

# **API Integration Report**

## **1. Introduction**

This report outlines the process of integrating content for **Food** and **Chef** schemas within the website. The integration involves the use of a headless CMS (Sanity.io) to manage content, making it easy to integrate and manage the website's dynamic content such as food items and chefs. I will also discuss the migration steps, API calls, and the tools used during the integration.

## **2. Sanity CMS Setup**

**Sanity.io** is a headless content management system (CMS) that allows developers to manage dynamic content (like food items and chefs) and provide APIs to fetch this content. This CMS was integrated into the website to provide a backend that enables easy content management and fetching.

## **3. Schemas Defined in Sanity**

### **3.1 Food Schema**

The **Food Schema** defines the structure of food items that can be displayed on the website. The schema includes fields such as:

- **Name** (string): The name of the food item.
- **Price** (number): The price of the food item.
- **Description** (text): A description of the food item.
- **Category** (string): The category to which the food item belongs.
- **Available** (boolean): Indicates if the food item is available for purchase.
- **Tags** (array): Tags associated with the food item for search and categorization.
- **Image URL** (image): URL of the image for the food item.

### **3.2 Chef Schema**

The **Chef Schema** is designed to store information related to chefs, such as:

- **Name** (string): The name of the chef.
- **Bio** (text): A short bio about the chef.
- **Experience** (string): The chef's experience level or other relevant details.
- **Image URL** (image): URL of the chef's image.
- **Specialties** (array): An array of food specialties the chef is known for.

These schemas are set up within Sanity to allow for easy content creation and retrieval via the API.

## 4. API Integration Process

The API integration process follows these steps:

### 4.1 Creating the API Client

A custom client (`env.local`) is created to interact with Sanity's API. It handles queries and connects with the backend using a token to authenticate the requests. Here's an example of how the client is set up:

```
hack_2 > studio-hackathon-2-template-9 > $ .env.local
1 NEXT_PUBLIC_SANITY_PROJECT_ID=wp3r5brv
2 NEXT_PUBLIC_SANITY_DATASET=production
3 SANITY_API_TOKEN=sKAFBg55GrkFMOUkcSKYAh6U60rpgZsr45et0eCvUySyYakpC0S7Eht24YudKrOVy5QyMKhK1z9lksqmuMVo0V8yX0I46rcDjzbG9XB4cZaLxkouVwiIQPfr88vwyJ
  BTNLW9nJUJb66sKeDLLCoB1ze9lBtRR4jyDW8JN8xYs8CSbsQW1W2
4
```

### 4.2 Fetching Food Content

To fetch food data, we use a GROQ query (Sanity's query language) to retrieve the required fields for food items. Here's the query for fetching food items by their `slug`

```
// Query to fetch a single product based on the slug
const query = `[_type == "food" && slug.current == $slug][0]{
  name, category,
  price, tags, available,
  "description": description,
  "image_url": image.asset->url
}`;
```

The query fetches the details of a specific food item based on the `slug`. The `slug` is a unique identifier for each food item, allowing the website to display individual product details.

### 4.3 Fetching Chef Content: Similarly, the query for fetching chef details is:

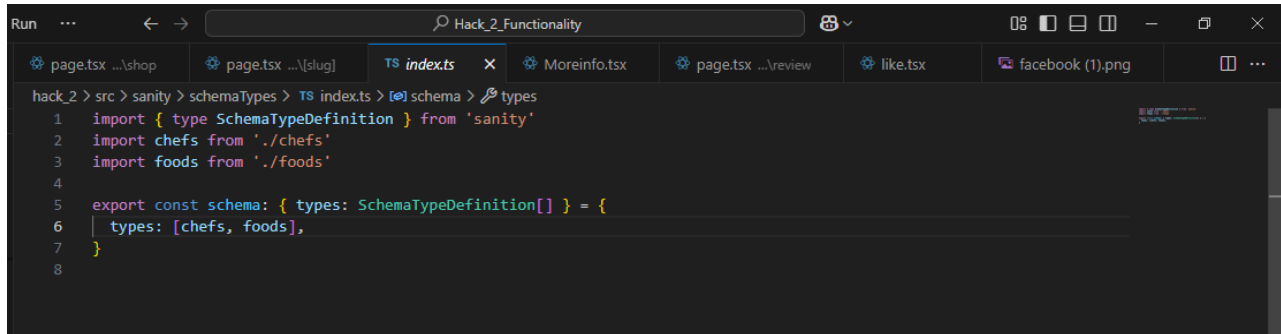
```
19
20 // Query to fetch a single product based on the slug
21 const query = `[_type == "chef" && slug.current == $slug][0]{
22   name, bio,
23   experience,
24   specialities,
25   "image_url": image.asset->url
26 }`;
27
28 // Product detail page
```

This query retrieves information about a specific chef using their `slug`, such as their name, bio, experience, image, and specialties.

