Market Blace Builder Hackathon 2025

Day 3 Report

API Integration & Data Migration

Project Name: “Foodtuck: A Q-Commerce Marketplace”

Date: 18/01/2025

1. Objective

The goal of Day 3 was to integrate the Sanity CMS backend with the Next.js frontend, enabling dynamic data fetching for both food and chef content. This included setting up schemas, configuring the client, and displaying fetched data on the website.

2. Tasks Completed

1. **Sanity CMS Setup**:
    a. Created schemas for **Food** and **Chefs**.
    b. Configured Sanity client to connect the Next.js application with the Sanity backend.
    c. **Performed Migration**: Ensured the schemas were correctly migrated to the Sanity backend.
2. **Data Fetching**:
    a. Fetched food and chef data from the Sanity backend dynamically.
    b. Ensured data was displayed correctly on the respective pages.
3. **Dynamic Routing**:
    a. Implemented dynamic routes for both **Food Detail Pages** and **Chef Detail Pages**.
4. **Error Handling**:
    a. Added basic error handling to ensure fallback messages when no data is available.

3. Schemas

a) Food Schema

The schema for food items included fields such as name, category, price, tags, image, description, and availability.

1. **Code Snippet:**

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

b) Chef Schema

The schema for chefs included fields such as name, specialty, bio, and image.

2. **Code Snippet:**

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

4. Sanity Client Configuration

Configured the client in `client.ts` to enable communication between Sanity and Next.js.

3. **Code Snippet:**

```
import { createClient } from 'next-sanity'

import { apiVersion, dataset, projectId } from '../env'

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  useCdn: true, // Set to false if statically generating pages, using ISR or tag-based revalidation
})

import imageUrlBuilder from '@sanity/image-url';

const builder = imageUrlBuilder(client);
export const urlFor = (source: any) => builder.image(source);
export default client;
```

5. Sanity Client Configuration

Migration ensures that the defined schemas in the codebase are applied to the Sanity backend, enabling proper data structure and integrity.

4. **Code Snippet:**

```
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
```

```javascript
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.SANITY_API_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();
```

6. Pages Created

a) Food Page

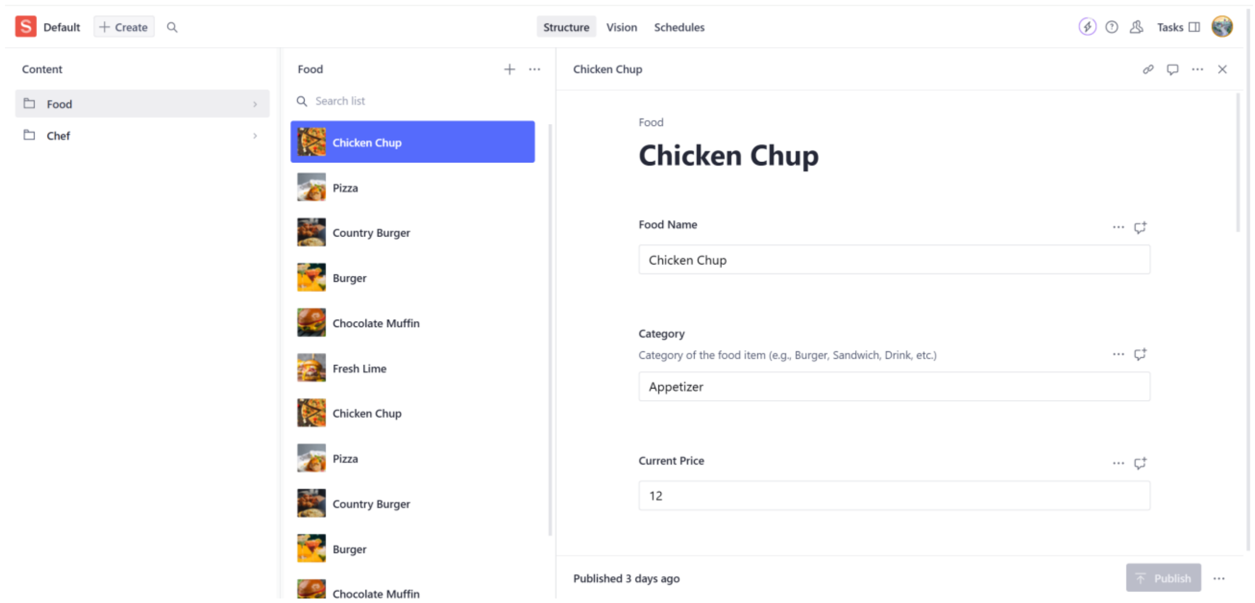
- Displays a list of food items fetched from Sanity.
- Implements dynamic routing to individual Food Detail Pages.

b) Chefs Page

- Displays a list of chefs fetched from Sanity.
- Implements dynamic routing to individual Chef Detail Pages.

