

# Day 3 - API Integration Report – Q-Commerce

## Introduction

The purpose of this report is to document the process of API integration, schema adjustments, and migration steps performed as part of the Q-commerce project. This task focused on ensuring seamless data handling between the backend and frontend by integrating two APIs to populate the food and chef data dynamically.

## API Integration Process

### 1. Overview:

We integrated two provided APIs into our Q-commerce application to populate the food and chef marketplace data. Below are the API endpoints utilized?

- **API URL (Foods Data):**

`https://sanity-nextjs-rouge.vercel.app/api/foods`

- **API URL (Chefs Data):**

`https://sanity-nextjs-rouge.vercel.app/api/chefs`

These APIs provided data such as food names, images, prices, chef details, and Descriptions. This data was migrated to Sanity CMS for structured storage and later fetched for frontend display.

### 2. Steps Taken:

#### Environment Variables:

Sensitive data was stored securely in `.env.local` to avoid hardcoding values. Key environment variables used:

- `NEXT_PUBLIC_SANITY_PROJECT_ID=[Insert Project ID]`
- `NEXT_PUBLIC_SANITY_DATASET=[Insert Dataset]`
- `NEXT_PUBLIC_SANITY_TOKEN=[Insert Token]`

## **Sanity Client Creation:**

Configured the Sanity client using the project ID and dataset. Handled sensitive data securely by using environment files.

## **Data Fetching:**

GROQ queries were used to fetch data from Sanity CMS. The following fields were fetched:

- For Food: id, title, food Image, price, ingredients, description
- For Chefs: id, name, profile Image, specialty, bio

## **Data Processing:**

Processed the fetched data to align with frontend requirements. Generated unique image URLs dynamically for frontend display.

## **Sanity Documentation Creation:**

Created schemas in Sanity CMS to align with the API structure for both foods and chefs. Key fields included:

- **Food Schema:** title, price, ingredients, foodImage, description, tags
- **Chef Schema:** name, specialty, profileImage, bio

## **Error Handling:**

Added error handling during API calls and data fetching. Errors were logged for debugging, and user-friendly messages were displayed in the frontend.

### **3. Migration Steps and Tools Used:**

#### **Migration Script:**

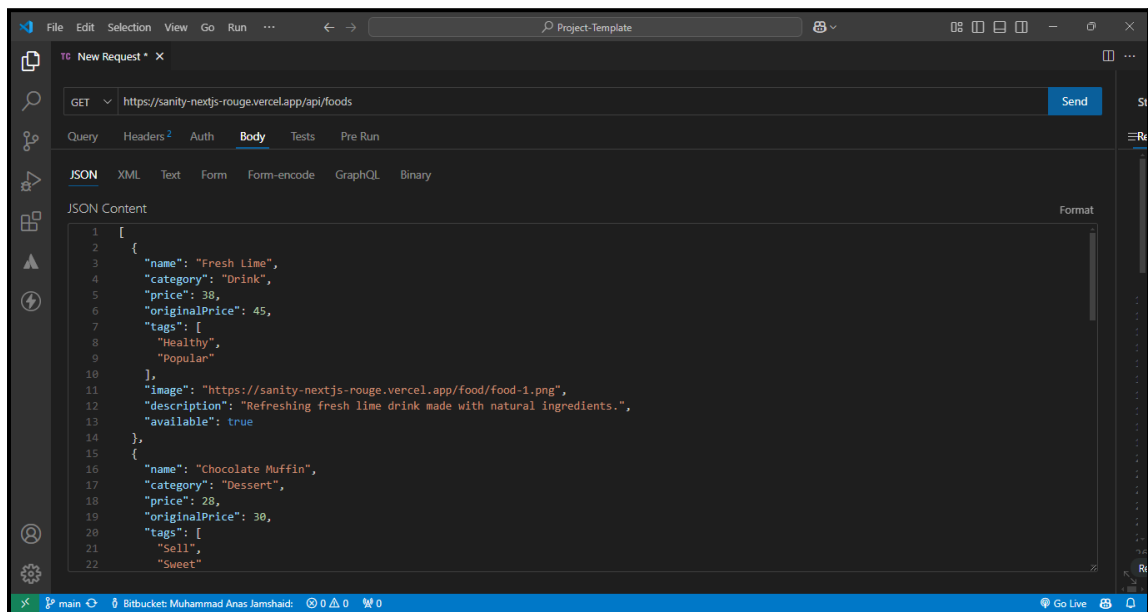
A migration script was created to fetch data from the provided APIs (/foods and /chefs) and populate Sanity CMS programmatically. Data accuracy was ensured during the migration process by validating the input fields.

