

Day 3:

API Integration And Data Migration

API Overview: The APIs provided for Day 3 will allow to populate Sanity CMS with relevant data. These APIs serve as a guidance tool for validating schemas and ensuring that data is structured according to the needs of the marketplace. Below are some key points regarding these APIs:

- **Read-Only APIs:** The provided APIs are read-only and cannot modify data in external systems. They serve as mock data sources or examples to help populate your CMS.
- **Custom Integration:** Students are free to integrate other APIs from platforms like Shopify, WooCommerce, and Magento if they are already working with such platforms, or they can integrate custom APIs as needed.
- **Data Migration Methods:** Students can manually import data into Sanity CMS or use the APIs to automatically populate product listings, user data, and orders. Both approaches are useful in different scenarios, depending on the scale and complexity of the project.

Steps for Integration and Migration:

1. Set Up API Integration:

- Configure API keys and authentication methods for third-party services like payment processors and shipment trackers.
- Integrate external APIs into the Next.js frontend using functions like `getServerSideProps()` or `getStaticProps()` to fetch and display data dynamically.

2. Create Sanity CMS Schemas:

- Define the content types and structures in Sanity CMS, including product listings, orders, and customer data.
- Use the Sanity CMS interface to create documents and fields that match the API data structure.

3. Migrate Data into Sanity CMS:

- Use scripts or third-party tools to import data from APIs or external eCommerce platforms into Sanity CMS.
- Ensure that the data is aligned with the templates used in the marketplace and validate for accuracy.

4. Test and Validate Integration:

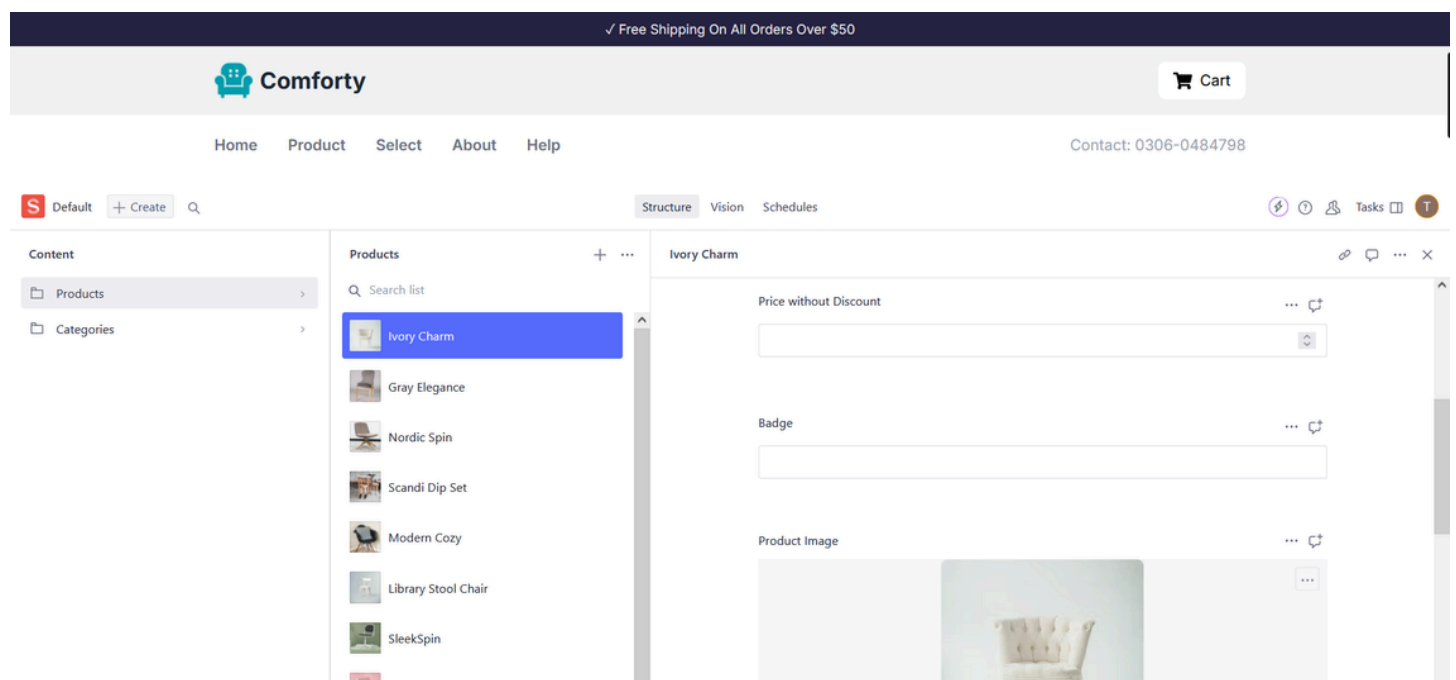
- Verify that all integrated data flows smoothly from the APIs into Sanity CMS and that the frontend displays the information correctly.
- Perform validation checks to ensure that no data is missing and that the schemas are consistent.