## 4.4 Using the API Provided by Sir Mubashir

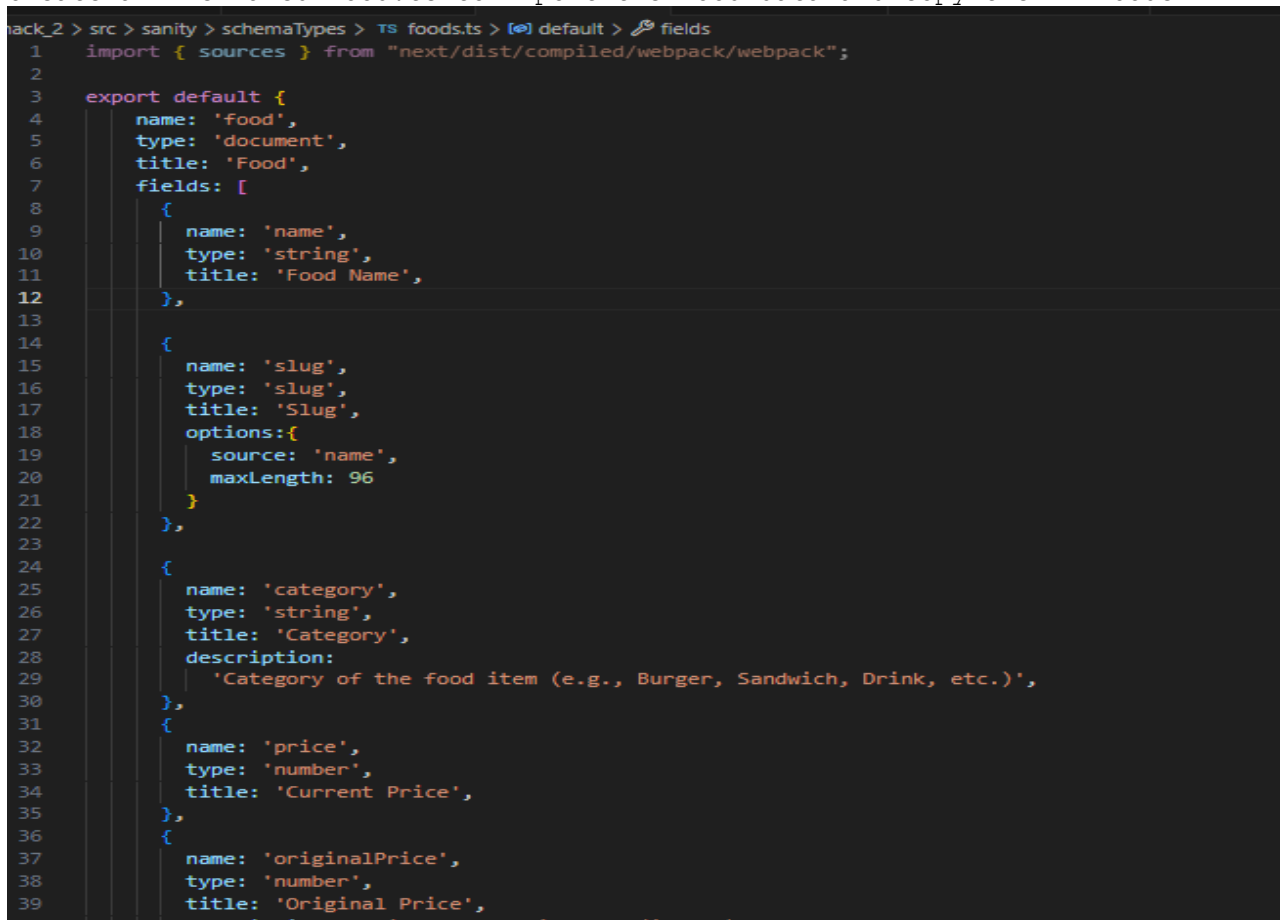
The data for **Food** and **Chef** content is fetched from the provided API link. No manual data entry is needed, as the data is fetched directly from the external source.

