# ***DAY 03 – API INTEGRATION REPORT OF FOOD TUNK***

## *Process Of API Integration*

1. ***Overview:*** The AP integration is connects an external API providing foods and chefs data to a Sanity CMS Project
2. ***Steps Taken:***

- **Environment Setups**:
  - ***i.*** Used .env to load environment variables from .env.local.
  - ***ii.*** Key Variables Includes: ○ ***NEXT_PUBLIC_PROJECT_ID*** ○ ***NEXT_PUBLIC_DATASET*** ○ ***SANITY_API_TOKEN***

***2. Data Fetching:***

○ Make concurrent API calls using axios to fetch food and chef data. ○ **Endpoints Accessed** :

i. https://sanity-nextjs-rouge.vercel.app/api/foods ii. https://sanity-nextjs-rouge.vercel.app/api/chefs

## *Understand the Provided API:*

### *1st API: Foods*

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

This API provides data related to food items. Below are the key details to note:

1. **Key Endpoint**: `/foods` ○ This endpoint likely returns a list of available food items.
   - Each food item may include details such as:
     - **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
2. **Data Use:**
   - Display food items on the frontend, ○ Create a dynamic routes for all products

### 2nd API: Chefs

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

This API provides data related to chefs. Below are the key details to note:

**1. Key Endpoint**: `/chefs`

- This endpoint likely returns a list of chefs.
- Each chef may include details such as:
  - **Name**: The name of the chef.
  - **Position**: The postion of the chef (e.g, Head Chef, Sous Chef, Executive Chefs) ○ **Specialty**: The chef's area of expertise (e.g., Italian cuisine, desserts). ○ **Experience**: The number of years the chef has been in the industry.
  - **Associated Foods**: A reference to the foods prepared by this chef.

**2. 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

The migration process leverages two APIs:

- **Foods API:** `https://sanity-nextjs-rouge.vercel.app/api/foods`
- **Chefs API:** `https://sanity-nextjs-rouge.vercel.app/api/chefs`

Instead of manually inputting data into Sanity, the script automates 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

- The script uses the `dotenv` library to load environment variables from `.env.local`. This ensures secure handling of credentials, including:
  ○ `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*

The Sanity client is initialized with the provided environment variables. The `useCdn` flag is set to `false` to ensure the latest data is fetched during operations.

### Data Fetching

- Data is fetched concurrently from the Foods and Chefs APIs using `Promise.all`. This reduces the overall execution time.

### Image Upload to Sanity

- The `uploadImageToSanity` function downloads and uploads images to Sanity, returning a reference ID for each uploaded asset.
- Images are handled as optional fields to accommodate cases where images are missing.

### Data Transformation and Upload

- **Foods**:
  - Fields such as `name`, `category`, `price`, and `tags` are mapped directly. o Optional fields like `originalPrice` and `description` are assigned default values if missing.
  - Uploaded images are linked to the document using Sanity's `_ref` system. • **Chefs**:
  - Fields like `name`, `position`, `experience`, and `specialty` are included.
  - Optional fields are handled similarly to the food documents..

---

## *Migration Process*

### 1. Approach: Using the Provided API

The migration process leverages two APIs:

- **Foods API:** `https://sanity-nextjs-rouge.vercel.app/api/foods`
- **Chefs API:** `https://sanity-nextjs-rouge.vercel.app/api/chefs` Instead of manually inputting data into Sanity, the script automates 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

- The script uses the `dotenv` library to load environment variables from `.env.local`. This ensures secure handling of credentials, including:
  - `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

The Sanity client is initialized with the provided environment variables. The `useCdn` flag is set to `false` to ensure the latest data is fetched during operations.

### Data Fetching

- Data is fetched concurrently from the Foods and Chefs APIs using `Promise.all`. This reduces the overall execution time.

### Image Upload to Sanity

- The `uploadImageToSanity` function downloads and uploads images to Sanity, returning a reference ID for each uploaded asset.
- Images are handled as optional fields to accommodate cases where images are missing.

