Designed a structured schema in Sanity to ensure data integrity.
- Rendered product data dynamically on the client-side using server-side rendering.
- Configured environment variables securely to manage sensitive information.
- Established a visually appealing front page for user engagement.