Day 03 – API Integration Report of Food Tunk

Process of API Integration

1. **Overview:**
   - The API integration connects an external API providing food and chef data to a Sanity CMS project.
2. **Steps Taken:**
   - **Environment Setups:**
     - Used `.env` to load environment variables from `.env.local`.
     - Key Variables Include:
       - `NEXT_PUBLIC_PROJECT_ID`
       - `NEXT_PUBLIC_DATASET`
       - `SANITY_API_TOKEN`
   - **Data Fetching:**
     - Made concurrent API calls using `axios` to fetch food and chef data.
     - Endpoints Accessed:
       - `https://sanity-nextjs-rouge.vercel.app/api/foods`
       - `https://sanity-nextjs-rouge.vercel.app/api/chefs`

Understanding the Provided API:

1. **1st API: Foods**
   - **URL:** `https://sanity-nextjs-rouge.vercel.app/api/foods`
   - **Key Details:**
     - Endpoint: `/foods`
       - Likely returns a list of available food items.
       - Each food item may include:
         - `name`: The name of the food item.
         - `description`: A brief description of the food.
         - `price`: The cost of the item.
         - `tags`: Type of food (e.g., healthy, sweet, crispy).
         - `availability`: Item is available or not.
   - **Data Use:**
     - Display food items on the frontend.
     - Create dynamic routes for all products.
2. **2nd API: Chefs**
   - **URL:** `https://sanity-nextjs-rouge.vercel.app/api/chefs`
   - **Key Details:**
     - Endpoint: `/chefs`
       - Likely returns a list of chefs.
       - Each chef may include:
         - `name`: The name of the chef.
         - `position`: The position of the chef (e.g., Head Chef, Sous Chef).
         - `specialty`: The chef's area of expertise (e.g., Italian cuisine, desserts).

            - `experience`: Number of years the chef has been in the industry.
            - `associatedFoods`: Reference to foods prepared by the chef.
    - **Data Use:**
        - Display chef profiles on the frontend.
        - Show a chef's details alongside the food items they prepare.

Migration Process

1. **Approach: Using the Provided API**
    - Leveraged two APIs:
        - Foods API: `https://sanity-nextjs-rouge.vercel.app/api/foods`
        - Chefs API: `https://sanity-nextjs-rouge.vercel.app/api/chefs`
    - Automated the following:
        - Fetching data from the APIs.
        - Transforming the data to match Sanity's schema.
        - Uploading images to Sanity's asset management system.
        - Creating documents for food and chef entities in Sanity.
2. **Script Breakdown**
    - **Environment Configuration:**
        - Used the `dotenv` library to load environment variables from `.env.local`.
        - Credentials include:
            - `NEXT_PUBLIC_SANITY_PROJECT_ID`: The Sanity project ID.
            - `NEXT_PUBLIC_SANITY_DATASET`: The Sanity dataset name.
            - `SANITY_API_TOKEN`: The API token for write access.
    - **Sanity Client Initialization:**
        - Initialized the Sanity client with the provided environment variables.
        - Set `useCdn` to `false` to fetch the latest data during operations.
    - **Data Fetching:**
        - Data fetched concurrently from Foods and Chefs APIs using `Promise.all` to reduce execution time.
    - **Image Upload to Sanity:**
        - The `uploadImageToSanity` function downloads and uploads images to Sanity, returning a reference ID for each uploaded asset.
        - Images handled as optional fields to accommodate missing cases.
    - **Data Transformation and Upload:**
        - **Foods:**
            - Fields such as `name`, `category`, `price`, and `tags` mapped directly.
            - Optional fields like `originalPrice` and `description` assigned default values if missing.
            - Uploaded images linked to the document using Sanity's `_ref` system.
        - **Chefs:**
            - Fields like `name`, `position`, `experience`, and `specialty` included.