### Data Transformation and Upload

- **Foods**:
  - Fields such as `name`, `category`, `price`, and `tags` are mapped directly.
  - Optional fields like `originalPrice` and `description` are assigned default values if missing.
  - Uploaded images are linked to the document using Sanity's `_ref` system.
- **Chefs**:
  - Fields like `name`, `position`, `experience`, and `specialty` are included.
  - Optional fields are handled similarly to the food documents.

### Error Handling

- Errors during API requests, image uploads, or document creation are logged to help identify and resolve issues.

---

*Advantages of This Approach*

1. **Efficiency**: ○ Automates the entire migration process, saving time and effort.
2. **Scalability**: ○ Can handle 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.

## *FOODS API CALL:*

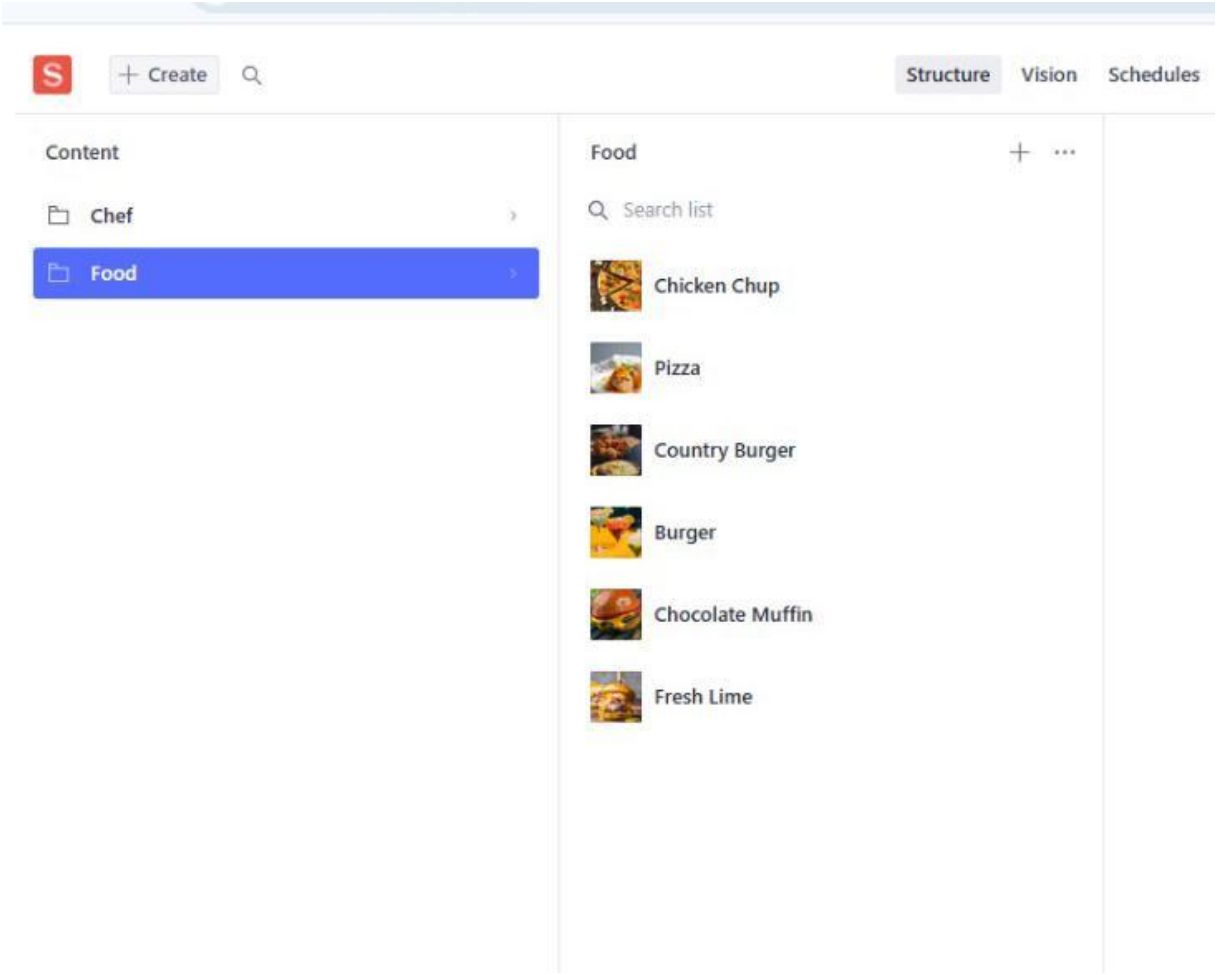
```
[
  {
    name: 'Chicken Chop',
    category: 'Appetizer',
    price: 12,
    originalPrice: 15,
    image: { _type: 'image', asset: [Object] },
    description: 'Crispy fried chicken bites served with dipping sauce.',
    available: true,
    tags: [ 'Sell', 'Crispy' ]
  },
  {
    name: 'Fresh Lime',
    category: 'Drink',
    price: 38,
    originalPrice: 45,
    image: { _type: 'image', asset: [Object] },
    description: 'Refreshing fresh lime drink made with natural ingredients.',
    available: true,
    tags: [ 'Healthy', 'Popular' ]
  },
  {
    available: true,
    tags: [ 'Sell', 'Sweet' ],
    name: 'Chocolate Muffin',
    category: 'Dessert',
    price: 28,
    originalPrice: 30,
    image: { _type: 'image', asset: [Object] },
    description: 'Soft and rich chocolate muffin topped with chocolate chips.'
  },
  {
    name: 'Country Burger',
    category: 'Sandwich',
    price: 45,
    originalPrice: 50,
    image: { _type: 'image', asset: [Object] },
    description: 'Classic country-style burger served with fries.',
    available: true,
    tags: [ 'Recommended' ]
  },
  {
    image: { _type: 'image', asset: [Object] },
    description: 'Juicy beef burger with fresh lettuce, tomatoes, and cheese.'
    available: true,
    tags: [ 'Popular' ],
    name: 'Burger',
    category: 'Sandwich',
    price: 21,
    originalPrice: 45
  },
```