## Sanity Schema Adjustments:

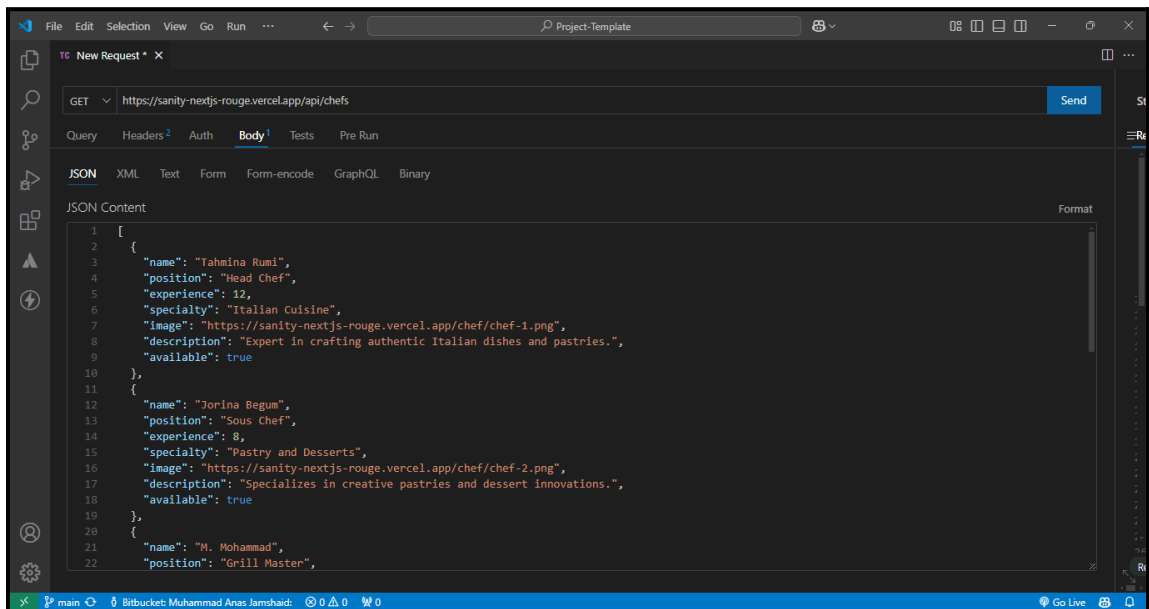
Adjusted the schema in Sanity CMS to match the structure of the API data. This includes adding necessary fields like price, ingredients, bio, and images for both food items and chefs to ensure seamless integration.

## Screenshots:

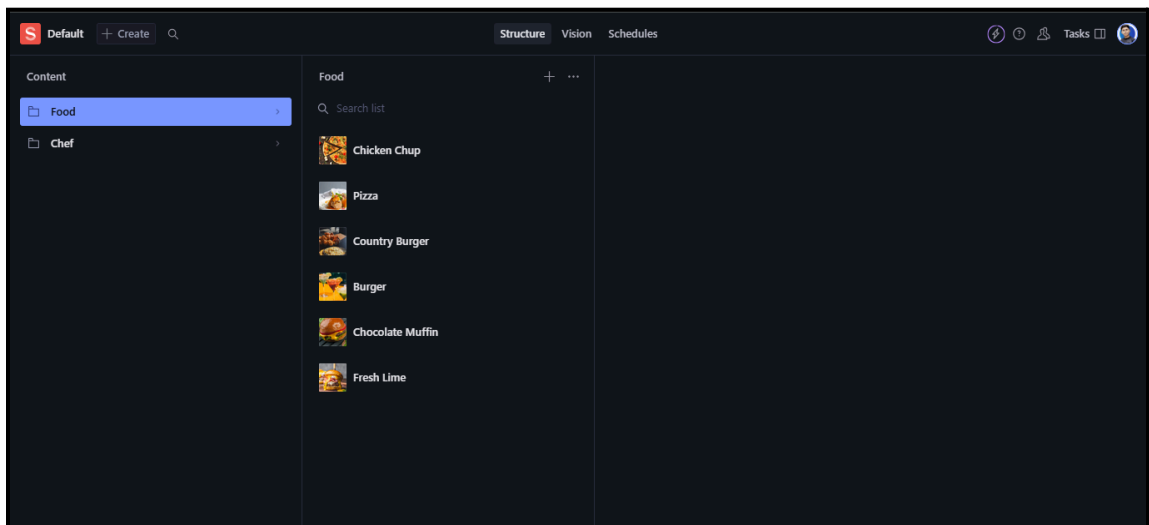
### 1. Food API Calls:



