Day 3 - API Integration Report - [FOODTUCK]

Explanation of What I Have Done in Day 3 - Hackathon

1. Cloning the Repository

- I started by cloning the repository containing the Sanity project and import scripts.
- I used the following command to clone the repository:

```
git clone https://github.com/mubashirimtiaz/sanity-nextjs.git
cd sanity-nextjs
```

- This gave me access to the project files, including the Sanity schema and the import script.

---

2. Installing Dependencies

- After cloning the repository, I installed all the required packages by running:

```
npm install
```

- This installed all the dependencies listed in the package.json file, including Sanity client, Next.js, and other necessary libraries.

---

3. Configuring Environment Variables

- I created a .env.local file in the root of the project directory to store sensitive environment variables.
- I added the following variables to the .env.local file:

```
NEXT_PUBLIC_SANITY_PROJECT_ID="{your-sanity-project-id}"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="{your-sanity-api-token}"
```

- **Explanation of Variables**:
  - NEXT_PUBLIC_SANITY_PROJECT_ID: Found in my Sanity project settings.
  - SANITY_API_TOKEN: Generated by navigating to **Settings > API > Add API Token** in the Sanity dashboard. I gave the token **read/write permissions**.
  - NEXT_PUBLIC_SANITY_DATASET: Set to production as the dataset name.

---

4. Importing Data

- I ran the following command to import sample data for the **Food** and **Chef** models:

```
npm run import-data
```

- This executed the import-data.mjs script, which:
  1. Fetched food and chef data from the external API (https://sanity-nextjs-rouge.vercel.app/api/foods and https://sanity-nextjs-rouge.vercel.app/api/chefs).

2. Uploaded images to Sanity and created references.
3. Created documents in Sanity for each food and chef item.

---

5. Verifying the Data in Sanity Studio

- After running the import script, I opened my **Sanity Studio** project.
- I verified that two models were created:
    1. **Food**: Contains fields like name, category, price, originalPrice, tags, image, description, and available.
    2. **Chef**: Contains fields like name, position, experience, specialty, image, description, and available.
- I confirmed that the sample data was successfully populated in both models.

What I Have Done After This

After completing the data import, I worked on the following:

---

1. Updated Schemas

- I updated the **Food** and **Chef** schemas in Sanity to match the data structure from the API.
- For example, I added fields like originalPrice and tags to the food schema and experience and specialty to the chef schema.

---

2. Built the Next.js Frontend

- I created two components in my Next.js application:
    1. **Food Component**:
        - Fetches food data from Sanity using a GROQ query.
        - Displays food items in a grid layout with sorting, filtering, and pagination features.
    2. **Chef Component**:
        - Fetches chef data from Sanity using a GROQ query.
        - Displays chefs in a grid layout with hover effects and availability status.

---

1. API Integration Process

1. Understanding the API

- The external API provided two endpoints:
    1. **Food Data**:
        - Endpoint: https://sanity-nextjs-rouge.vercel.app/api/foods
        - Returns a list of food items with fields like name, category, price, originalPrice, tags, image, description, and available.
    2. **Chef Data**:
        - Endpoint: https://sanity-nextjs-rouge.vercel.app/api/chefs
        - Returns a list of chefs with fields like name, position, experience, specialty, image, description, and available.

---

2. Fetching Data from the API

- I used **Axios** to fetch data from the API endpoints.
- Example for fetching **Food Data**:

```
const foodsResponse = await axios.get('https://sanity-nextjs-rouge.vercel.app/api/foods');
const foods = foodsResponse.data;
```

- Example for fetching **Chef Data**:

```
const chefsResponse = await axios.get('https://sanity-nextjs-rouge.vercel.app/api/chefs');
const chefs = chefsResponse.data;
```

---

3. Uploading Images to Sanity

- For each food and chef item, I checked if an image URL was provided.

```
async function uploadImageToSanity(imageUrl) {
  try {
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(),
    });
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}
```

---

4. Creating Documents in Sanity

- After uploading images, I created documents in Sanity for each food and chef item.
- Example for **Food**:

```
for (const food of foods) {
 const sanityFood = {
  _type: 'food',
  name: food.name,
  category: food.category || null,
  price: food.price,
  originalPrice: food.originalPrice || null,
  tags: food.tags || [],
  description: food.description || '',
  available: food.available !== undefined ? food.available : true,
  image: imageRef
   ? {
     _type: 'image',
     asset: {
      _type: 'reference',
      _ref: imageRef,
     },
    }
   : undefined,
 };
 await client.create(sanityFood);
}
```

- Example for **Chef**:

```
for (const chef of chefs) {
 const sanityChef = {
  _type: 'chef',
  name: chef.name,
  position: chef.position || null,
  experience: chef.experience || 0,
  specialty: chef.specialty || '',
  description: chef.description || '',
  available: chef.available !== undefined ? chef.available : true,
  image: imageRef
   ? {
     _type: 'image',
     asset: {
      _type: 'reference',
      _ref: imageRef,
     },
    }
   : undefined,
 };
 await client.create(sanityChef);
}
```

---

5. Running the Migration Script

- I ran the migration script using the following command:

```
npm run import-data
```

- This executed the import-data.mjs script, which:
    1. Fetched data from the API.

2. Uploaded images to Sanity.
3. Created documents in Sanity for each food and chef item.

---

6. Verifying the Data in Sanity Studio

- After running the script, I opened **Sanity Studio** to verify that the data was successfully imported.
- I checked the **Food** and **Chef** models to ensure that all fields were populated correctly.

Tools Used

1. Sanity Client

- The **Sanity client** (@sanity/client) was used to interact with the Sanity CMS.
- It allowed me to:
    - Upload images using client.assets.upload.
    - Create documents using client.create.

2. Axios

- **Axios** was used to fetch data from the external API.
- It handled HTTP requests and responses efficiently.

3. Node.js

- The migration script was executed using **Node.js**.
- Node.js provided the runtime environment for running the script.

4. Environment Variables

- I used a .env.local file to store sensitive environment variables like:
    - NEXT_PUBLIC_SANITY_PROJECT_ID
    - SANITY_API_TOKEN
    - NEXT_PUBLIC_SANITY_DATASET

5. Sanity Studio

- **Sanity Studio** was used to:
    - Define schemas for the **Food** and **Chef** models.
    - Verify that the imported data was correctly populated.

Summary of API Integration Process

1. **Fetched Data from the API**:
   - Used Axios to fetch food and chef data from the external API.
2. **Uploaded Images to Sanity**:
   - Used the Sanity client to upload images and create references.
3. **Created Documents in Sanity**:
   - Used the Sanity client to create documents for each food and chef item.
4. **Ran the Migration Script**:
   - Executed the script using `npm run import-data`.
5. **Verified the Data**:
   - Checked Sanity Studio to ensure the data was imported correctly.
6. **Tools Used**:
   - Sanity Client, Axios, Node.js, Environment Variables, and Sanity Studio.

What Worked Well

- The migration script successfully fetched data from the API and uploaded it to Sanity CMS.
- The use of environment variables ensured that sensitive information was securely stored.
- Sanity Studio made it easy to verify that the data was correctly imported.

2. Adjustments Made to Schemas:

1. Food Schema

- I updated the **Food schema** in Sanity CMS to match the data structure from the external API.
- The updated schema includes the following fields:

```
const food = defineType({
 name: 'food',
 title: 'Food',
 type: 'document',
 fields: [
  defineField({
   name: 'name',
   title: 'Food Name',
   type: 'string',
  }),
  defineField({
   name: 'category',
   title: 'Category',
   type: 'string',
   description: 'Category of the food item (e.g., Burger, Sandwich, Drink, etc.)',
  }),
  defineField({
   name: 'price',
   title: 'Current Price',
   type: 'number',
  }),
  defineField({
   name: 'originalPrice',
   title: 'Original Price',
   type: 'number',
   description: 'Price before discount (if any)',
  }),
  defineField({
   name: 'tags',
   title: 'Tags',
   type: 'array',
   of: [{ type: 'string' }],
   options: {
     layout: 'tags',
   },
   description: 'Tags for categorization (e.g., Best Seller, Popular, New)',
  }),
  defineField({
   name: 'image',
   title: 'Food Image',
   type: 'image',
   options: {
     hotspot: true,
   },
  }),
  defineField({
   name: 'description',
   title: 'Description',
   type: 'text',
   description: 'Short description of the food item',
  }),
  defineField({
   name: 'available',
   title: 'Available',
   type: 'boolean',
   description: 'Availability status of the food item',
  }),
 ],
});
```

Key Adjustments:

- Added originalPrice to store the original price of the food item (before discounts).
- Added tags as an array of strings to categorize food items (e.g., Best Seller, Popular, New).
- Added description to provide a short description of the food item.
- Added available as a boolean field to indicate the availability status of the food item.

---

2. Chef Schema

- I updated the **Chef schema** in Sanity CMS to match the data structure from the external API.
- The updated schema includes the following fields:

```
const chef = defineType({
  name: 'chef',
  title: 'Chef',
  type: 'document',
  fields: [
    defineField({
      name: 'name',
      title: 'Chef Name',
      type: 'string',
    }),
    defineField({
      name: 'position',
      title: 'Position',
      type: 'string',
      description: 'Role or title of the chef (e.g., Head Chef, Sous Chef)',
    }),
    defineField({
      name: 'experience',
      title: 'Years of Experience',
      type: 'number',
      description: 'Number of years the chef has worked in the culinary field',
    }),
    defineField({
      name: 'specialty',
      title: 'Specialty',
      type: 'string',
      description: 'Specialization of the chef (e.g., Italian Cuisine, Pastry)',
    }),
    defineField({
      name: 'image',
      title: 'Chef Image',
      type: 'image',
      options: {
        hotspot: true,
      },
    }),
    defineField({
      name: 'description',
      title: 'Description',
      type: 'text',
      description: 'Short bio or introduction about the chef',
    }),
    defineField({
      name: 'available',
      title: 'Currently Active',
      type: 'boolean',
      description: 'Availability status of the chef',
    }),
  ],
});
```

Key Adjustments:

- Added experience as a number field to store the number of years the chef has worked.
- Added specialty as a string field to store the chef's specialization (e.g., Italian Cuisine, Pastry).
- Added description to provide a short bio or introduction about the chef.
- Added available as a boolean field to indicate the availability status of the chef.