7. Screenshots of Functionality and Results

1. **Successfully Migrate 'food' data into Sanity CMS**



Chicken Chup
Pizza
Country Burger
Burger
Chocolate Muffin
Fresh Lime
Chicken Chup
Pizza
Country Burger
Burger
Chocolate Muffin

Food

Chicken Chup

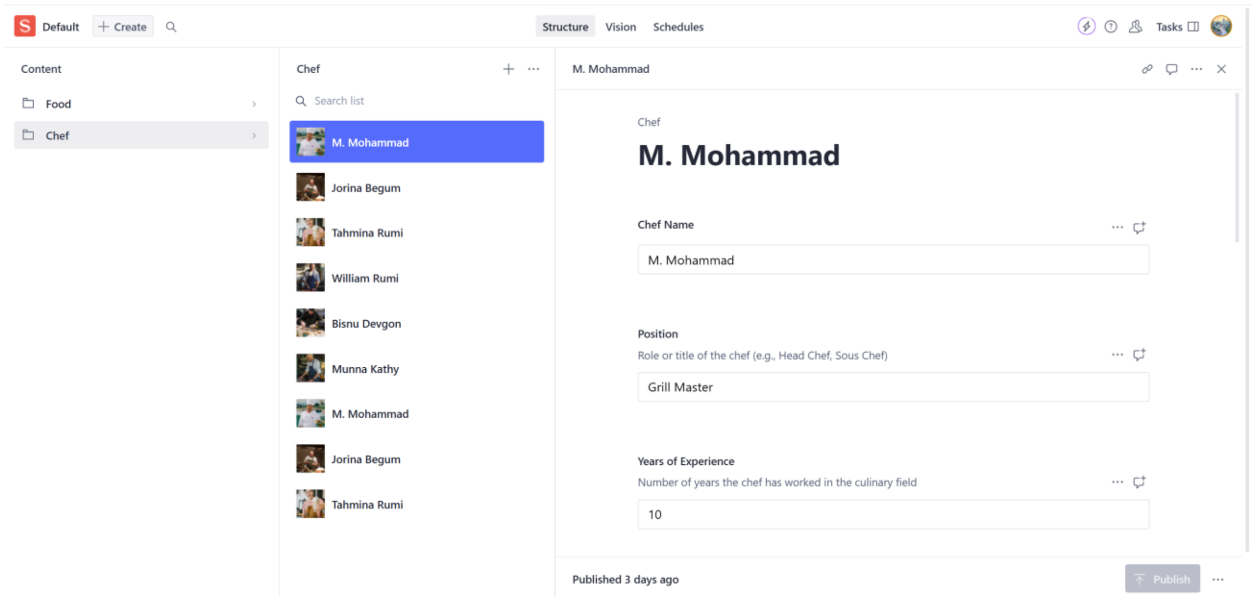
Food Name
Chicken Chup

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

Current Price
12

Published 3 days ago

2. **Successfully Migrate 'chef' data into Sanity CMS**


Default + Create Structure Vision Schedules Tasks

Content
- Food
- Chef

Chef
Search list
- M. Mohammad
- Jorina Begum
- Tahmina Rumi
- William Rumi
- Bisnu Devgon
- Munna Kathy
- M. Mohammad
- Jorina Begum
- Tahmina Rumi