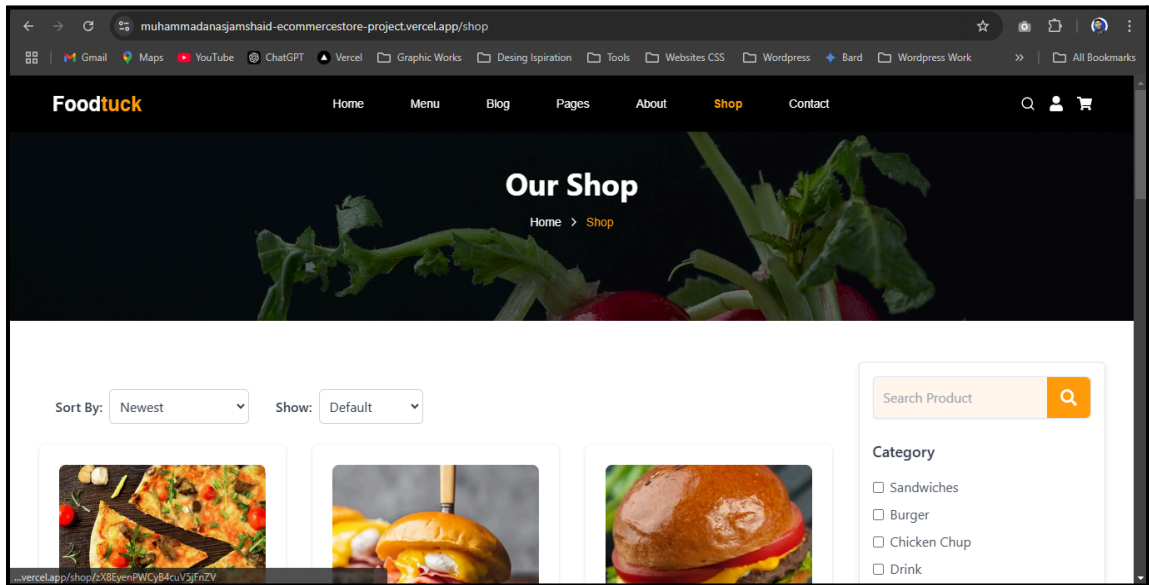
### 2. Chefs API Calls:



