

# Day 3: API Integration and Data Migration Report

## (Prepared by Abdul Rafay)

### Objectives

- Integrate APIs into a Next.js project.
- Migrate data from APIs into Sanity CMS.—
- Validate and align schemas with data sources.

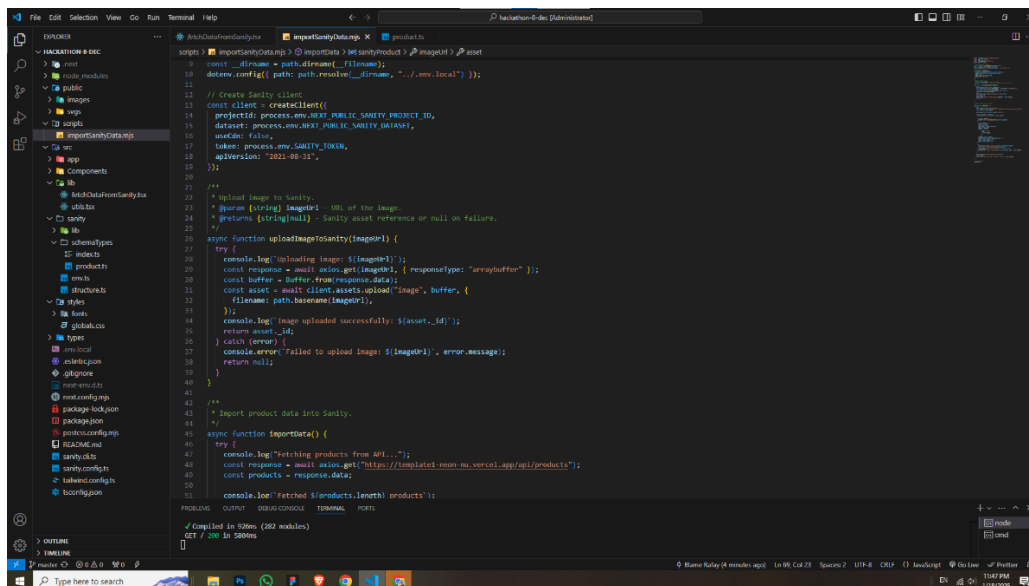
### Key Learning Outcomes

- Understand API integration techniques.
- Explore data migration processes using provided or external APIs.
- Customize schemas for Sanity CMS to ensure compatibility.

## 1. Fetching Data from External APIs

The first step was to fetch data from external APIs. In my Next.js application, where the frontend code was already implemented, I created a folder called **scripts**. Within this folder, I added a file named **importSanityData.mjs**, where I wrote the necessary code to fetch data from an external API. After testing the script both in the application and with Postman API, I successfully fetched the data.

### Example of fetching data from an external API:

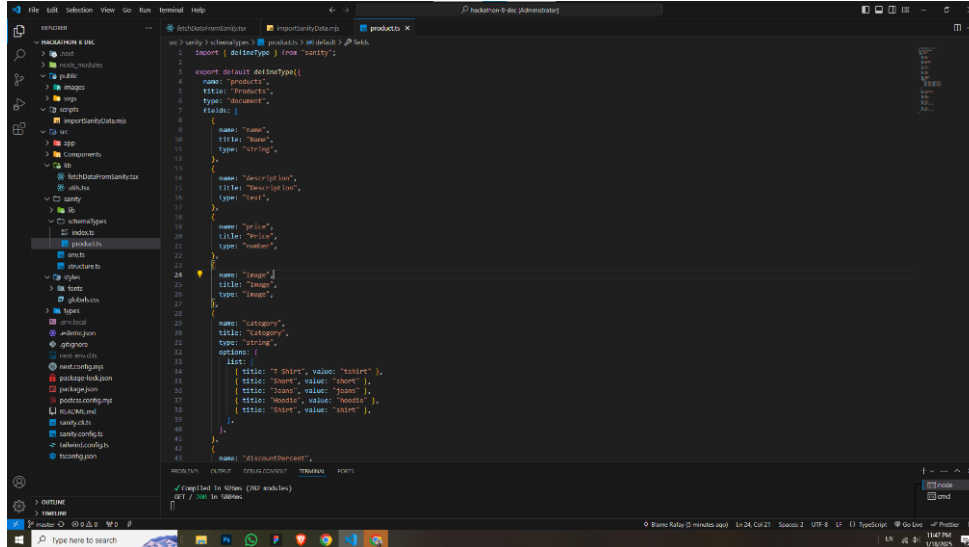


```
1  scripts importSanityData.mjs > @ import axios from 'axios';
2  const __dirname = path.resolve(__filename);
3  dotenv.config({ path: path.resolve(__dirname, '../env.local') });
4
5  // Create Sanity client
6  const client = createClient({
7    projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
8    dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
9    useCdn: false,
10    token: process.env.SANITY_TOKEN,
11    apiVersion: "2021-03-11",
12  });
13
14  /**
15   * Upload image to Sanity.
16   * @param {string} imageUrl URL of the image.
17   * @returns {string|null} - Sanity asset reference or null on failure.
18   */
19  async function uploadImageToSanity(imageUrl) {
20    try {
21      console.log('Uploading image: ', imageUrl);
22      const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
23      const buffer = Buffer.from(response.data);
24      const asset = await client.assets.upload('image', buffer, {
25        filename: path.basename(imageUrl),
26      });
27      console.log('Image uploaded successfully: ', asset._id);
28      return asset._id;
29    } catch (error) {
30      console.error('Failed to upload image: ', imageUrl, error.message);
31      return null;
32    }
33  }
34
35  /**
36   * Import product data into Sanity.
37   */
38  async function importData() {
39    try {
40      console.log('Fetching products from API...');
41      const response = await axios.get('https://tapi1.mon-nu.vercel.app/api/products');
42      const products = response.data;
43
44      console.log('Fetched ', products.length, ' products');
45    } catch (error) {
46      console.error('Error fetching products: ', error.message);
47    }
48  }
49
50  // Run the script
51  importData();
```