## *CHEFS API CALL:*

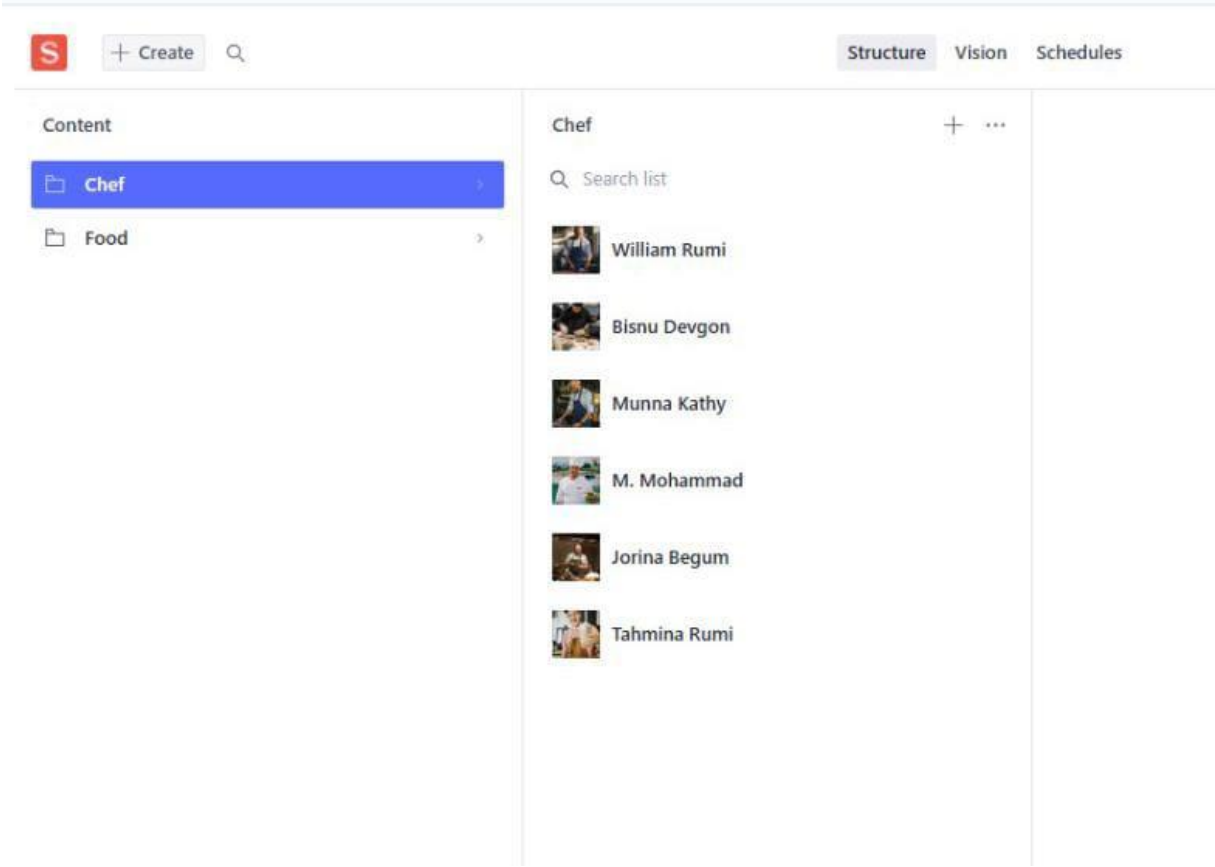
```
[
  {
    position: 'Head Chef',
    experience: 12,
    specialty: 'Italian Cuisine',
    image: { _type: 'image', asset: [Object] },
    description: 'Expert in crafting authentic Italian dishes and pastries.',
    available: true,
    name: 'Tahmina Rumi'
  },
  {
    description: 'Renowned for creating perfectly grilled meats and vegetables.',
    available: true,
    name: 'M. Mohammad',
    position: 'Grill Master',
    experience: 10,
    specialty: 'Grilled Dishes',
    image: { asset: [Object], _type: 'image' }
  },
  {
    image: { _type: 'image', asset: [Object] },
    description: 'Expert in international cuisines and menu planning.',
    available: true,
    name: 'Bisnu Devgon',
    position: 'Executive Chef',
    experience: 20,
    specialty: 'Global Cuisine'
  },
  {
    position: 'Chef de Cuisine',
    experience: 18,
    specialty: 'Seafood Specialties',
    image: { _type: 'image', asset: [Object] },
    description: 'Master of crafting exquisite seafood dishes with unique flavors.
    available: true,
    name: 'William Rumi'
  },
  {
    available: true,
    name: 'Jorina Begum',
    position: 'Sous Chef',
    experience: 8,
    specialty: 'Pastry and Desserts',
    image: { _type: 'image', asset: [Object] },
    description: 'Specializes in creative pastries and dessert innovations.'
  },
```