## 2. Populated Sanity CMS Fields:

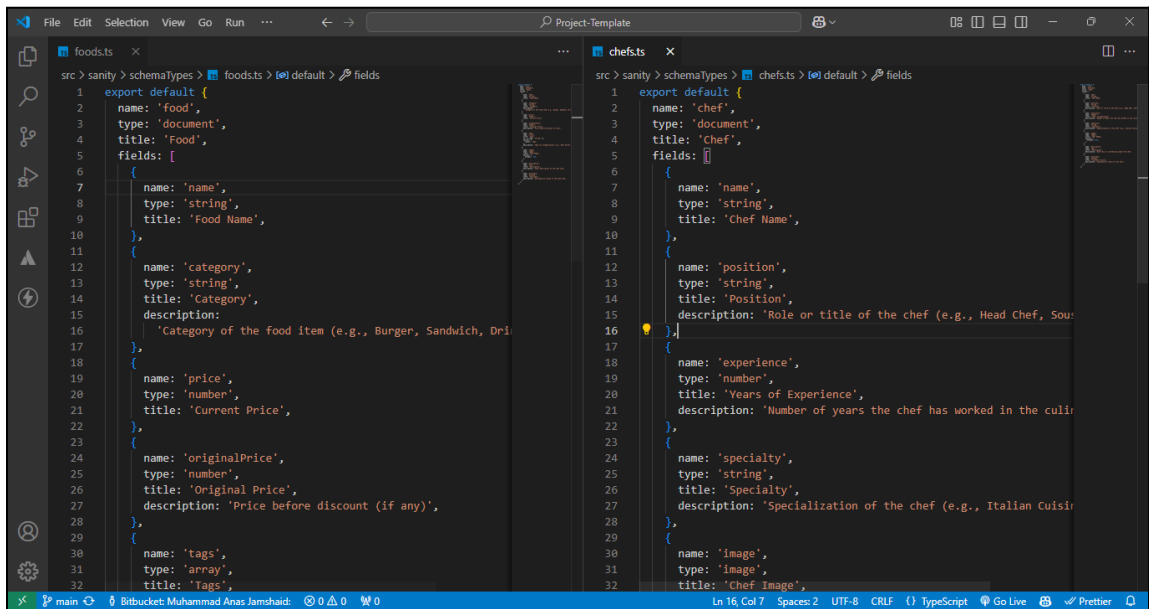


## 3. Data Successfully Displayed on Frontend:

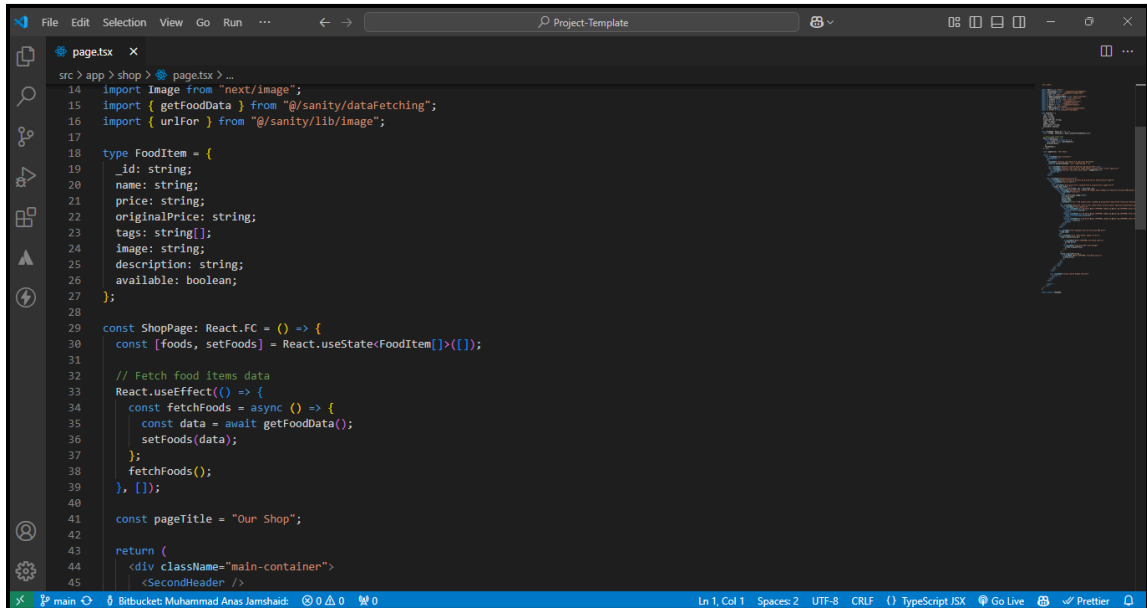


## Code Snippets

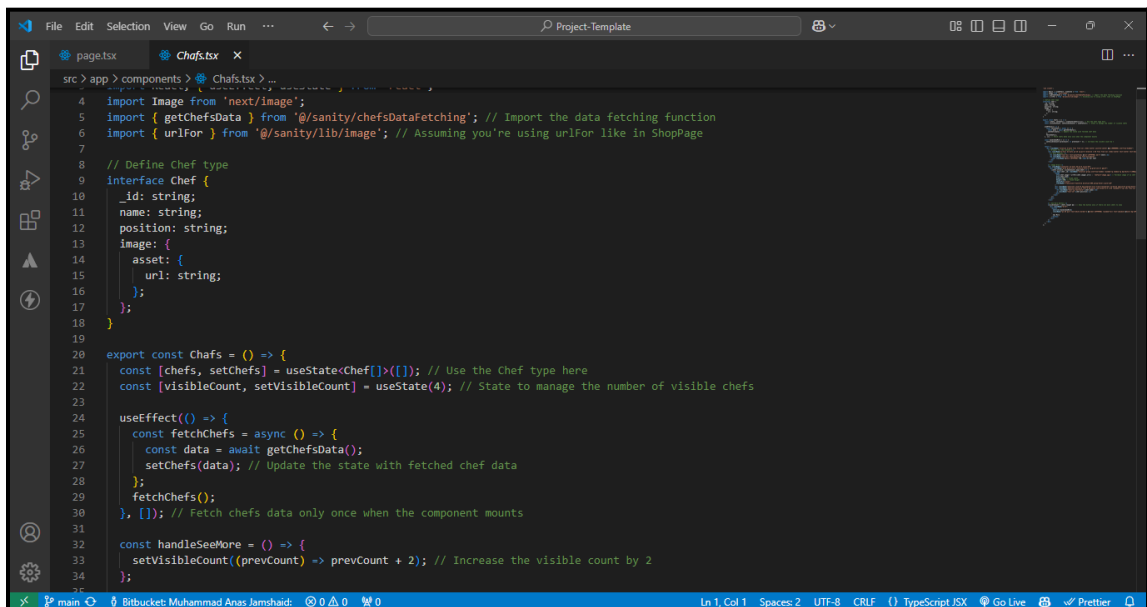
### 1.Sanity



### 2.Frontend Integration:



```
src > app > shop > page.tsx > ...
14 import Image from 'next/image';
15 import { getFoodData } from '@sanity/datafetching';
16 import { urlFor } from '@sanity/lib/image';
17
18 type FoodItem = {
19   _id: string;
20   name: string;
21   price: string;
22   originalPrice: string;
23   tags: string[];
24   image: string;
25   description: string;
26   available: boolean;
27 };
28
29 const ShopPage: React.FC = () => {
30   const [foods, setFoods] = React.useState<FoodItem[]>([]);
31
32   // Fetch food items data
33   React.useEffect(() => {
34     const fetchFoods = async () => {
35       const data = await getFoodData();
36       setFoods(data);
37     };
38     fetchFoods();
39   }, []);
40
41   const pageTitle = "Our Shop";
42
43   return (
44     <div className="main-container">
45       <SecondHeader />
46     </div>
47   );
48 }
```



```
src > app > components > Chafs.tsx > ...
4 import Image from 'next/image';
5 import { getChefsData } from '@sanity/chefsDataFetching'; // Import the data fetching function
6 import { urlFor } from '@sanity/lib/image'; // Assuming you're using urlFor like in ShopPage
7
8 // Define Chef type
9 interface Chef {
10   _id: string;
11   name: string;
12   position: string;
13   image: {
14     asset: {
15       url: string;
16     };
17   };
18 }
19
20 export const Chafs = () => {
21   const [chefs, setChefs] = useState<Chef[]>([]); // Use the Chef type here
22   const [visibleCount, setVisibleCount] = useState(4); // State to manage the number of visible chefs
23
24   useEffect(() => {
25     const fetchChefs = async () => {
26       const data = await getChefsData();
27       setChefs(data); // Update the state with fetched chef data
28     };
29     fetchChefs();
30   }, []); // Fetch chefs data only once when the component mounts
31
32   const handleSeeMore = () => {
33     setVisibleCount((prevCount) => prevCount + 2); // Increase the visible count by 2
34   };
35 }
```

### 3.MigrationScript:

```

1 import { createClient } from '@sanity/client';
2 import axios from 'axios';
3 import dotenv from 'dotenv';
4 import { fileURLToPath } from 'url';
5 import path from 'path';
6
7 // Load environment variables from .env.local