## 2. Comparing API Data with Sanity Schema

Once the data was fetched, I compared the data structure from the API with the Sanity schema. This step was crucial in ensuring the data from the API was compatible with the schema set up in Sanity CMS. The goal was to ensure that the product data could be stored and managed in Sanity without conflicts.

### Sanity Schema Example (for products):



## 3. Setting Up the Environment

For the migration to work, the Sanity project needed to be correctly configured. I generated a token from Sanity's official website and added the project ID, dataset name, and token into the `.env.local` file to authenticate requests.



## 4. Modifying `package.json`

To streamline the process of running the migration script, I added a custom command to the `package.json` file. This allows me to run the migration script with a simple command in the terminal.

```

package.json X
package.json > {} dependencies
1  {
2    "name": "hackathon-2",
3    "version": "0.1.0",
4    "private": true,
5    "scripts": {
6      "dev": "next dev",
7      "build": "next build",
8      "start": "next start",
9      "lint": "next lint",
10     "import-data": "node scripts/importSanityData.mjs"
11   },
12   "dependencies": {}

```

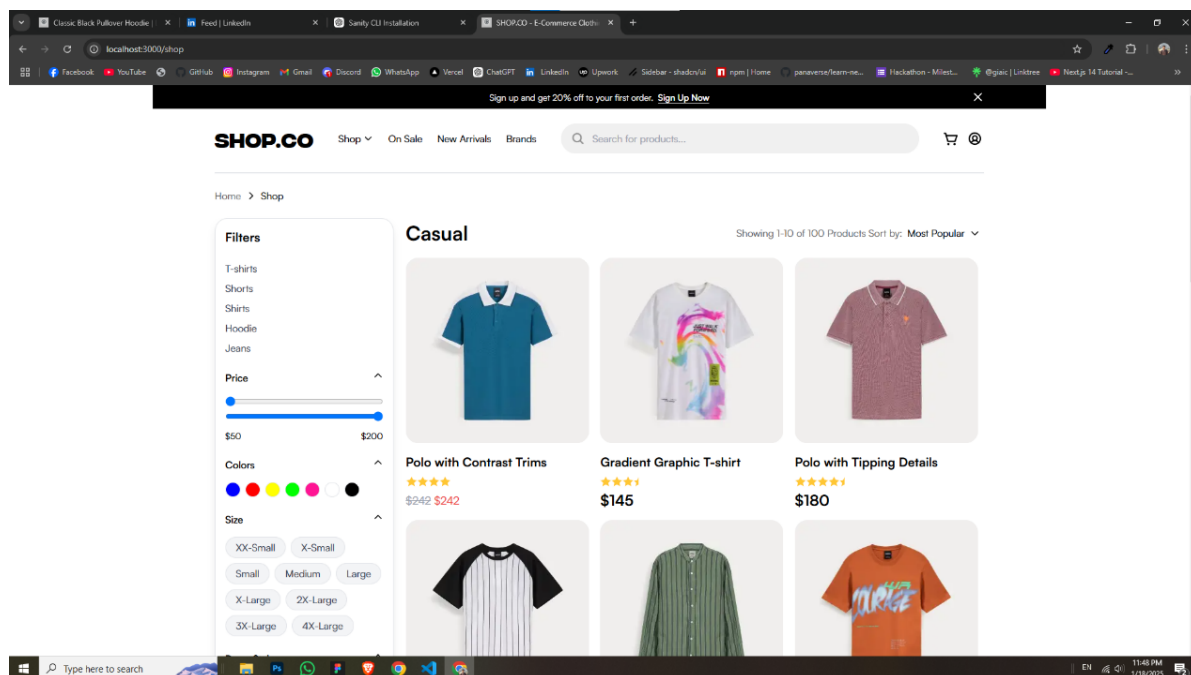
## 5. Importing Data into Sanity

With everything configured, I ran the migration script to import the data into Sanity. The following command triggered the data migration:

`npm run import-data`

## 6. Displaying Data in the Frontend

Finally, I mapped the migrated product data to display it on the frontend. Using components like a product card, I showcased the product details on the website, including the product name, description, price, and image



## Conclusion

Through this process, I successfully automated the migration of data from an external API to Sanity CMS. The key steps included:

1. Fetching data from external APIs.
2. Comparing API data with Sanity schema to ensure compatibility.
3. Writing an automated migration script.
4. Configuring the environment with the necessary tokens and project settings.
5. Running the migration script to insert data into Sanity.
6. Displaying the migrated data in the frontend.

This process helped me understand the intricacies of working with APIs, data migration, and CMS integration while ensuring data consistency and schema alignment.

## Key Achievements

- API Understanding: ✓
  - Schema Validation: ✓
  - Automated Data Migration: ✓
- 

Made by **Abdul Rafay**

(Roll No:299941)