Why These Adjustments Were Made

- **Original Price**: To display discounted prices and show the original price for comparison.
- **Tags**: To categorize food items and improve searchability.
- **Experience and Specialty**: To provide more details about the chefs and their expertise.
- **Description**: To give users more information about food items and chefs.
- **Availability**: To indicate whether a food item or chef is currently available.

How the Adjustments Were Implemented

1. **Updated Schemas in Sanity Studio**:
   - I modified the food and chef schemas in Sanity Studio to include the new fields.
2. **Updated the Migration Script**:
   - I ensured that the migration script (import-data.mjs) mapped the API data to the updated schema fields.
   - Example for **Food**:

```
const sanityFood = {
  _type: 'food',
  name: food.name,
  category: food.category || null,
  price: food.price,
  originalPrice: food.originalPrice || null,
  tags: food.tags || [],
  description: food.description || '',
  available: food.available !== undefined ? food.available : true,
  image: imageRef
    ? {
        _type: 'image',
        asset: {
          _type: 'reference',
          _ref: imageRef,
        },
      }
    : undefined,
};
```

   - Example for **Chef**:

```
const sanityChef = {
  _type: 'chef',
  name: chef.name,
  position: chef.position || null,
  experience: chef.experience || 0,
  specialty: chef.specialty || '',
  description: chef.description || '',
  available: chef.available !== undefined ? chef.available : true,
  image: imageRef
    ? {
        _type: 'image',
        asset: {
          _type: 'reference',
          _ref: imageRef,
        },
      }
    : undefined,
};
```

3. **Tested the Schemas**:
   - I ran the migration script and verified that the data was correctly populated in Sanity Studio.
   - I checked that all fields were mapped correctly and displayed as expected.

Summary of Schema Adjustments

1. **Food Schema**:
   - Added originalPrice, tags, description, and available fields.
   - Updated the schema to match the API data structure.
2. **Chef Schema**:
   - Added experience, specialty, description, and available fields.
   - Updated the schema to match the API data structure.
3. **Migration Script**:
   - Updated the script to map API data to the new schema fields.
4. **Testing**:
   - Verified that the data was correctly populated in Sanity Studio.

What Worked Well

- The schema adjustments ensured that the data from the API was correctly mapped and stored in Sanity CMS.
- The new fields provided more detailed information about food items and chefs, improving the user experience.

3. Migration Steps:

1. Setting Up the Migration Script

- I used the import-data.mjs script provided in the repository to migrate data from the external API to Sanity CMS.

- The script is written in JavaScript and uses the **Sanity client** and **Axios** libraries to fetch and upload data.

---

2. Fetching Data from the API

- The script fetches data from two API endpoints:
    1. **Food Data**:

```
const foodsResponse = await axios.get('https://sanity-nextjs-rouge.vercel.app/api/foods');
const foods = foodsResponse.data;
```

    2. **Chef Data**:

```
const chefsResponse = await axios.get('https://sanity-nextjs-rouge.vercel.app/api/chefs');
const chefs = chefsResponse.data;
```

---

3. Uploading Images to Sanity

- For each food and chef item, the script checks if an image URL is provided.
- If an image exists, it uploads the image to Sanity using the `client.assets.upload` method:

```
async function uploadImageToSanity(imageUrl) {
  try {
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(),
    });
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}
```

---

4. Creating Documents in Sanity

- After uploading images, the script creates documents in Sanity for each food and chef item.
- Example for **Food**:

```
for (const food of foods) {
  const sanityFood = {
    _type: 'food',
    name: food.name,
    category: food.category || null,
    price: food.price,
    originalPrice: food.originalPrice || null,
    tags: food.tags || [],
    description: food.description || '',
    available: food.available !== undefined ? food.available : true,
    image: imageRef
      ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        }
      : undefined,
  };
  await client.create(sanityFood);
}
```

- Example for **Chef**:

```
for (const chef of chefs) {
  const sanityChef = {
    _type: 'chef',
    name: chef.name,
    position: chef.position || null,
    experience: chef.experience || 0,
    specialty: chef.specialty || '',
    description: chef.description || '',
    available: chef.available !== undefined ? chef.available : true,
    image: imageRef
      ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        }
      : undefined,
  };
  await client.create(sanityChef);
}
```

---

5. Running the Migration Script

- I ran the migration script using the following command:

```
npm run import-data
```

- This executed the `import-data.mjs` script, which:
  1. Fetched data from the API.
  2. Uploaded images to Sanity.
  3. Created documents in Sanity for each food and chef item.

---

6. Verifying the Data in Sanity Studio

- After running the script, I opened **Sanity Studio** to verify that the data was successfully imported.

- I checked the **Food** and **Chef** models to ensure that all fields were populated correctly.

Tools Used

1. Sanity Client

- The **Sanity client** (@sanity/client) was used to interact with the Sanity CMS.
- It allowed me to:
  - Upload images using client.assets.upload.
  - Create documents using client.create.

2. Axios

- **Axios** was used to fetch data from the external API.
- It handled HTTP requests and responses efficiently.

3. Node.js

- The migration script was executed using **Node.js**.
- Node.js provided the runtime environment for running the script.

4. Environment Variables

- I used a .env.local file to store sensitive environment variables like:
  - NEXT_PUBLIC_SANITY_PROJECT_ID
  - SANITY_API_TOKEN
  - NEXT_PUBLIC_SANITY_DATASET

5. Sanity Studio

- **Sanity Studio** was used to:
  - Define schemas for the **Food** and **Chef** models.
  - Verify that the imported data was correctly populated.

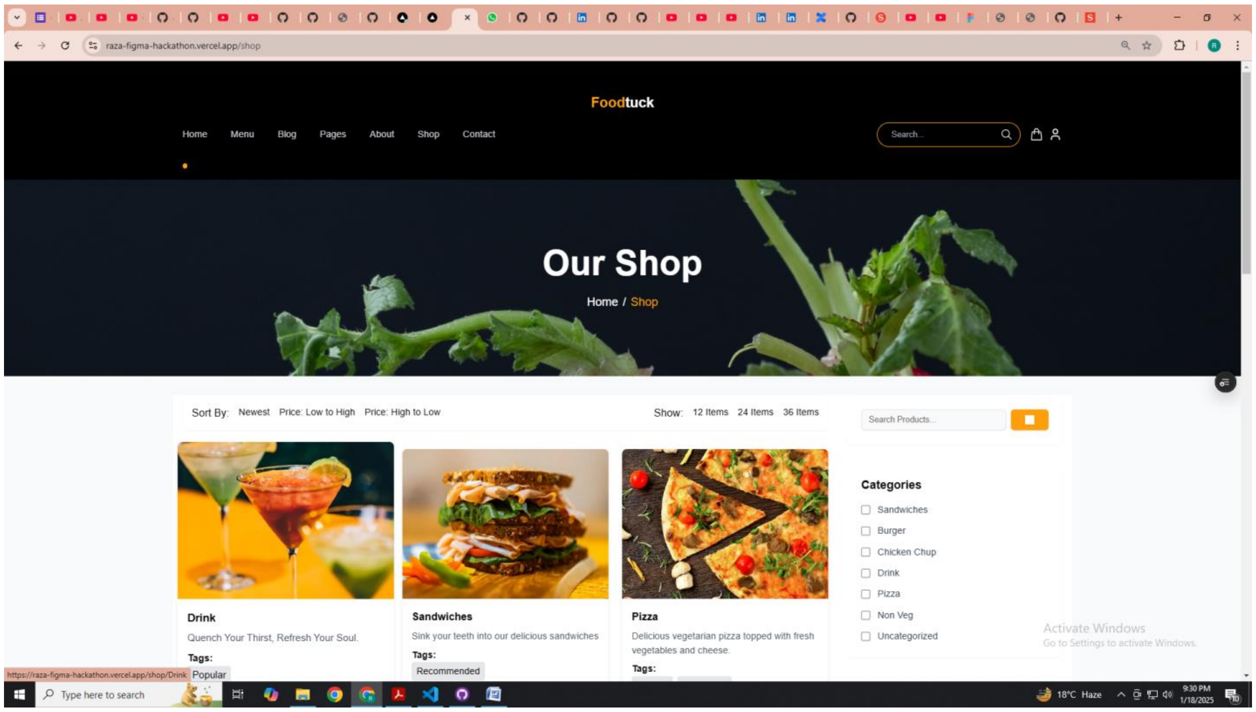
Summary of Migration Steps and Tools

1. **Set Up the Migration Script**:
   - Used import-data.mjs to fetch and upload data.
2. **Fetched Data from the API**:
   - Used Axios to fetch food and chef data from the external API.
3. **Uploaded Images to Sanity**:

   - Used the Sanity client to upload images and create references.
4. **Created Documents in Sanity**:
   - Used the Sanity client to create documents for each food and chef item.
5. **Ran the Migration Script**:
   - Executed the script using `npm run import-data`.
6. **Verified the Data**:
   - Checked Sanity Studio to ensure the data was imported correctly.
7. **Tools Used**:
   - Sanity Client, Axios, Node.js, Environment Variables, and Sanity Studio.