***Data Successfully Import On Sanity:***

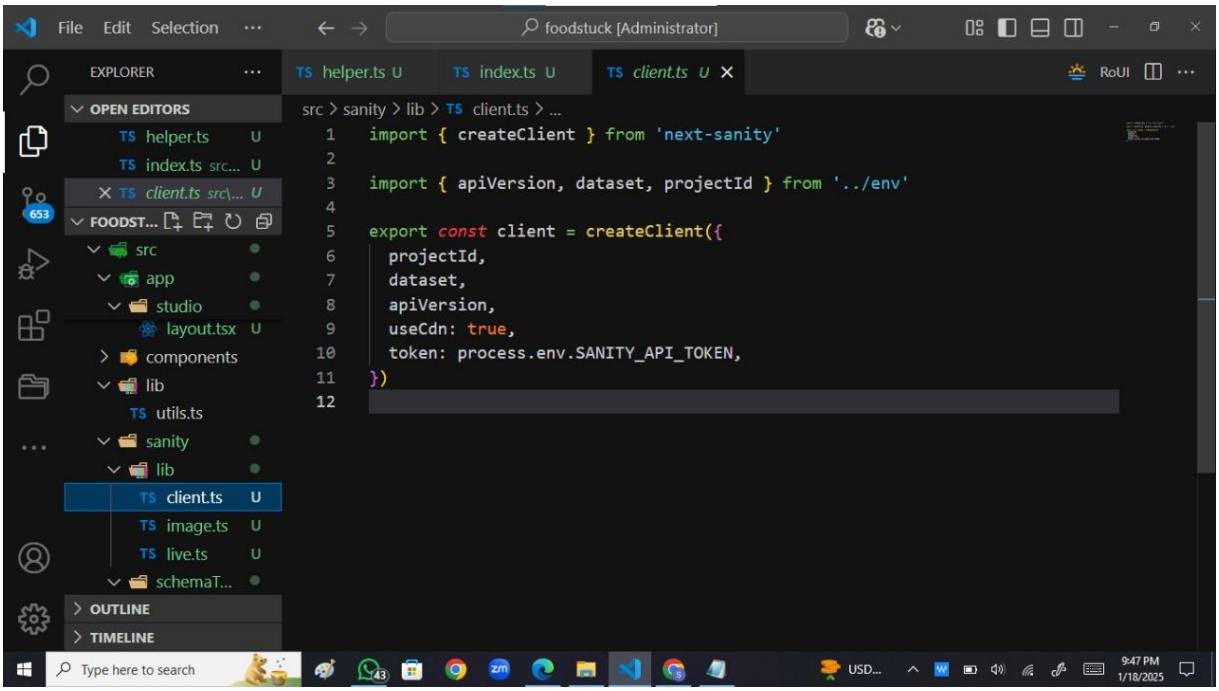
***1. Foods Data:***



1. ***Chefs Data***




***Display On My Frontend:***

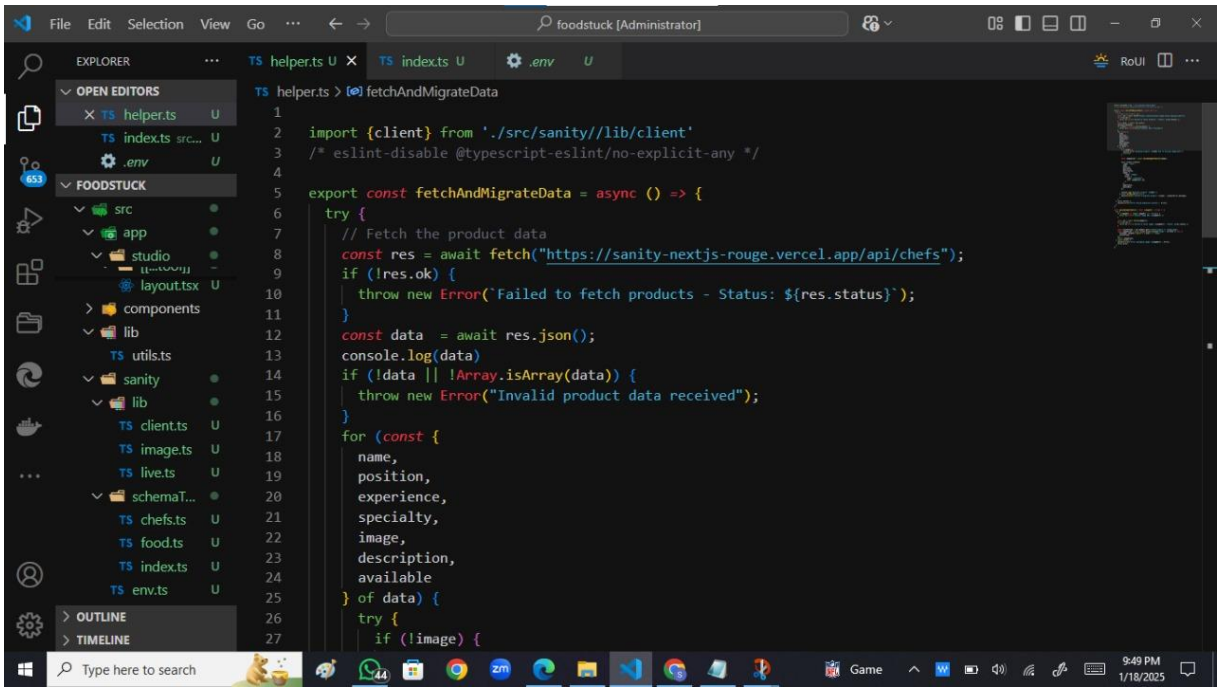


### 1. Setting Up the Sanity Client

To fetch data from Sanity, a client is required for communication between the frontend and the Sanity backend.