5. Handle Edge Cases and Errors:

- Ensure proper error handling for API requests, such as timeouts, missing data, or incorrect formats.
- Implement fallback strategies in case the API data is unavailable or incomplete.

Sanity Migration:

In this i migrate my Furniture website into the sanity with mokeapi. The migrated view of sanity are as under:



Sanity Code For Migration:

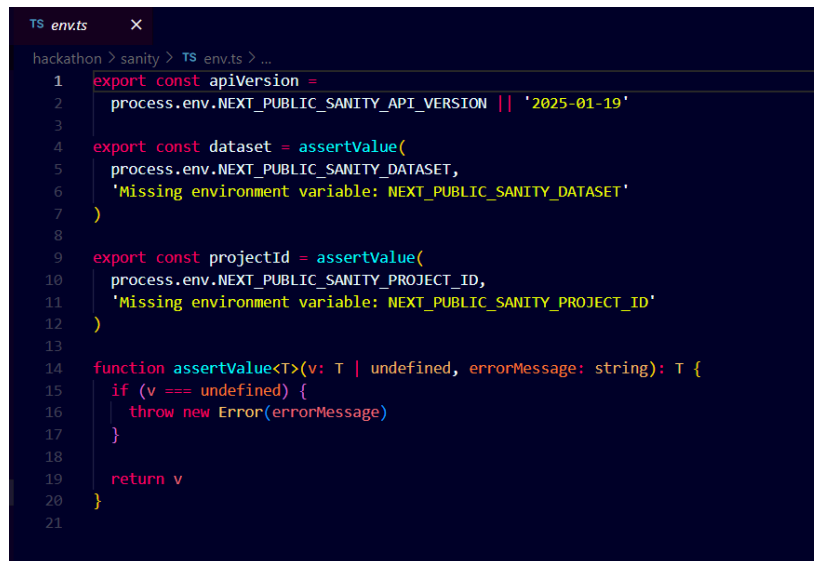
The code for sanity migration has few files which are viewed one by one 😊

- **env.ts:**

This file is generally used to manage environment variables in a TypeScript project. It abstracts sensitive or configurable data like API keys, database URLs, and secrets, ensuring security and flexibility.

Purpose:

- Centralize sensitive configuration details.
- Simplify changes to environment variables during development, staging, and production.



```
TS env.ts X
hackathon > sanity > TS env.ts > ...
1 export const apiVersion =
2   process.env.NEXT_PUBLIC_SANITY_API_VERSION || '2025-01-19'
3
4 export const dataset = assertValue(
5   process.env.NEXT_PUBLIC_SANITY_DATASET,
6   'Missing environment variable: NEXT_PUBLIC_SANITY_DATASET'
7 )
8
9 export const projectId = assertValue(
10  process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
11  'Missing environment variable: NEXT_PUBLIC_SANITY_PROJECT_ID'
12 )
13
14 function assertValue<T>(v: T | undefined, errorMessage: string): T {
15   if (v === undefined) {
16     throw new Error(errorMessage)
17   }
18   return v
19 }
20
21
```

- **structure.ts:**

This file typically defines the structure of the Sanity Studio desk. It customizes how documents and categories appear in the Sanity CMS UI.