What Worked Well

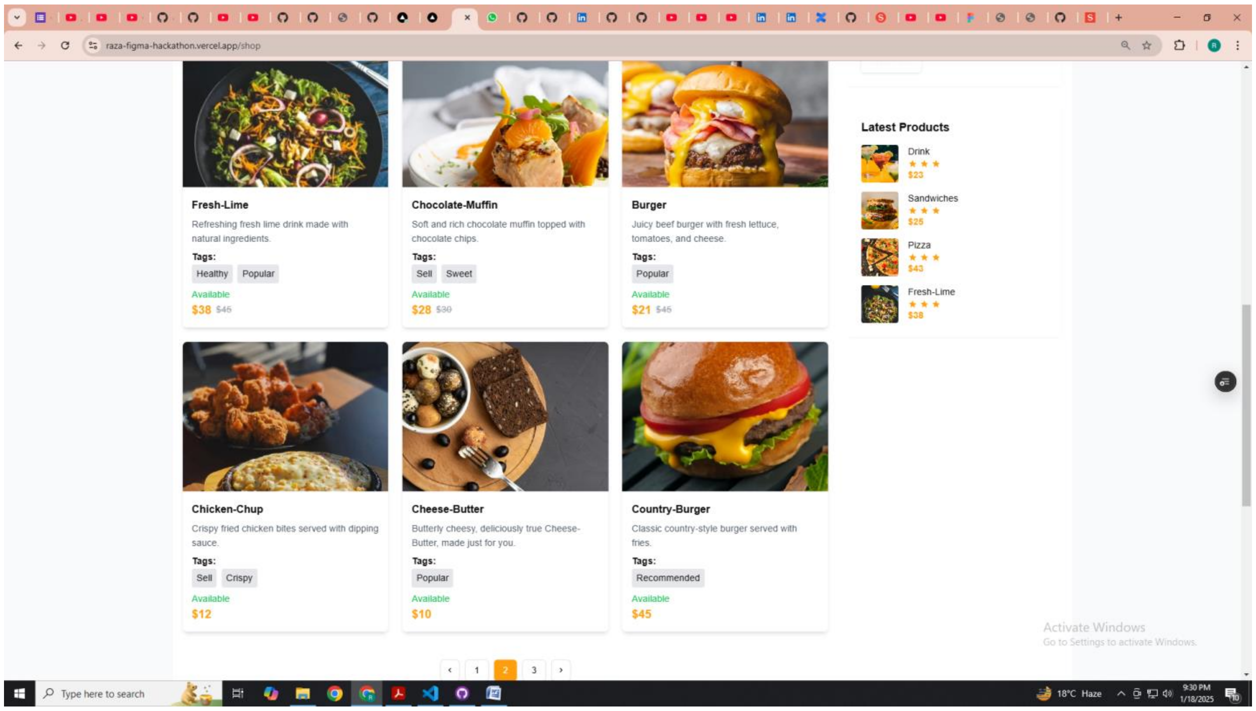
- The migration script successfully fetched data from the API and uploaded it to Sanity CMS.
- The use of environment variables ensured that sensitive information was securely stored.
- Sanity Studio made it easy to verify that the data was correctly imported.



Quench Your Thirst, Refresh Your Soul.

Sink your teeth into our delicious sandwiches

Delicious vegetarian pizza topped with fresh vegetables and cheese.



raza-figma-hackathon.vercel.app/shop

Fresh-Lime
Refreshing fresh lime drink made with natural ingredients.
Tags: Healthy Popular
Available
$38 ~~$45~~

Chocolate-Muffin
Soft and rich chocolate muffin topped with chocolate chips.
Tags: Sell Sweet
Available
$28 ~~$30~~

Burger
Juicy beef burger with fresh lettuce, tomatoes, and cheese.
Tags: Popular
Available
$21 ~~$45~~

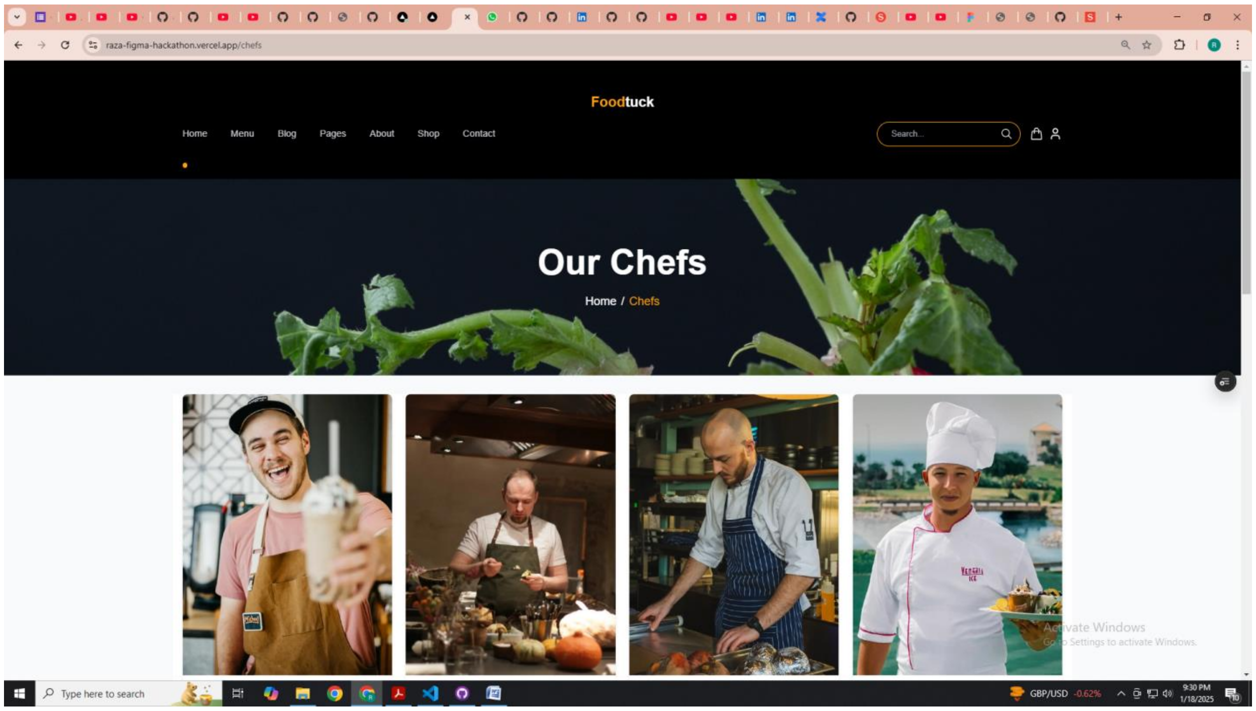
Latest Products
Drink ★★★ $23
Sandwiches ★★★ $25
Pizza ★★★ $43
Fresh-Lime ★★★ $38

Chicken-Chup
Crispy fried chicken bites served with dipping sauce.
Tags: Sell Crispy
Available
$12

Cheese-Butter
Butterly cheesy, deliciously true Cheese-Butter, made just for you.
Tags: Popular
Available
$10

Country-Burger
Classic country-style burger served with fries.
Tags: Recommended
Available
$45

‹ 1 2 3 ›



raza-figma-hackathon.vercel.app/chefs

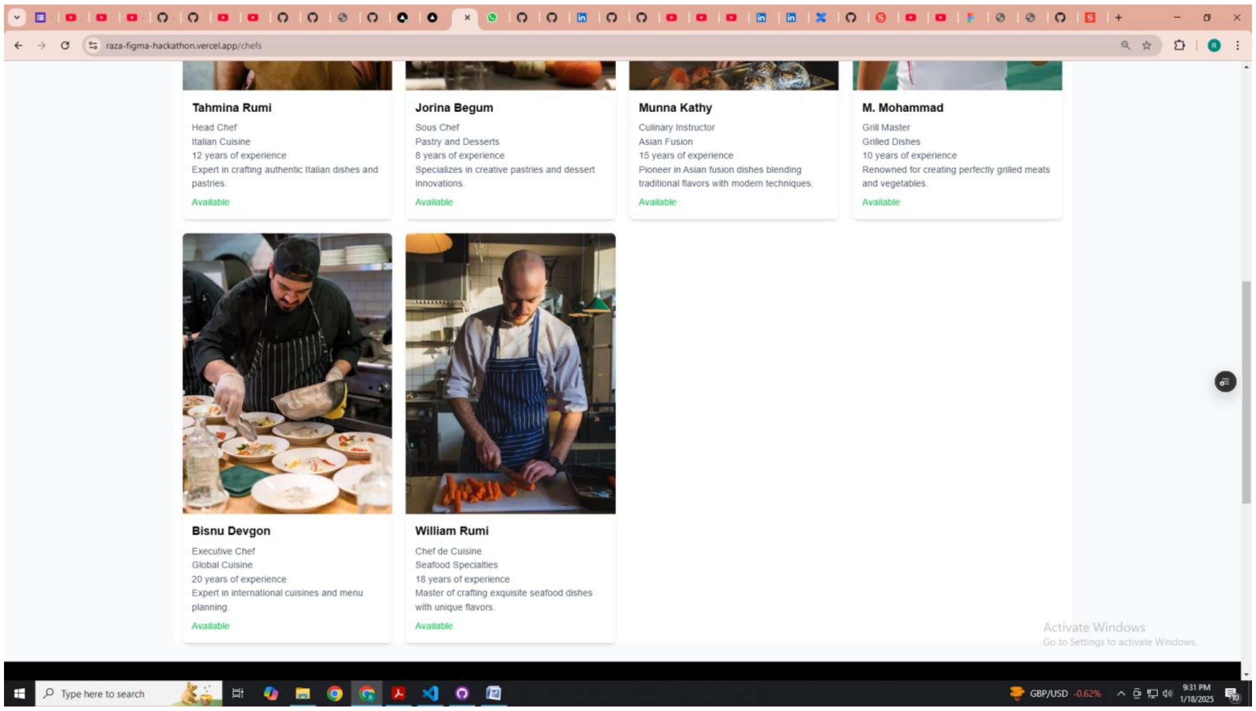
Foodtuck

Home Menu Blog Pages About Shop Contact

Search...