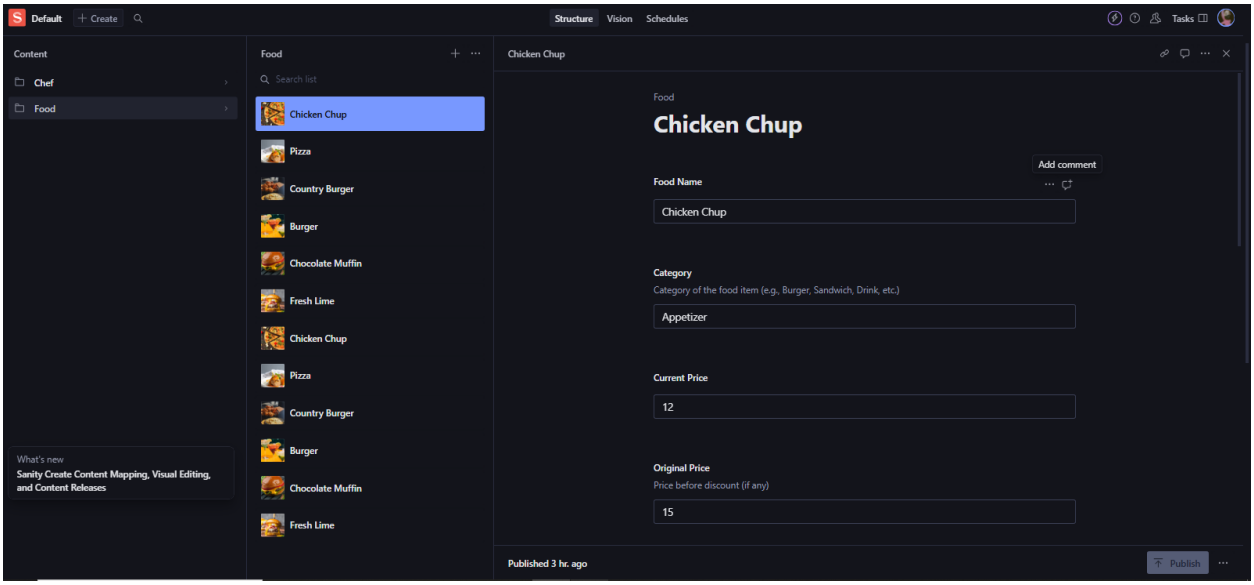
        - Optional fields handled similarly to food documents.
    - **Error Handling:**
        - Logged errors during API requests, image uploads, or document creation to identify and resolve issues.

Advantages of This Approach:

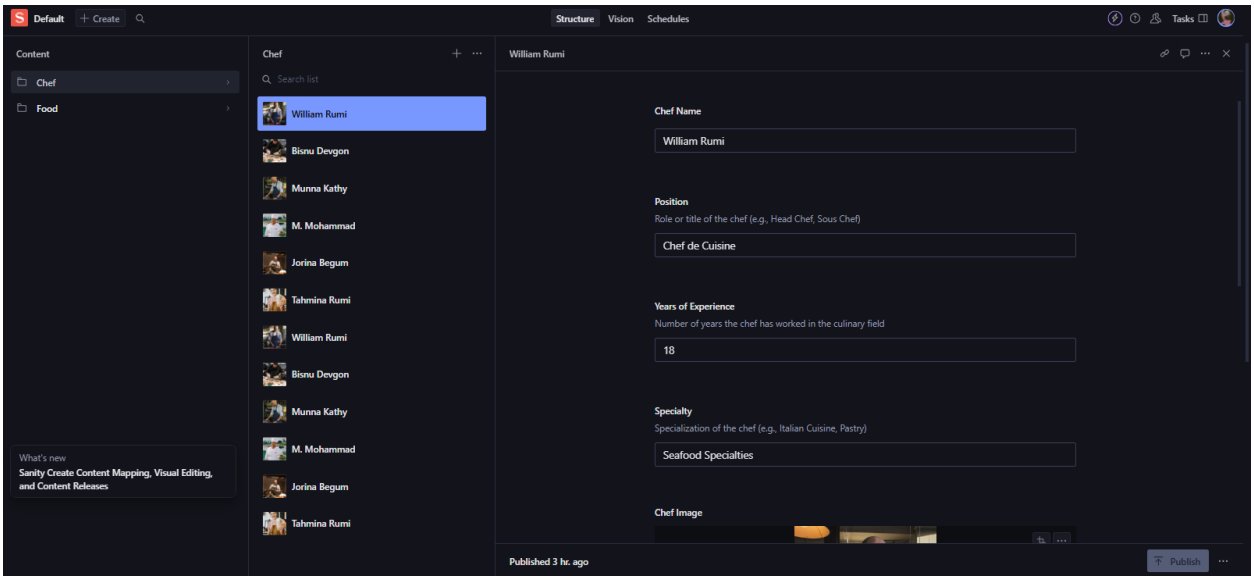
1. **Efficiency:**
    - Automates the entire migration process, saving time and effort.
2. **Scalability:**
    - Handles large datasets without manual intervention.
3. **Accuracy:**
    - Reduces human errors associated with manual data entry.
4. **Reusability:**
    - The script can be reused for future migrations with minimal modifications.