In `INDEX.ts` file in Sanity folder define the `schemaTypes`



```
Run ... Hack_2_Functionality
page.tsx ...shop page.tsx ...[slug] TS index.ts x Moreinfo.tsx page.tsx ...review like.tsx facebook (1).png ...
hack_2 > src > sanity > schemaTypes > TS index.ts > schema > types
1 import { type SchemaTypeDefinition } from 'sanity'
2 import chefs from './chefs'
3 import foods from './foods'
4
5 export const schema: { types: SchemaTypeDefinition[] } = {
6   types: [chefs, foods],
7 }
8
```

Create a file named `Food.ts` to import the food data and copy the API Code



```
hack_2 > src > sanity > schemaTypes > TS foods.ts > default > fields
1 import { sources } from "next/dist/compiled/webpack/webpack";
2
3 export default {
4   name: 'Food',
5   type: 'document',
6   title: 'Food',
7   fields: [
8     {
9       name: 'name',
10      type: 'string',
11      title: 'Food Name',
12    },
13
14    {
15      name: 'slug',
16      type: 'slug',
17      title: 'Slug',
18      options: {
19        source: 'name',
20        maxLength: 96
21      }
22    },
23
24    {
25      name: 'category',
26      type: 'string',
27      title: 'Category',
28      description:
29        'Category of the food item (e.g., Burger, Sandwich, Drink, etc.)',
30    },
31
32    {
33      name: 'price',
34      type: 'number',
35      title: 'Current Price',
36    },
37
38    {
39      name: 'originalPrice',
40      type: 'number',
41      title: 'Original Price',
42      description: 'Price before discount (if any)'
43    }
44  ]
45 }
```

```

40     description: 'Price before discount (if any)',
41   },
42   {
43     name: 'tags',
44     type: 'array',
45     title: 'Tags',
46     of: [{ type: 'string' }],
47     options: {
48       layout: 'tags',
49     },
50     description: 'Tags for categorization (e.g., Best Seller, Popular, New)',
51   },
52   {
53     name: 'image',
54     type: 'image',
55     title: 'Food Image',
56     options: {
57       hotspot: true,
58     },
59   },
60   {
61     name: 'description',
62     type: 'text',
63     title: 'Description',
64     description: 'Short description of the food item',
65   },
66   {
67     name: 'available',
68     type: 'boolean',
69     title: 'Available',
70     description: 'Availability status of the food item',
71   },
72 ],
73 };
74

```

Similarly, Create a file named Chefs.ts to import the chefs data and copy the API CODE:

hack\_2 > src > sanity > schemaTypes > TS chefs.ts > default

```
1 export default {
2   name: 'chef',
3   type: 'document',
4   title: 'Chef',
5   fields: [
6     {
7       name: 'name',
8       type: 'string',
9       title: 'Chef Name',
10    },
11    {
12      name: 'position',
13      type: 'string',
14      title: 'Position',
15      description: 'Role or title of the chef (e.g., Head Chef, Sous Chef)',
16    },
17    {
18      name: 'experience',
19      type: 'number',
20      title: 'Years of Experience',
21      description: 'Number of years the chef has worked in the culinary field',
22    },
23    {
24      name: 'specialty',
25      type: 'string',
26      title: 'Specialty',
27      description: 'Specialization of the chef (e.g., Italian Cuisine, Pastry)',
28    },
29    {
30      name: 'image',
31      type: 'image',
32      title: 'Chef Image',
33      options: {
34        hotspot: true,
35      },
36    },
37    {
38      name: 'description',
39      type: 'text',
40      title: 'Description'
```

Ln 1, Col 11 (1 sel)

```
5   fields: [
29     {
30       name: 'image',
31       type: 'image',
32       title: 'Chef Image',
33       options: {
34         hotspot: true,
35       },
36     },
37     {
38       name: 'description',
39       type: 'text',
40       title: 'Description',
41       description: 'Short bio or introduction about the chef',
42     },
43     {
44       name: 'available',
45       type: 'boolean',
46       title: 'Currently Active',
47       description: 'Availability status of the chef',
48     },
49   ],
50 };
```

## 4.5 Making API Calls in React Components

Specifically, you are making a **GET request** to fetch data from a Sanity API using the `client.fetch()` method. This is essentially an API call to fetch food data from your Sanity content platform.