Our Chefs

Home / Chefs


raza-figma-hackathon.vercel.app/chefs

Tahmina Rumi
Head Chef
Italian Cuisine
12 years of experience
Expert in crafting authentic Italian dishes and pastries.
Available

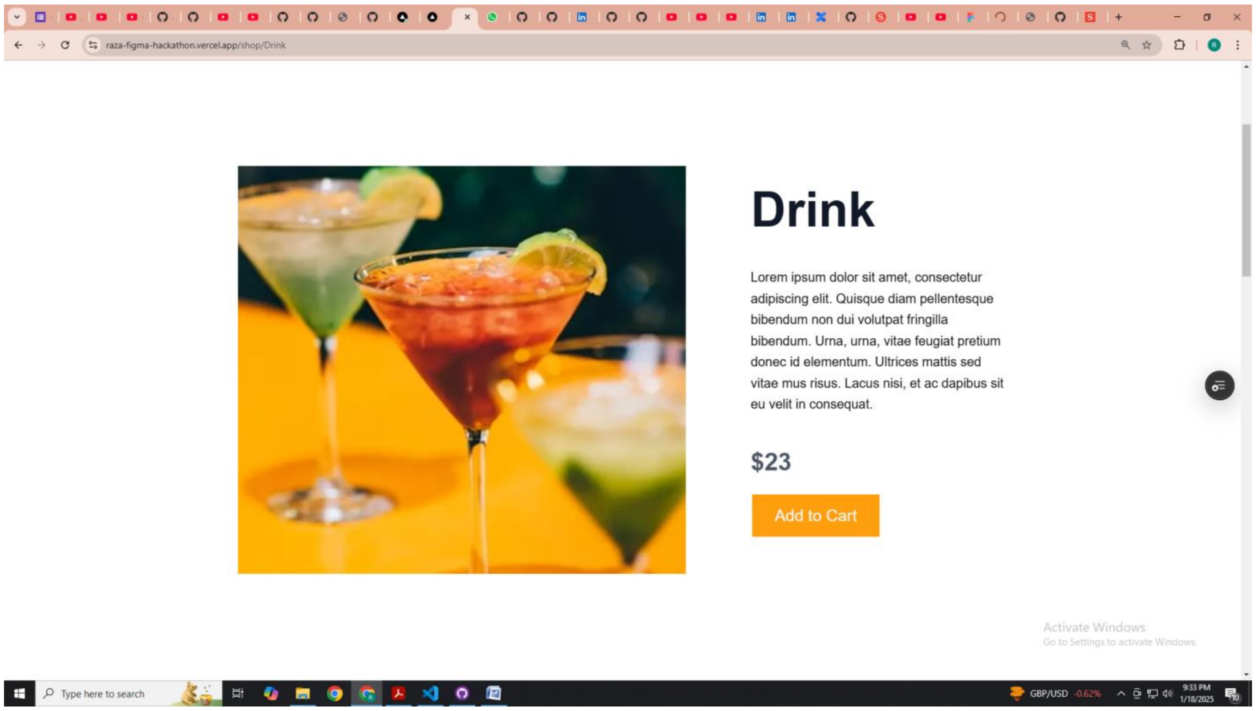
Jorina Begum
Sous Chef
Pastry and Desserts
8 years of experience
Specializes in creative pastries and dessert innovations.
Available

Munna Kathy
Culinary Instructor
Asian Fusion
15 years of experience
Pioneer in Asian fusion dishes blending traditional flavors with modern techniques.
Available

M. Mohammad
Grill Master
Grilled Dishes
10 years of experience
Renowned for creating perfectly grilled meats and vegetables.
Available

Bisnu Devgon
Executive Chef
Global Cuisine
20 years of experience
Expert in international cuisines and menu planning.
Available

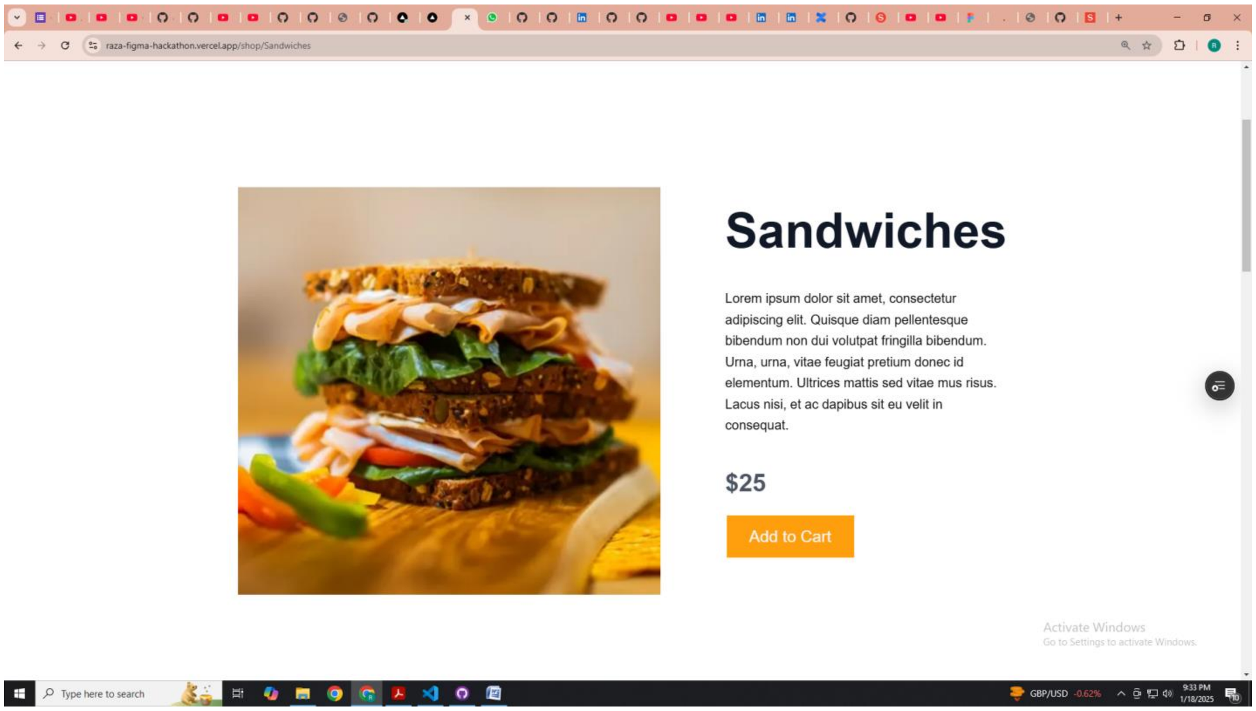
William Rumi
Chef de Cuisine
Seafood Specialties
18 years of experience
Master of crafting exquisite seafood dishes with unique flavors.
Available


raza-figma-hackathon.vercel.app/shop/Drink

Drink

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque diam pellentesque bibendum non dui volutpat fringilla bibendum. Urna, urna, vitae feugiat pretium donec id elementum. Ultrices mattis sed vitae mus risus. Lacus nisi, et ac dapibus sit eu velit in consequat.

$23

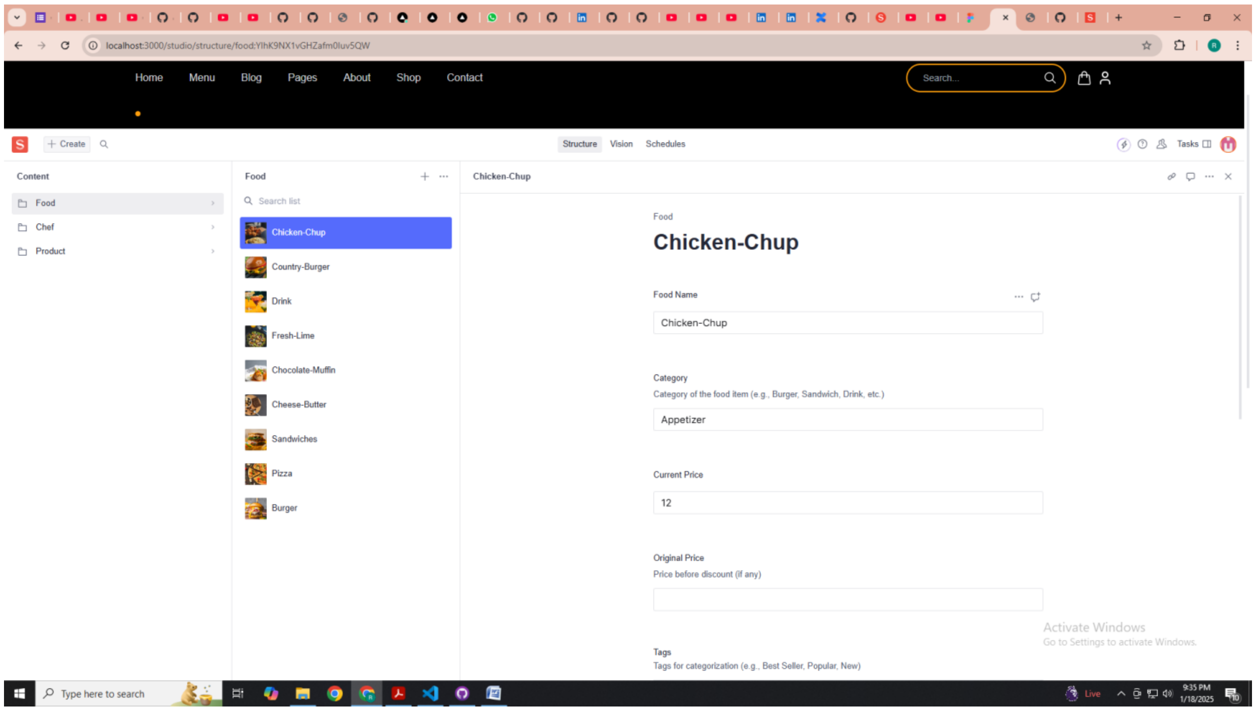


raza-figma-hackathon.vercel.app/shop/Sandwiches

Sandwiches

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque diam pellentesque bibendum non dui volutpat fringilla bibendum. Urna, urna, vitae feugiat pretium donec id elementum. Ultrices mattis sed vitae mus risus. Lacus nisi, et ac dapibus sit eu velit in consequat.