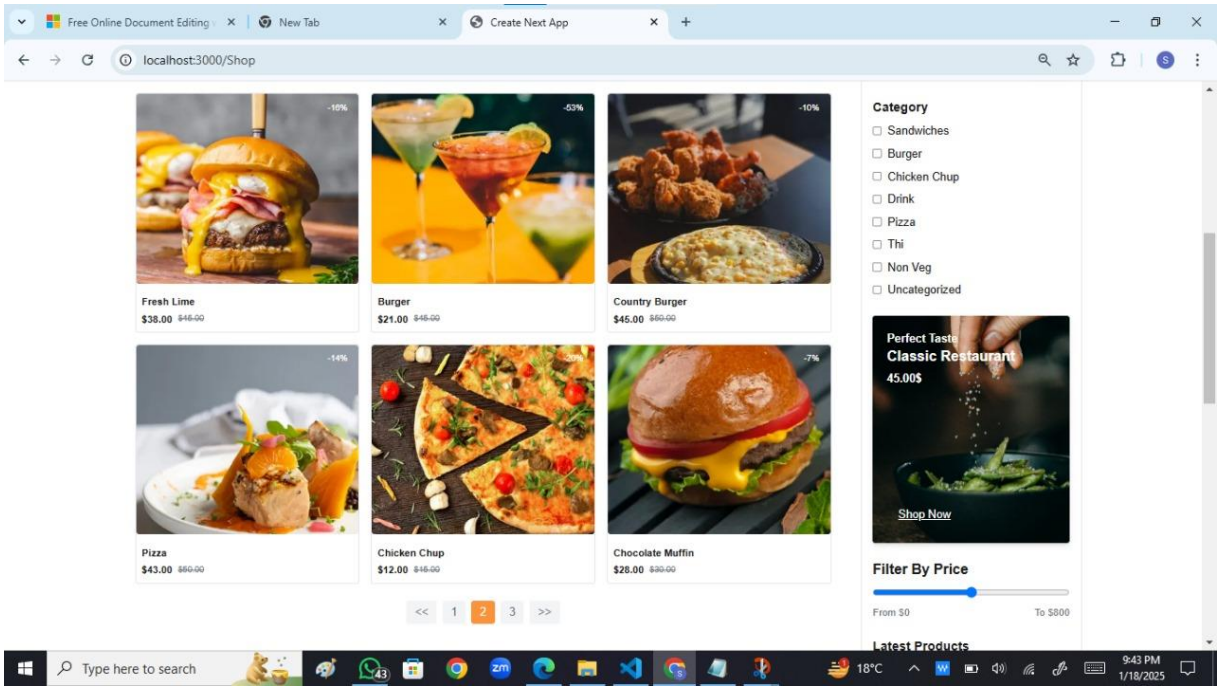
- **`projectId`**: The unique identifier for your Sanity project.
- **`dataset`**: The dataset to query (e.g., `production`).
- **`useCdn`**: Enabled for faster read operations by caching results.
- **`apiVersion`**: The version of the Sanity API to ensure compatibility.