M. Mohammad

Chef
M. Mohammad

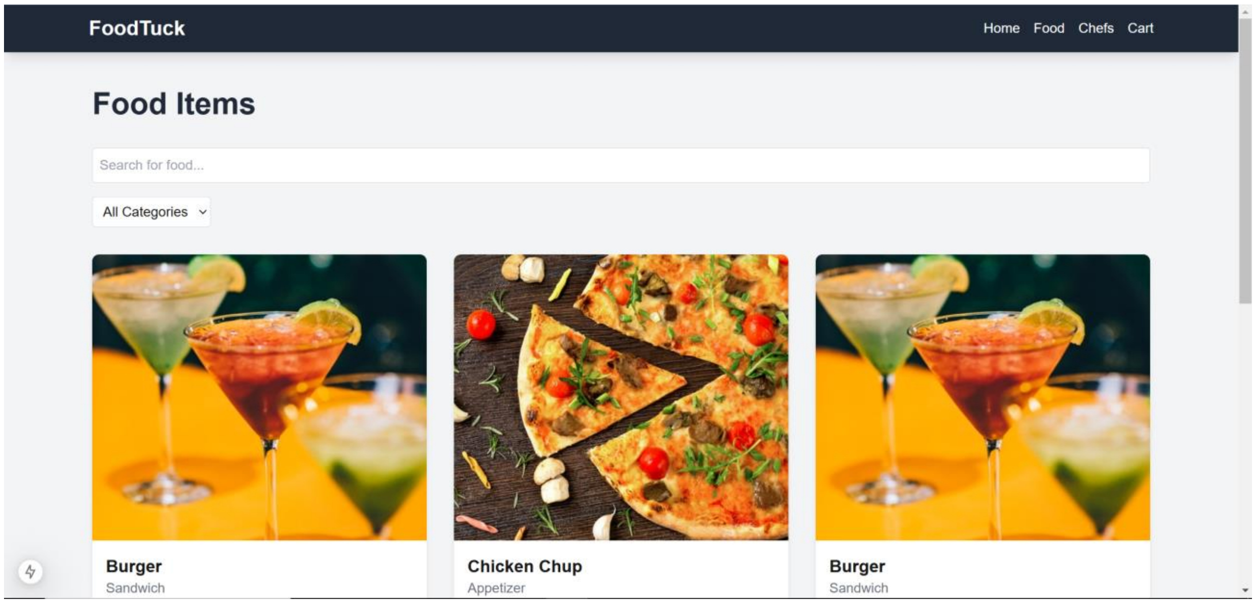
Chef Name
M. Mohammad

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

Years of Experience
Number of years the chef has worked in the culinary field
10

Published 3 days ago
Publish

3. **Successfully Rendered/fetched all food data in food page**


FoodTuck
Home Food Chefs Cart

Food Items

Search for food...

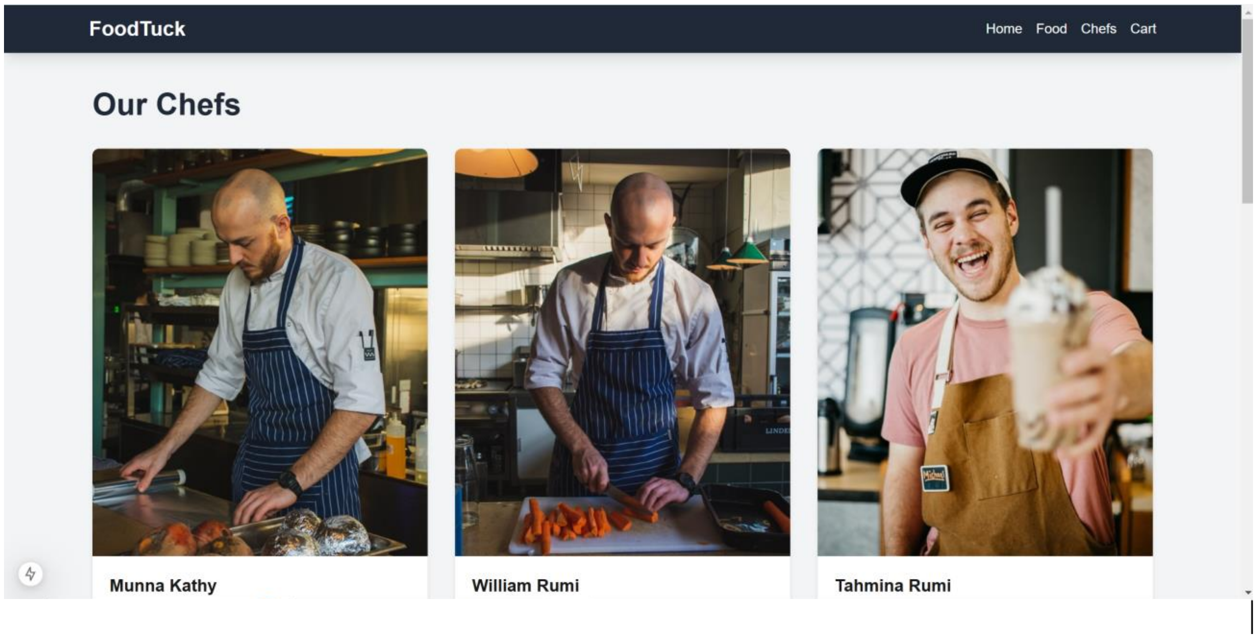
All Categories

Burger
Sandwich

Chicken Chup
Appetizer

Burger
Sandwich

5. **Successfully Rendered/fetched all chefs data in chef page**



8. Challenges Faced

1. **Data Fetching**:
   a. Ensured correct queries were used to fetch data dynamically from Sanity.
2. **Dynamic Routing**:
   a. Implemented `/food/[id]` and `/chefs/[id]` routes for detailed pages.
3. **Error Handling**:
   a. Handled empty or missing data gracefully by showing fallback UI.

9. Conclusion

The integration of Sanity CMS was successful, with dynamic data fetching for food and chef pages fully implemented. Dynamic routing and error handling were also achieved, laying the foundation for further enhancements.

Prepared by: Mahnoor M. Ayub