$25



localhost:3000/studio/structure/food;YlhK9NX1vGHZafm0luv5QW

Home Menu Blog Pages About Shop Contact

Content: Food, Chef, Product

Food: Chicken-Chup, Country-Burger, Drink, Fresh-Lime, Chocolate-Muffin, Cheese-Butter, Sandwiches, Pizza, Burger

Food

Chicken-Chup

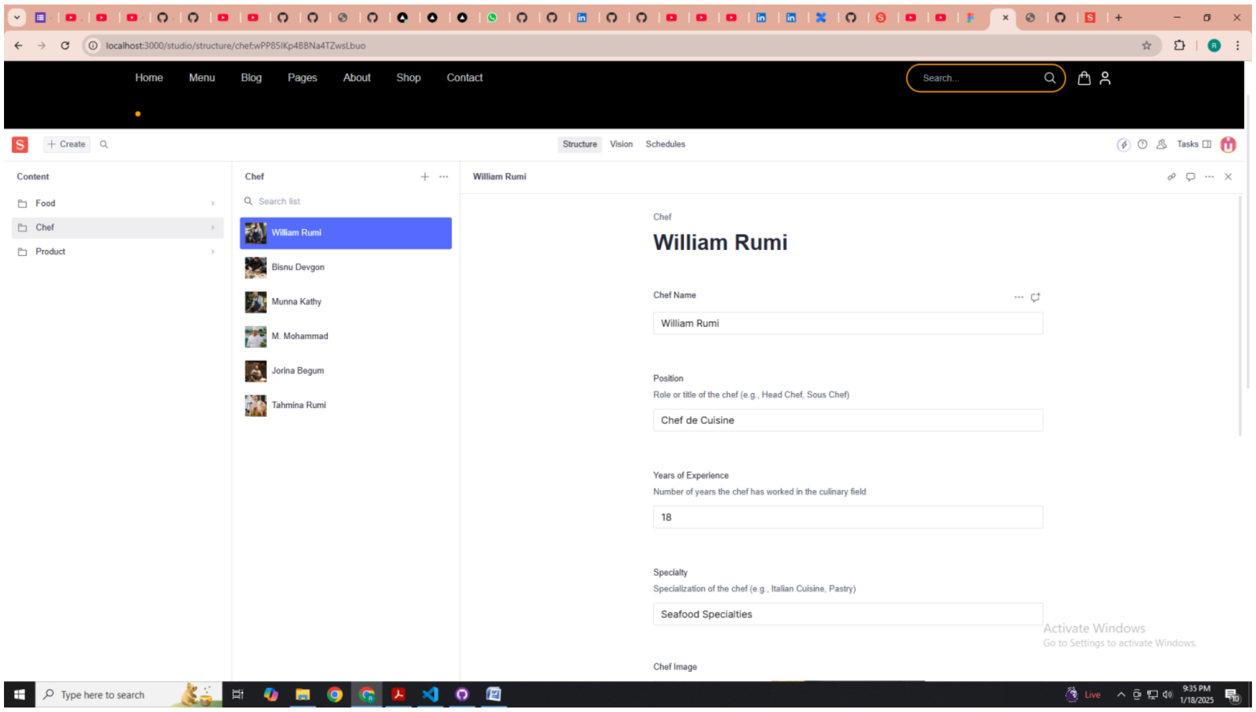
Food Name
Chicken-Chup

Category
Category of the food item (e.g., Burger, Sandwich, Drink, etc.)
Appetizer

Current Price
12

Original Price
Price before discount (if any)

Tags
Tags for categorization (e.g., Best Seller, Popular, New)



localhost:3000/studio/structure/chef;wPP85IKp488Na4TZwsLbuo

Home Menu Blog Pages About Shop Contact

Content: Food, Chef, Product

Chef: William Rumi, Bisnu Devgon, Munna Kathy, M. Mohammad, Jorina Begum, Tahmina Rumi

Chef

William Rumi

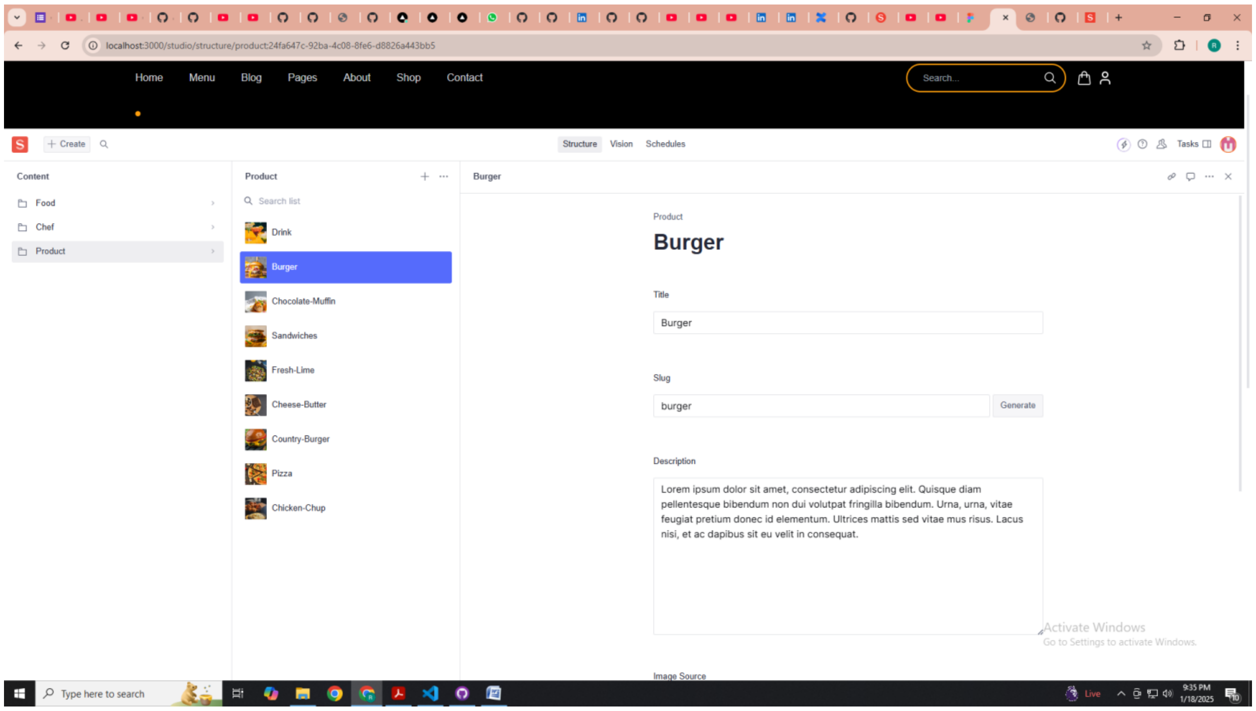
Chef Name
William Rumi

Position
Role or title of the chef (e.g., Head Chef, Sous Chef)
Chef de Cuisine

Years of Experience
Number of years the chef has worked in the culinary field
18

Specialty
Specialization of the chef (e.g., Italian Cuisine, Pastry)
Seafood Specialties

Chef Image



Product

Burger

Title

Burger

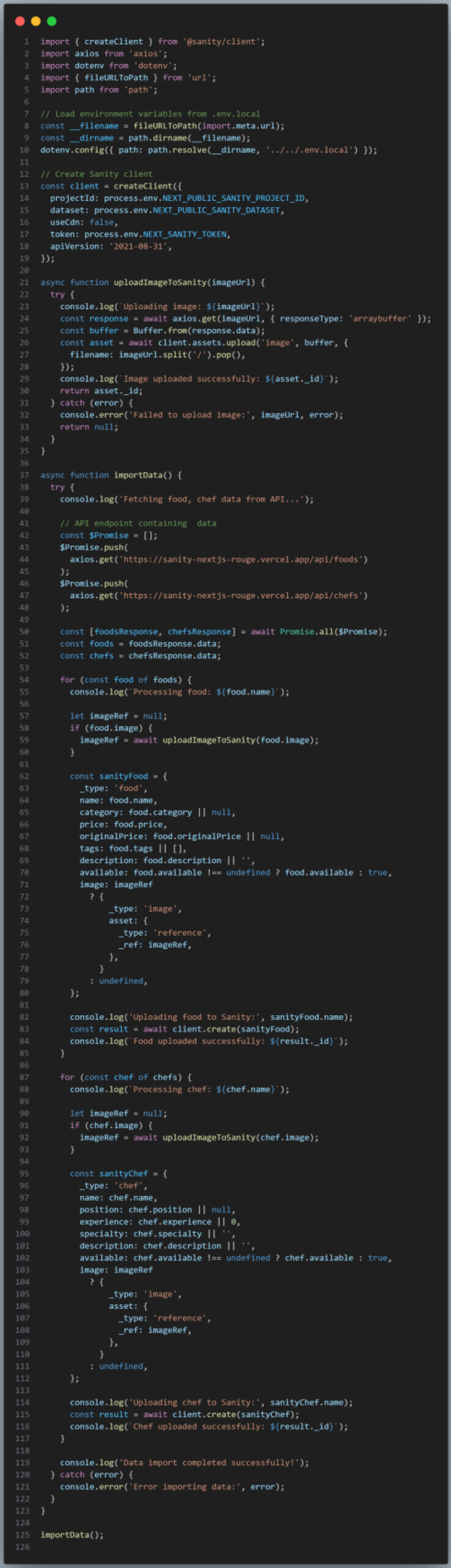
Slug

burger

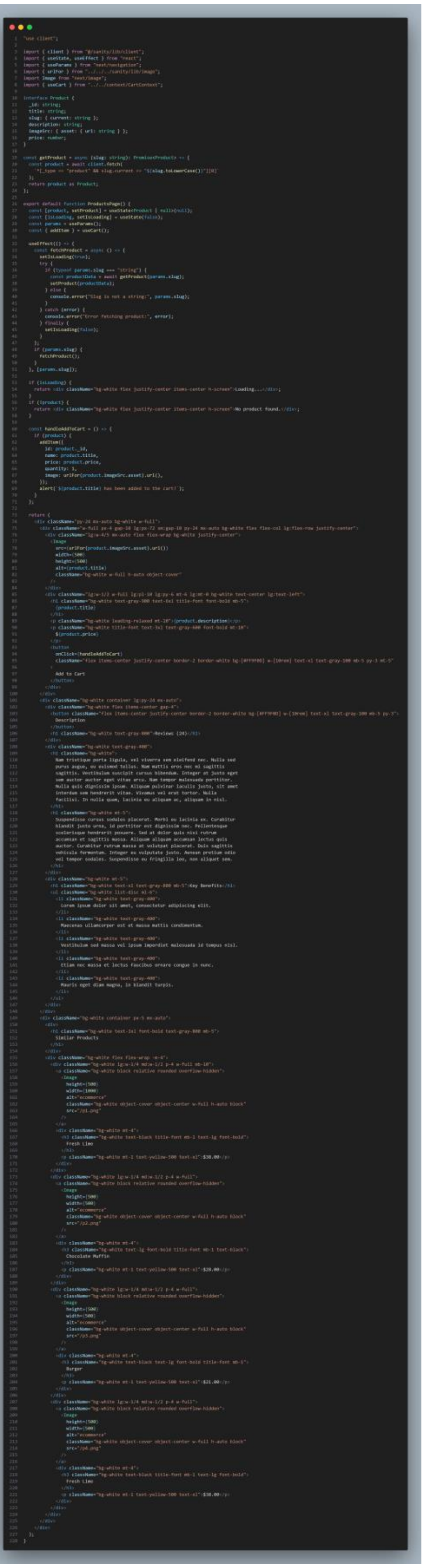
Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque diam pellentesque bibendum non dui volutpat fringilla bibendum. Urna, urna, vitae feugiat pretium donec id elementum. Ultrices mattis sed vitae mus risus. Lacus nisi, et ac dapibus sit eu velit in consequat.