Here's the relevant part of your code where the API call is made:

```
hack_2 > src > app > shop > @ page.tsx > Shop
1  import Link from "next/link";
2  import Image from "next/image";
3  import { client } from "@sanity/lib/client";
4
5
6  // TypeScript type for food items
7  type FoodItem = {
8    name: string;
9    price: number;
10   originalPrice: number;
11   slug: string;
12   image_url: string;
13 };
14
15 // Sanity query to fetch food items
16 const query = `*[_type=='food']{
17   name,
18   price,
19   originalPrice,
20   "slug": slug.current,
21   "image_url": image.asset->url
22 }`;
23
24 // Async server component to fetch food data
25 export default async function Shop() {
26   const food: FoodItem[] = await client.fetch(query);
27
28   return (
29     <div className="w-full overflow-x-hidden">
30       <div className="relative font-[sans-serif] pt-20 before:absolute before:w-full before:h-full before:inset-0 before:bg-black before:opacity-50">
31         </div>
32
33         <div className="mt-10 mb-10 mx-auto flex flex-wrap lg:flex-nowrap justify-center gap-4">
34           <div className="flex flex-col gap-4 w-full lg:w-[70%]">
35             <div className="flex flex-wrap gap-4 mt-20 items-center justify-center">
36
37             </div>
38
39             <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6 mt-8">
40               {food.map((item, index) => (
41                 <Link key={index} href={`/${shop}/${item.slug}`}>
42                   <div className="flex flex-col items-center cursor-pointer">
43                     <Image
44                       src={item.image_url}
45                       alt={item.name}
46                       width={300}
47                       height={300}
48                       className="object-cover rounded-lg"
49                     />
50                     <p className="text-2xl text-[#333333] font-bold">{item.name}</p>
51                     <p className="text-[#FF9F00]">
52                       {item.price.toFixed(2)}{" "}
53                       <del>{item.originalPrice.toFixed(2)}</del>
54                     </p>
55                   </div>
56                 </Link>
57               ))}
58             </div>
59           </div>
60         </div>
61       </div>
62     );
63   }
64 }
65
66
```

## 4.6 Handling API Errors

While fetching content from the API, error handling is important to ensure the website functions smoothly. Using `try-catch` blocks around API calls allows the app to gracefully handle issues like network failures.

```
try {
  food = await client.fetch(query); // Fetching data from Sanity
} catch (error) {
  console.error("Error fetching data:", error);
  errorMessage = "Failed to load food items. Please try again later.";
}
```

## 5. Migration Steps

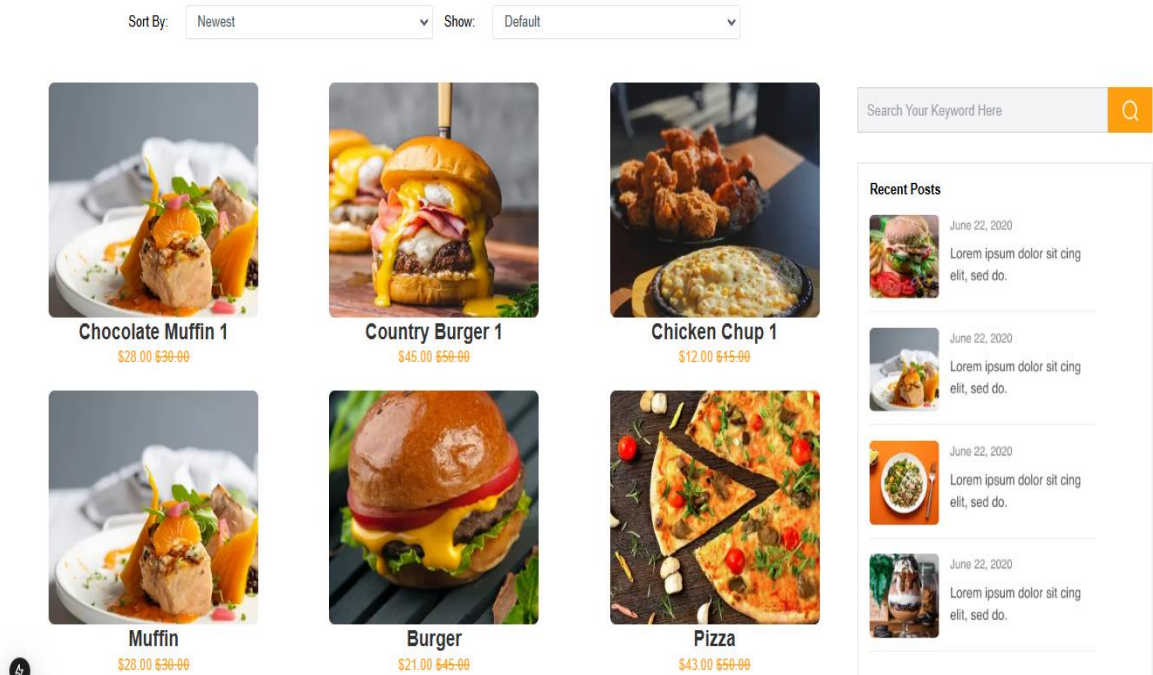
Since the data for **food** and **chef** content is fetched from a provided API, the migration steps focus on integrating this external data into the Sanity CMS rather than manually creating schemas or entering data manually through the Sanity Studio.