- ***FETCH CHEFS DATA Using API***



## Data successfully displayed in the frontend:

- **Foods Data**




- **Chefs Data** :

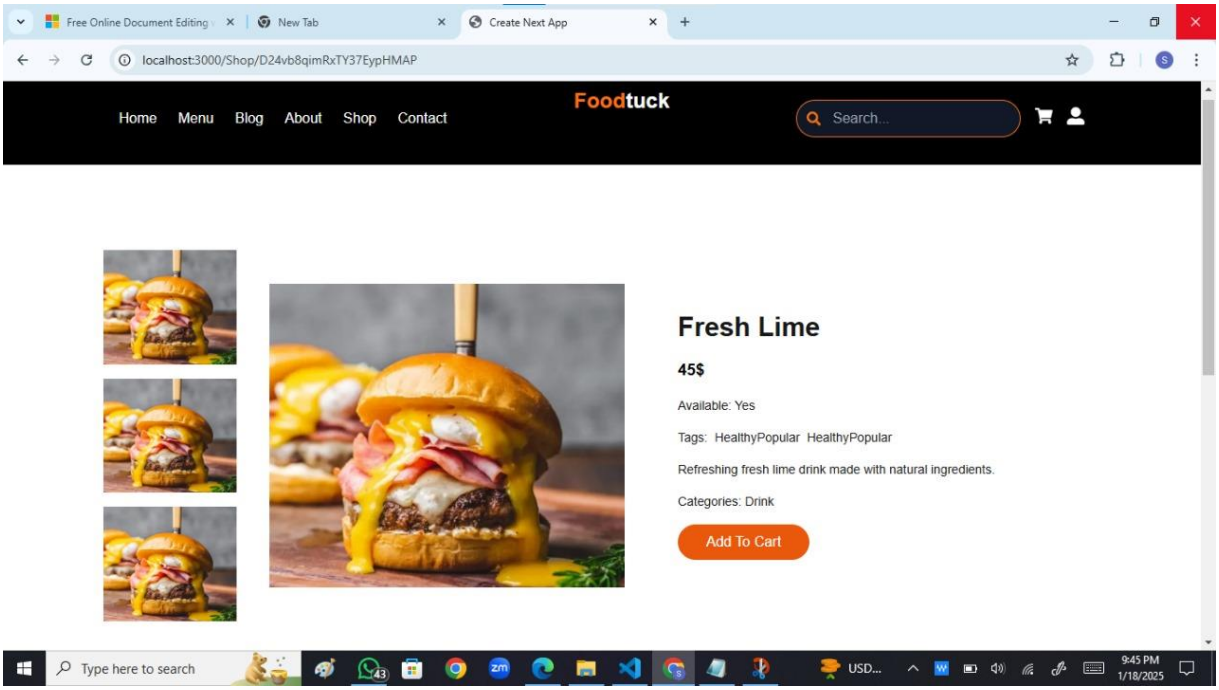
## *3.. **Dynamic Routing for Details***

To create a dynamic route for food or chef details, configure a `[slug].ts` file in the page's directory.

```ts
async function Productpage({ params }: { params: { slug: string } }) {
  const product:IProduct =
    await client.fetch(`*[_type == "food" && slug.current == $slug][0] {
      name,
      description,
      price,
      originalPrice,
      tags,
      "imageUrl": image.asset->url,
      "slug": slug.current,
    }`,{slug:params.slug});

```

- **Frontend Display:**




*Self-Validation Checklist*

### API Understanding

- **Status:** ✔
  *Verify that I have a clear understanding of how the API works, including its endpoints, request methods, and expected responses.*

---

### Schema Validation

- **Status:** ✔
  *Confirm that the schemas in Sanity are properly configured, matching the structure of the data to be migrated. Ensure required fields are set correctly and optional fields are handled appropriately.*

---

### Data Migration

- **Status:** ✔
  *Validate that the data from the API has been successfully migrated into Sanity. Check for completeness, accuracy, and proper image uploads.*

---

### API Integration in Next.js

- **Status:** ✔
  *Ensure that the frontend successfully fetches data from Sanity using the GROQ queries. Validate that the data is displayed correctly in the intended format.*

---

### Submission Preparation

- **Status:** ✔
  *Confirm that all requirements are met, code is clean and organized, and the project is ready for submission. Include documentation, screenshots, or demo links.*

## *Conclusion:*

The Day 3 tasks were completed successfully, as follows:

1. **Efficient Automation:** The migration script streamlined the import of API data into sanity, It saved 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 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 a complex client project in a professional environment.