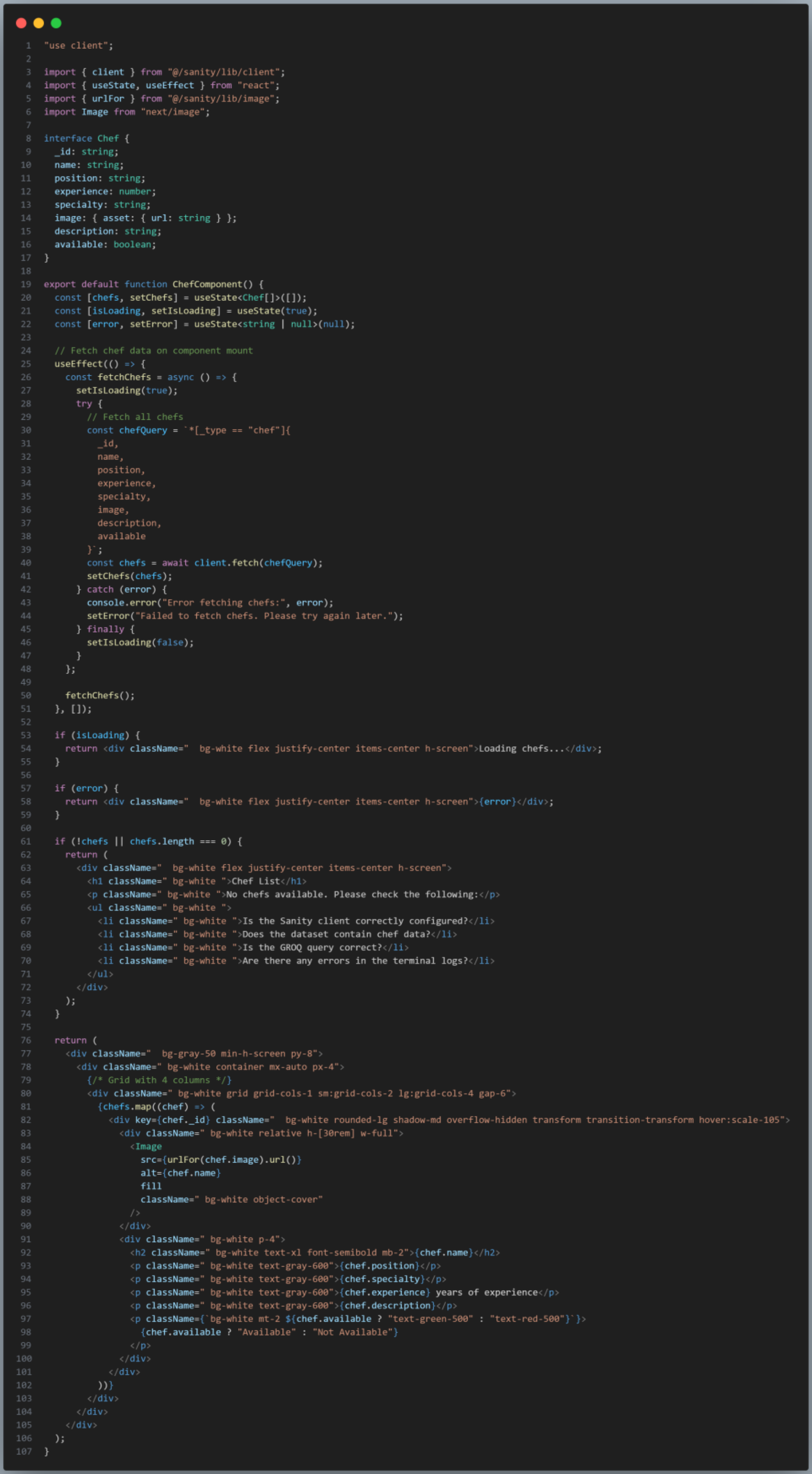
Image Source



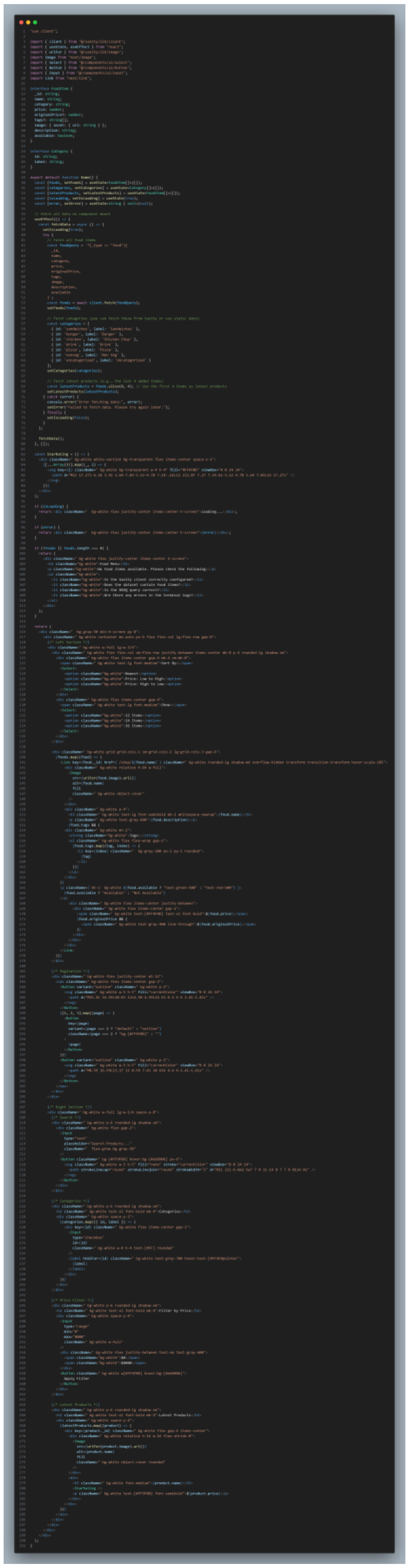
```javascript
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

// Load environment variables from .env.local
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: path.resolve(__dirname, '../../.env.local') });

// Create Sanity client
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: false,
  token: process.env.NEXT_PUBLIC_SANITY_TOKEN,
  apiVersion: '2021-08-31',
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(),
    });
    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}

async function importData() {
  try {
    console.log('Fetching food, chef data from API...');

    // API endpoint containing  data
    const $Promise = [];
    $Promise.push(
      axios.get('https://sanity-nextjs-rouge.vercel.app/api/foods')
    );
    $Promise.push(
      axios.get('https://sanity-nextjs-rouge.vercel.app/api/chefs')
    );

    const [foodsResponse, chefsResponse] = await Promise.all($Promise);
    const foods = foodsResponse.data;
    const chefs = chefsResponse.data;

    for (const food of foods) {
      console.log(`Processing food: ${food.name}`);

      let imageRef = null;
      if (food.image) {
        imageRef = await uploadImageToSanity(food.image);
      }

      const sanityFood = {
        _type: 'food',
        name: food.name,
        category: food.category || null,
        price: food.price,
        originalPrice: food.originalPrice || null,
        tags: food.tags || [],
        description: food.description || '',
        available: food.available !== undefined ? food.available : true,
        image: imageRef
          ? {
              _type: 'image',
              asset: {
                _type: 'reference',
                _ref: imageRef,
              },
            }
          : undefined,
      };

      console.log('Uploading food to Sanity:', sanityFood.name);
      const result = await client.create(sanityFood);
      console.log(`Food uploaded successfully: ${result._id}`);
    }

    for (const chef of chefs) {
      console.log(`Processing chef: ${chef.name}`);

      let imageRef = null;
      if (chef.image) {
        imageRef = await uploadImageToSanity(chef.image);
      }

      const sanityChef = {
        _type: 'chef',
        name: chef.name,
        position: chef.position || null,
        experience: chef.experience || 0,
        specialty: chef.specialty || '',
        description: chef.description || '',
        available: chef.available !== undefined ? chef.available : true,
        image: imageRef
          ? {
              _type: 'image',
              asset: {
                _type: 'reference',
                _ref: imageRef,
              },
            }
          : undefined,
      };

      console.log('Uploading chef to Sanity:', sanityChef.name);
      const result = await client.create(sanityChef);
      console.log(`Chef uploaded successfully: ${result._id}`);
    }

    console.log('Data import completed successfully!');
  } catch (error) {
    console.error('Error importing data:', error);
  }
}

importData();
```



```tsx
"use client";

import { client } from "@/sanity/lib/client";
import { useState, useEffect } from "react";
import { useParams } from "next/navigation";
import { urlFor } from "../../../sanity/lib/image";
import Image from "next/image";
import { useCart } from "../../context/CartContext";

interface Product {
  _id: string;
  title: string;
  slug: { current: string };
  description: string;
  imageSrc: { asset: { url: string } };
  price: number;
}

const getProduct = async (slug: string): Promise<Product> => {
  const product = await client.fetch(
    `*[_type == "product" && slug.current == "${slug.toLowerCase()}"][0]`
  );
  return product as Product;
};

export default function ProductsPage() {
  const [product, setProduct] = useState<Product | null>(null);
  const [isLoading, setIsLoading] = useState(false);
  const params = useParams();
  const { addItem } = useCart();

  useEffect(() => {
    const fetchProduct = async () => {
      setIsLoading(true);
      try {
        if (typeof params.slug === "string") {
          const productData = await getProduct(params.slug);
          setProduct(productData);
        } else {
          console.error("Slug is not a string:", params.slug);
        }
      } catch (error) {
        console.error("Error fetching product:", error);
      } finally {
        setIsLoading(false);
      }
    };
    if (params.slug) {
      fetchProduct();
    }
  }, [params.slug]);

  if (isLoading) {
    return <div className="bg-white flex justify-center items-center h-screen">Loading...</div>;
  }
  if (!product) {
    return <div className="bg-white flex justify-center items-center h-screen">No product found.</div>;
  }

  const handleAddToCart = () => {
    if (product) {
      addItem({
        id: product._id,
        name: product.title,
        price: product.price,
        quantity: 1,
        image: urlFor(product.imageSrc.asset).url(),
      });
      alert(`${product.title} has been added to the cart!`);
    }
  };

  return (
    <div className="py-24 mx-auto bg-white w-full">
      <div className="w-full px-4 gap-10 lg:px-72 sm:gap-10 py-24 mx-auto bg-white flex flex-col lg:flex-row justify-center">
        <div className="lg:w-4/5 mx-auto flex flex-wrap bg-white justify-center">
          <Image
            src={urlFor(product.imageSrc.asset).url()}
            width={500}
            height={500}
            alt={product.title}
            className="bg-white w-full h-auto object-cover"
          />
        </div>
        <div className="lg:w-1/2 w-full lg:pl-10 lg:py-6 mt-6 lg:mt-0 bg-white text-center lg:text-left">
          <h1 className="bg-white text-gray-900 text-6xl title-font font-bold mb-5">
            {product.title}
          </h1>
          <p className="bg-white leading-relaxed mt-10">{product.description}</p>
          <p className="bg-white title-font text-3xl text-gray-600 font-bold mt-10">
            ${product.price}
          </p>
          <button
            onClick={handleAddToCart}
            className="flex items-center justify-center border-2 border-white bg-[#FF9F0D] w-[10rem] text-xl text-gray-100 mb-5 py-3 mt-5"
          >
            Add to Cart
          </button>
        </div>
      </div>
      <div className="bg-white container lg:py-24 mx-auto">
        <div className="bg-white flex items-center gap-4">
          <button className="flex items-center justify-center border-2 border-white bg-[#FF9F0D] w-[10rem] text-xl text-gray-100 mb-5 py-3">
            Description
          </button>
          <h1 className="bg-white text-gray-800">Reviews (24)</h1>
        </div>
        <div className="bg-white text-gray-400">
          <h1 className="bg-white">
            Nam tristique porta ligula, vel viverra sem eleifend nec. Nulla sed
            purus augue, eu euismod tellus. Nam mattis eros nec mi sagittis
            sagittis. Vestibulum suscipit cursus bibendum. Integer at justo eget
            sem auctor auctor eget vitae arcu. Nam tempor malesuada porttitor.
            Nulla quis dignissim ipsum. Aliquam pulvinar iaculis justo, sit amet
            interdum sem hendrerit vitae. Vivamus vel erat tortor. Nulla
            facilisi. In nulla quam, lacinia eu aliquam ac, aliquam in nisl.
          </h1>
          <h1 className="bg-white mt-5">
            Suspendisse cursus sodales placerat. Morbi eu lacinia ex. Curabitur
            blandit justo urna, id porttitor est dignissim nec. Pellentesque
            scelerisque hendrerit posuere. Sed at dolor quis nisi rutrum
            accumsan et sagittis massa. Aliquam aliquam accumsan lectus quis