### 5.1 Fetching Data from the API

The data for food and chef content is fetched from an external API provided by Sir Mubashir. This eliminates the need for manual entry of data and allows for dynamic and automated data fetching. The data is fetched directly using API calls, and the relevant content is integrated into the system.

- **Food Data:** The content for food items (name, price, description, image, etc.) is fetched by querying the API and then used within the application.
- **Chef Data:** Similarly, information about chefs (name, bio, photo, etc.) is fetched from the API and used in the application.

## • DATA SUCCESSFULLY DISPLAYED ON FRONT END:



## 5.2 Data Integration into Sanity

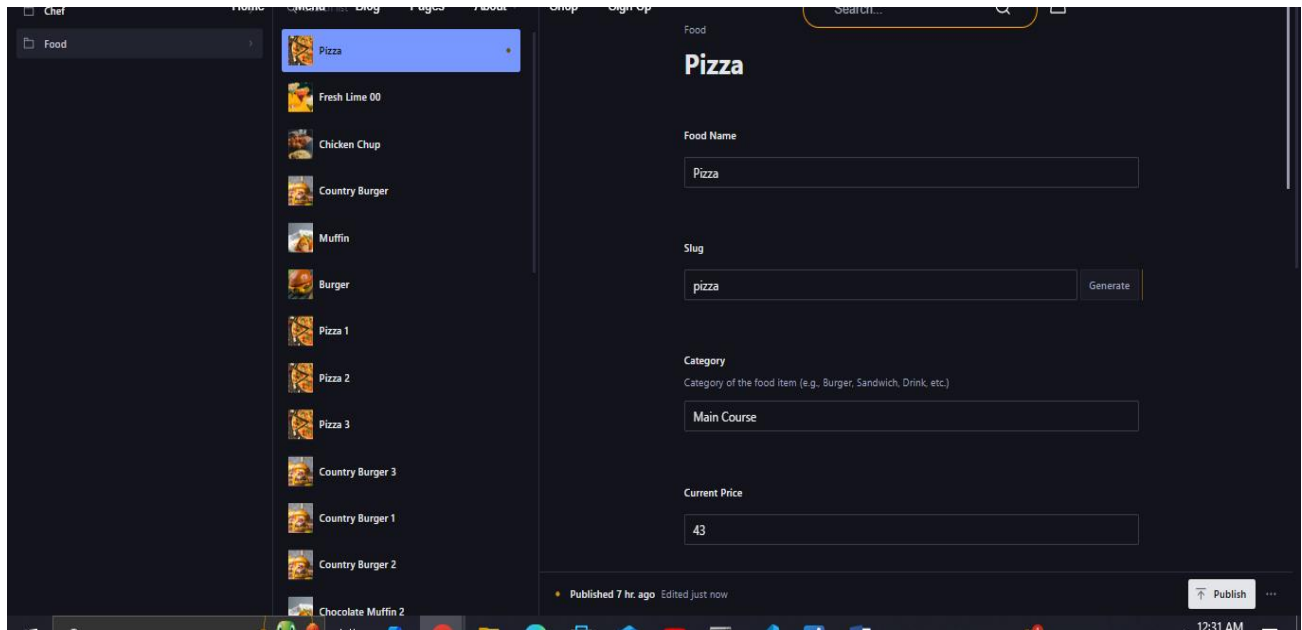
Once the data is fetched from the provided API, it is parsed and inserted into the **Sanity CMS** through the use of Sanity's client SDK.