Data Successfully Imported on Sanity:

1. Foods Data



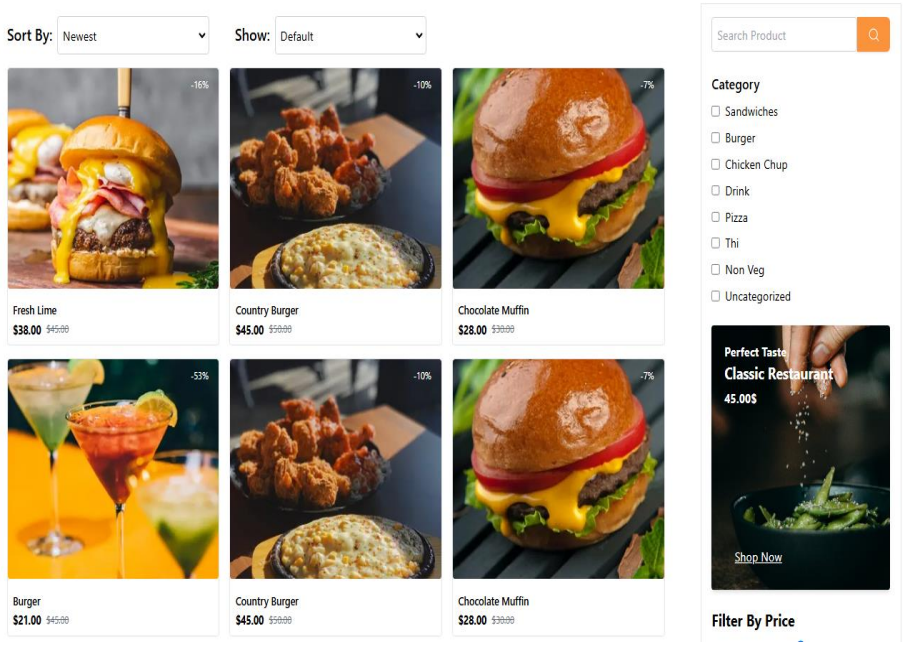
2. Chefs Data



Display on Frontend:

1. **Setting Up the Sanity Client:**
   - To fetch data from Sanity, a client is required for communication between the frontend and the Sanity backend.
     - `projectId`: The unique identifier for your Sanity project.
     - `dataset`: The dataset to query (e.g., production).
     - `useCdn`: Enabled for faster read operations by caching results.
     - `apiVersion`: The version of the Sanity API to ensure compatibility.
2. **Fetching Data Using GROQ:**
   - GROQ is used to query data stored in Sanity. Queries can be customized to fetch specific documents or fields.
     - Fetch Food Data Using GROQ Query.
     - Fetch Chefs Data Using GROQ Query.
     - Data successfully displayed in the frontend:

Foods Data.



- • Chefs Data

-

3. **Dynamic Routing for Details:**

   Created dynamic routes for food or chef details using a `[slug].ts` file in the page's directory.

Self-Validation Checklist:

1. **API Understanding:**
    - Status: ✔ Verified clear understanding of API endpoints, request methods, and expected responses.
2. **Schema Validation:**
    - Status: ✔ Confirmed schemas in Sanity are properly configured, matching the data structure.
3. **Data Migration:**
    - Status: ✔ Validated successful data migration from the API into Sanity.
4. **API Integration in Next.js:**
    - Status: ✔ Ensured frontend fetches data from Sanity using GROQ queries and displays it correctly.
5. **Submission Preparation:**
    - Status: ✔ Met all requirements, ensured code is clean, and prepared project for submission.

Conclusion:

1. **Efficient Automation:**
    - The migration script streamlined the import of API data into Sanity, saving significant time compared to manual input.
2. **Seamless Integration:**
    - The use of GROQ queries enabled smooth integration of Sanity data with the frontend.
3. **Real-World Practice:**
    - This experience simulated real-world scenarios of handling APIs, validating schemas, and integrating a headless CMS with Next.js.

Outcome:

The project is now functional, with APIs integrated, data migrated into Sanity, and displayed dynamically on the frontend. These skills are essential for handling complex client projects in a professional environment.