8 const __filename = fileURLToPath(import.meta.url);
9 const __dirname = path.dirname(__filename);
10 dotenv.config({ path: path.resolve(__dirname, '../.env.local') });
11
12 // Create Sanity client
13 const client = createClient({
14   projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
15   dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
16   useCdn: true,
17   token: process.env.SANITY_API_TOKEN,
18   apiVersion: '2021-08-31',
19 });
20
21 async function uploadImageToSanity(imageUrl) {
22   try {
23     console.log(`Uploading image: ${imageUrl}`);
24     const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
25     const buffer = Buffer.from(response.data);
26     const asset = await client.assets.upload('image', buffer, {
27       filename: imageUrl.split('/').pop(),
28     });
29     console.log(`Image uploaded successfully: ${asset._id}`);
30     return asset._id;
31   } catch (error) {
32     console.error('Failed to upload image:', imageUrl, error);
33     return null;
34   }
35 }
36
37 async function importData() {
38   try {
39     console.log('Fetching food, chef data from API...');
40
41     // API endpoint containing data
42     const $Promise = [];
43     $Promise.push(
44       axios.get('https://sanity-nextjs-rouge.vercel.app/api/foods')
45     );
46     $Promise.push(
47       axios.get('https://sanity-nextjs-rouge.vercel.app/api/chefs')
48     );
49
50     const [foodsResponse, chefsResponse] = await Promise.all($Promise);
51     const foods = foodsResponse.data;
52     const chefs = chefsResponse.data;
53
54     for (const food of foods) {
55       console.log(`Processing food: ${food.name}`);
56
57       let imageRef = null;
58       if (food.image) {
59         imageRef = await uploadImageToSanity(food.image);
60       }
61
62       const sanityFood = {
63         _type: 'food',
64         name: food.name,
65         category: food.category || null,
66         price: food.price,
67         originalPrice: food.originalPrice || null,
68         tags: food.tags || [],
69         description: food.description || '',
70         available: food.available !== undefined ? food.available : true,
71         image: imageRef
72         ? {
73             _type: 'image',
74             asset: {
75               _type: 'reference',
76               _ref: imageRef,
77             },
78           }
79         : null;
80     }
81   } catch (error) {
82     console.error('Error importing data:', error);
83   }
84 }

```