            auctor. Curabitur rutrum massa at volutpat placerat. Duis sagittis
            vehicula fermentum. Integer eu vulputate justo. Aenean pretium odio
            vel tempor sodales. Suspendisse eu fringilla leo, non aliquet sem.
          </h1>
        </div>
        <div className="bg-white mt-5">
          <h1 className="bg-white text-xl text-gray-800 mb-5">Key Benefits</h1>
          <ul className="bg-white list-disc ml-6">
            <li className="bg-white text-gray-400">
              Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            </li>
            <li className="bg-white text-gray-400">
              Maecenas ullamcorper est et massa mattis condimentum.
            </li>
            <li className="bg-white text-gray-400">
              Vestibulum sed massa vel ipsum imperdiet malesuada id tempus nisl.
            </li>
            <li className="bg-white text-gray-400">
              Etiam nec massa et lectus faucibus ornare congue in nunc.
            </li>
            <li className="bg-white text-gray-400">
              Mauris eget diam magna, in blandit turpis.
            </li>
          </ul>
        </div>
      </div>
      <div className="bg-white container px-5 mx-auto">
        <div>
          <h1 className="bg-white text-3xl font-bold text-gray-800 mb-5">
            Similar Products
          </h1>
        </div>
        <div className="bg-white flex flex-wrap -m-4">
          <div className="bg-white lg:w-1/4 md:w-1/2 p-4 w-full mb-10">
            <a className="bg-white block relative rounded overflow-hidden">
              <Image
                height={500}
                width={1000}
                alt="ecommerce"
                className="bg-white object-cover object-center w-full h-auto block"
                src="/p1.png"
              />
            </a>
            <div className="bg-white mt-4">
              <h3 className="bg-white text-black title-font mb-1 text-lg font-bold">
                Fresh Lime
              </h3>
              <p className="bg-white mt-1 text-yellow-500 text-xl">$38.00</p>
            </div>
          </div>
          <div className="bg-white lg:w-1/4 md:w-1/2 p-4 w-full">
            <a className="bg-white block relative rounded overflow-hidden">
              <Image
                height={500}
                width={500}
                alt="ecommerce"
                className="bg-white object-cover object-center w-full h-auto block"
                src="/p2.png"
              />
            </a>
            <div className="bg-white mt-4">
              <h3 className="bg-white text-lg font-bold title-font mb-1 text-black">
                Chocolate Muffin
              </h3>
              <p className="bg-white mt-1 text-yellow-500 text-xl">$28.00</p>
            </div>
          </div>
          <div className="bg-white lg:w-1/4 md:w-1/2 p-4 w-full">
            <a className="bg-white block relative rounded overflow-hidden">
              <Image
                height={500}
                width={500}
                alt="ecommerce"
                className="bg-white object-cover object-center w-full h-auto block"
                src="/p3.png"
              />
            </a>
            <div className="bg-white mt-4">
              <h3 className="bg-white text-black text-lg font-bold title-font mb-1">
                Burger
              </h3>
              <p className="bg-white mt-1 text-yellow-500 text-xl">$21.00</p>
            </div>
          </div>
          <div className="bg-white lg:w-1/4 md:w-1/2 p-4 w-full">
            <a className="bg-white block relative rounded overflow-hidden">
              <Image
                height={500}
                width={500}
                alt="ecommerce"
                className="bg-white object-cover object-center w-full h-auto block"
                src="/p4.png"
              />
            </a>
            <div className="bg-white mt-4">
              <h3 className="bg-white text-black title-font mb-1 text-lg font-bold">
                Fresh Lime
              </h3>
              <p className="bg-white mt-1 text-yellow-500 text-xl">$38.00</p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```



```tsx
"use client";

import { client } from "@/sanity/lib/client";
import { useState, useEffect } from "react";
import { urlFor } from "@/sanity/lib/image";
import Image from "next/image";

interface Chef {
  _id: string;
  name: string;
  position: string;
  experience: number;
  specialty: string;
  image: { asset: { url: string } };
  description: string;
  available: boolean;
}

export default function ChefComponent() {
  const [chefs, setChefs] = useState<Chef[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  // Fetch chef data on component mount
  useEffect(() => {
    const fetchChefs = async () => {
      setIsLoading(true);
      try {
        // Fetch all chefs
        const chefQuery = `*[_type == "chef"]{
          _id,
          name,
          position,
          experience,
          specialty,
          image,
          description,
          available
        }`;
        const chefs = await client.fetch(chefQuery);
        setChefs(chefs);
      } catch (error) {
        console.error("Error fetching chefs:", error);
        setError("Failed to fetch chefs. Please try again later.");
      } finally {
        setIsLoading(false);
      }
    };

    fetchChefs();
  }, []);

  if (isLoading) {
    return <div className="  bg-white flex justify-center items-center h-screen">Loading chefs...</div>;
  }

  if (error) {
    return <div className="  bg-white flex justify-center items-center h-screen">{error}</div>;
  }

  if (!chefs || chefs.length === 0) {
    return (
      <div className="  bg-white flex justify-center items-center h-screen">
        <h1 className=" bg-white ">Chef List</h1>
        <p className=" bg-white ">No chefs available. Please check the following:</p>
        <ul className=" bg-white ">
          <li className=" bg-white ">Is the Sanity client correctly configured?</li>
          <li className=" bg-white ">Does the dataset contain chef data?</li>
          <li className=" bg-white ">Is the GROQ query correct?</li>
          <li className=" bg-white ">Are there any errors in the terminal logs?</li>
        </ul>
      </div>
    );
  }

  return (
    <div className="  bg-gray-50 min-h-screen py-8">
      <div className=" bg-white container mx-auto px-4">
        {/* Grid with 4 columns */}
        <div className=" bg-white grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-6">
          {chefs.map((chef) => (
            <div key={chef._id} className="  bg-white rounded-lg shadow-md overflow-hidden transform transition-transform hover:scale-105">
              <div className=" bg-white relative h-[30rem] w-full">
                <Image
                  src={urlFor(chef.image).url()}
                  alt={chef.name}
                  fill
                  className=" bg-white object-cover"
                />
              </div>
              <div className=" bg-white p-4">
                <h2 className=" bg-white text-xl font-semibold mb-2">{chef.name}</h2>
                <p className=" bg-white text-gray-600">{chef.position}</p>
                <p className=" bg-white text-gray-600">{chef.specialty}</p>
                <p className=" bg-white text-gray-600">{chef.experience} years of experience</p>
                <p className=" bg-white text-gray-600">{chef.description}</p>
                <p className={`bg-white mt-2 ${chef.available ? "text-green-500" : "text-red-500"}`}>
                  {chef.available ? "Available" : "Not Available"}
                </p>
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}
```