- **Sanity Client:** The Sanity client (initialized in the project) is used to manage the connection with the Sanity CMS and to ensure that the fetched content is integrated correctly.
- **Sanity Queries:** A custom query is used to map the data from the API into the structure expected by the Sanity schemas. The fetched data is then stored in Sanity to ensure it can be dynamically rendered by the front-end of the website.

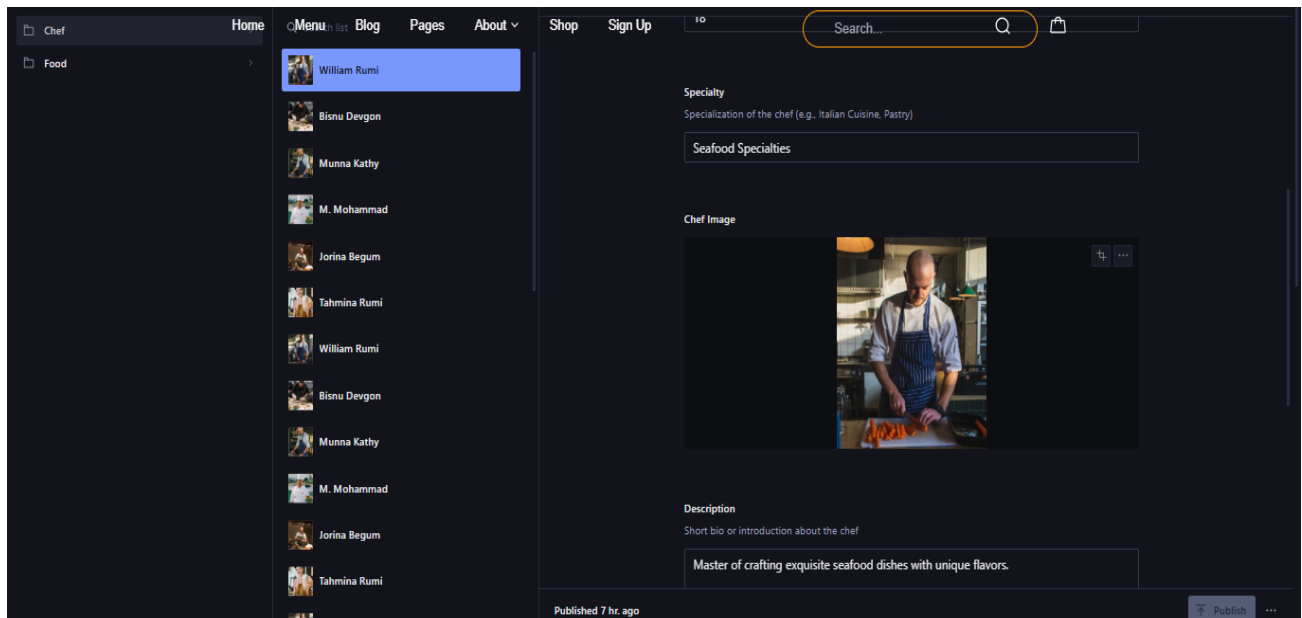
## \* POPULATED SANITY CMS FIELDS:

FOOD FIELDS:





## CHEF FIELDS:



## 5.3 Handling API Integration

Instead of manually entering the data into the Sanity Studio, the application is set up to **automatically fetch** and render food and chef data by making API calls to the provided endpoint.

- The Sanity schemas are designed to align with the structure of the fetched data (for example, the **food** and **chef** types in the Sanity schema).
- The **Sanity client** fetches the data from the API and populates the CMS dynamically.

## 6. Tools Used

- **Sanity.io**: Used as the headless CMS to store food and chef content.
- **Next.js**: Used as the front-end framework for building the React application.
- **React**: Used to build the components that display content on the frontend.
- **GROQ**: Sanity's query language used to fetch data from the CMS.
- **Axios or Fetch**: For making API calls to retrieve data from Sanity.

## 7. Conclusion

The integration of food and chef schemas into the website was successfully carried out using Sanity.io as a headless CMS. This allowed dynamic content management, making it easier to update and manage food items and chef details on the website. API calls were structured using GROQ queries, and the data was integrated into React components to dynamically render content on the frontend. The process involved using defined schemas, creating API calls, and handling potential errors in the system to ensure a smooth user experience.