Purpose:

- Organize and customize the Sanity Studio UI.
- Group schemas (e.g., categories, products) logically for better navigation.



```
structure.ts X
hackathon > sanity > TS structure.ts > ...
1 import type {StructureResolver} from 'sanity/structure'
2
3 // https://www.sanity.io/docs/structure-builder-cheat-sheet
4 export const structure: StructureResolver = (S) =>
5   S.list()
6     .title('Content')
7     .items(S.documentTypeListItems())
8
```

Sannity/SchemaTypes:

- **categories.ts:**

This file defines the schema for categories in Sanity CMS. Categories are often used to

organize products or content.

Purpose:

- Define the fields (e.g., title, slug, description) and structure of category data in Sanity CMS.
- Enable linking categories to other schemas like products.

```
TS categories.ts X
hackathon > sanity > schemaTypes > TS categories.ts > [e] categorySchema > name
1  import { defineType } from "sanity";
2
3  export const categorySchema = defineType({
4    name: 'categories',
5    title: 'Categories',
6    type: 'document',
7    fields: [
8      {
9        name: 'title',
10       title: 'Category Title',
11       type: 'string',
12     },
13     {
14       name: 'image',
15       title: 'Category Image',
16       type: 'image',
17     },
18     {
19       title: 'Number of Products',
20       name: 'products',
21       type: 'number',
22     }
23   ],
24 });
```

- **index.ts:**

This file consolidates all individual schema files (e.g., categories.ts, product.ts) and exports them as a single array to be used in Sanity CMS.

Purpose:

- Serve as the entry point for all schema definitions.
- Ensure that all schemas are registered and available in Sanity Studio.

```
TS index.ts X
hackathon > sanity > schemaTypes > TS index.ts > ...
1  import { type SchemaTypeDefinition } from "sanity";
2  import { productSchema } from "../products";
3  import { categorySchema } from "../categories";
4
5  export const schema: { types: SchemaTypeDefinition[] } = {
6    types: [productSchema, categorySchema],
7  };
```

- **product.ts:**

This file defines the schema for products in Sanity CMS. Products typically include details like name, price, description, category, and images.

Purpose:

- Define the structure and fields for product data.
- Allow linking products to other schemas like categories.



```
TS products.ts X
hackathon > sanity > schemaTypes > TS products.ts > [e] products
1  import { defineType } from "sanity";
2
3  export const productSchema = defineType({
4    name: "products",
5    title: "Products",
6    type: "document",
7    fields: [
8      {
9        name: "title",
10       title: "Product Title",
11       type: "string",
12     },
13     {
14       name: "price",
15       title: "Price",
16       type: "number",
17     },
18     {
19       title: "Price without Discount",
20       name: "priceWithoutDiscount",
21       type: "number",
22     },
23     {
24       name: "badge",
25       title: "Badge",
26       type: "string",
27     },
28     {
29       name: "image",
30       title: "Product Image",
31       type: "image",
32     },
33     {
34       name: "category",
35       title: "Category",
36       type: "reference",
37       to: [{ type: "categories" }],
38     },
39   ],
40  });
```

Sanity/lib:

- **client.ts:**

This file typically initializes and exports the Sanity client instance, which is used to interact with the Sanity CMS backend.

Purpose:

- Provide a reusable Sanity client for querying and managing data.
- Ensure consistent configuration across the project.

```
TS client.ts X
hackathon > sanity > lib > TS client.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, // Set to false if statically generating
10 })
11
```

Workflow and Integration:

1. **Environment (env.ts):** Configures project settings such as API credentials.
2. **Schema Structure (structure.ts):** Defines the layout for content types in Sanity Studio.
3. **Schemas (categories.ts and product.ts):** Establish content types and relationships in the CMS.
4. **Index (index.ts):** Bundles schemas for Sanity Studio configuration.
5. **Client (client.ts):** Manages API interactions between the Next.js app and Sanity CMS.

Conclusion:

Focuses on the critical task of integrating APIs and migrating data into Sanity CMS. By the end of the day, i be able to connect marketplace's backend with third-party APIs and migrate data effectively