```tsx
"use client";

import { client } from "@/sanity/lib/client";
import { useState, useEffect } from "react";
import { urlFor } from "@/sanity/lib/image";
import Image from "next/image";
import { Select } from "@/components/ui/select";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import Link from "next/link";

interface FoodItem {
  _id: string;
  name: string;
  category: string;
  price: number;
  originalPrice?: number;
  tags?: string[];
  image: { asset: { url: string } };
  description: string;
  available: boolean;
}

interface Category {
  id: string;
  label: string;
}

export default function Home() {
  const [foods, setFoods] = useState<FoodItem[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [latestProducts, setLatestProducts] = useState<FoodItem[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  // Fetch all data on component mount
  useEffect(() => {
    const fetchData = async () => {
      setIsLoading(true);
      try {
        // Fetch all food items
        const foodQuery = `*[_type == "food"]{
          _id,
          name,
          category,
          price,
          originalPrice,
          tags,
          image,
          description,
          available
        }`;
        const foods = await client.fetch(foodQuery);
        setFoods(foods);

        // Fetch categories (you can fetch these from Sanity or use static data)
        const categories = [
          { id: 'sandwiches', label: 'Sandwiches' },
          { id: 'burger', label: 'Burger' },
          { id: 'chicken', label: 'Chicken Chup' },
          { id: 'drink', label: 'Drink' },
          { id: 'pizza', label: 'Pizza' },
          { id: 'nonveg', label: 'Non Veg' },
          { id: 'uncategorized', label: 'Uncategorized' }
        ];
        setCategories(categories);

        // Fetch latest products (e.g., the last 4 added items)
        const latestProducts = foods.slice(0, 4); // Use the first 4 items as latest products
        setLatestProducts(latestProducts);
      } catch (error) {
        console.error("Error fetching data:", error);
        setError("Failed to fetch data. Please try again later.");
      } finally {
        setIsLoading(false);
      }
    };

    fetchData();
  }, []);

  const StarRating = () => (
    <div className=" bg-white white-section bg-transparent flex items-center space-x-1">
      {[...Array(5)].map((_, i) => (
        <svg key={i} className=" bg-white bg-transparent w-4 h-4" fill="#FF9F0D" viewBox="0 0 24 24">
          <path d="M12 17.27l-6.18 3.41 1.64-7.03-5.52-4.78 7.19-.61L12 2l2.87 7.27 7.19.61-5.52 4.78 1.64 7.03L12 17.27z" />
        </svg>
      ))}
    </div>
  );

  if (isLoading) {
    return <div className="  bg-white flex justify-center items-center h-screen">Loading...</div>;
  }

  if (error) {
    return <div className="  bg-white flex justify-center items-center h-screen">{error}</div>;
  }

  if (!foods || foods.length === 0) {
    return (
      <div className=" bg-white flex justify-center items-center h-screen">
        <h1 className="bg-white">Food Menu</h1>
        <p className="bg-white">No food items available. Please check the following:</p>
        <ul className="bg-white">
          <li className="bg-white">Is the Sanity client correctly configured?</li>
          <li className="bg-white">Does the dataset contain food items?</li>
          <li className="bg-white">Is the GROQ query correct?</li>
          <li className="bg-white">Are there any errors in the terminal logs?</li>
        </ul>
      </div>
    );
  }

  return (
    <div className="  bg-gray-50 min-h-screen py-8">
      <div className=" bg-white container mx-auto px-4 flex flex-col lg:flex-row gap-8">
        {/* Left Section */}
        <div className=" bg-white w-full lg:w-3/4">
          <div className=" bg-white flex flex-col sm:flex-row justify-between items-center mb-8 p-4 rounded-lg shadow-sm">
            <div className=" bg-white flex items-center gap-4 mb-4 sm:mb-0">
              <span className=" bg-white text-lg font-medium">Sort By:</span>
              <Select>
                <option className="bg-white">Newest</option>
                <option className="bg-white">Price: Low to High</option>
                <option className="bg-white">Price: High to Low</option>
              </Select>
            </div>
            <div className=" bg-white flex items-center gap-4">
              <span className=" bg-white text-lg font-medium">Show:</span>
              <Select>
                <option className="bg-white">12 Items</option>
                <option className="bg-white">24 Items</option>
                <option className="bg-white">36 Items</option>
              </Select>
            </div>
          </div>

          <div className=" bg-white grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
            {foods.map((food) => (
              <Link key={food._id} href={`/shop/${food.name}`} className=" bg-white rounded-lg shadow-md overflow-hidden transform transition-transform hover:scale-105">
                <div className=" bg-white relative h-64 w-full">
                  <Image
                    src={urlFor(food.image).url()}
                    alt={food.name}
                    fill
                    className=" bg-white object-cover"
                  />
                </div>
                <div className=" bg-white p-4">
                  <h3 className=" bg-white text-lg font-semibold mb-2 whitespace-nowrap">{food.name}</h3>
                  <p className=" bg-white text-gray-600">{food.description}</p>
                  {food.tags && (
                <div className=" bg-white mt-2">
                  <strong className="bg-white">Tags:</strong>
                  <ul className=" bg-white flex flex-wrap gap-2">
                    {food.tags.map((tag, index) => (
                      <li key={index} className="  bg-gray-200 px-2 py-1 rounded">
                        {tag}
                      </li>
                    ))}
                  </ul>
                </div>
              )}
              <p className={`mt-2  bg-white ${food.available ? "text-green-500" : "text-red-500"}`}>
                {food.available ? "Available" : "Not Available"}
              </p>
                  <div className=" bg-white flex items-center justify-between">
                    <div className=" bg-white flex items-center gap-2">
                      <span className=" bg-white text-[#FF9F0D] text-xl font-bold">${food.price}</span>
                      {food.originalPrice && (
                        <span className=" bg-white text-gray-400 line-through">${food.originalPrice}</span>
                      )}
                    </div>
                  </div>
                </div>
              </Link>
            ))}
          </div>

          {/* Pagination */}
          <div className=" bg-white flex justify-center mt-12">
            <nav className=" bg-white flex items-center gap-2">
              <Button variant="outline" className=" bg-white p-2">
                <svg className=" bg-white w-5 h-5" fill="currentColor" viewBox="0 0 24 24">
                  <path d="M15.41 16.59L10.83 12l4.58-4.59L14 6l-6 6 6 6 1.41-1.41z" />
                </svg>
              </Button>
              {[1, 2, 3].map((page) => (
                <Button
                  key={page}
                  variant={page === 2 ? "default" : "outline"}
                  className={page === 2 ? "bg-[#FF9F0D]" : ""}
                >
                  {page}
                </Button>
              ))}
              <Button variant="outline" className=" bg-white p-2">
                <svg className=" bg-white w-5 h-5" fill="currentColor" viewBox="0 0 24 24">
                  <path d="M8.59 16.59L13.17 12 8.59 7.41 10 6l6 6-6 6-1.41-1.41z" />
                </svg>
              </Button>
            </nav>
          </div>
        </div>

        {/* Right Section */}
        <div className=" bg-white w-full lg:w-1/4 space-y-8">
          {/* Search */}
          <div className=" bg-white p-6 rounded-lg shadow-sm">
            <div className=" bg-white flex gap-2">
              <Input
                type="text"
                placeholder="Search Products..."
                className="  flex-grow bg-gray-50"
              />
              <Button className=" bg-[#FF9F0D] hover:bg-[#e68906] px-6">
                <svg className=" bg-white w-5 h-5" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                  <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z" />
                </svg>
              </Button>
            </div>
          </div>

          {/* Categories */}
          <div className=" bg-white p-6 rounded-lg shadow-sm">
            <h2 className=" bg-white text-xl font-bold mb-4">Categories</h2>
            <div className=" bg-white space-y-3">
              {categories.map(({ id, label }) => (
                <div key={id} className=" bg-white flex items-center gap-3">
                  <Input
                    type="checkbox"
                    id={id}
                    className=" bg-white w-4 h-4 text-[#FF] rounded"
                  />
                  <label htmlFor={id} className=" bg-white text-gray-700 hover:text-[#FF9F0D]pointer">
                    {label}
                  </label>
                </div>
              ))}
            </div>
          </div>

          {/* Price Filter */}
          <div className=" bg-white p-6 rounded-lg shadow-sm">
            <h2 className=" bg-white text-xl font-bold mb-4">Filter by Price</h2>
            <div className=" bg-white space-y-4">
              <Input
                type="range"
                min="0"
                max="1000"
                className=" bg-white w-full"
              />
              <div className=" bg-white flex justify-between text-sm text-gray-600">
                <span className="bg-white">$0</span>
                <span className="bg-white">$1000</span>
              </div>
              <Button className=" bg-white w[#FF9F0D] hover:bg-[#e68906]">
                Apply Filter
              </Button>
            </div>
          </div>

          {/* Latest Products */}
          <div className=" bg-white p-6 rounded-lg shadow-sm">
            <h2 className=" bg-white text-xl font-bold mb-4">Latest Products</h2>
            <div className=" bg-white space-y-4">
              {latestProducts.map((product) => (
                <div key={product._id} className=" bg-white flex gap-4 items-center">
                  <div className=" bg-white relative h-16 w-16 flex-shrink-0">
                    <Image
                      src={urlFor(product.image).url()}
                      alt={product.name}
                      fill
                      className=" bg-white object-cover rounded"
                    />
                  </div>
                  <div>
                    <h3 className=" bg-white font-medium">{product.name}</h3>
                    <StarRating />
                    <p className=" bg-white text-[#FF9F0D] font-semibold">${product.price}</p>
                  </div>
                </div>
              ))}
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```