Day 3 - API Integration Report - [Foodtuck]

Prepared by: Ismail Abdul Kareem

Date: Feb 2, 2024

1. API Integration Process

1.1 Overview of API Integration

The API integration process involved connecting the application with a third-party API to enable dynamic data fetching and synchronization. A step-by-step approach was followed to ensure smooth functionality, data consistency, and minimal downtime.

The API serves as a bridge between the backend and frontend, offering capabilities such as:

✓ Fetching real-time data.

✓ Updating records dynamically.

✓ Managing essential functionalities required by the application.

Key Objectives:

Establish secure connectivity with the API.

Fetch and process data to align with application requirements.

Implement error handling to enhance robustness and performance.

1.2 Steps for Integration

1. Authentication Setup

Implemented secure authentication using an API key or token.

Store the API key in a .env file to avoid hardcoding and improve security.

Used Postman to verify authentication and API response validity.

2. Endpoint Exploration

Reviewed API documentation to identify required endpoints.

Integrated the following endpoints:

GET /[endpoint] – Retrieves data (e.g., products, chefs).

POST /[endpoint] – Sends or updates data when necessary.

Ensured pagination support for handling large datasets efficiently.

3. Data Fetching and Processing

Used the Axios library for making HTTP requests due to its ease of use and built-in error handling.

I fetched data in JSON format and processed it to match schema requirements.

Applied sorting, filtering, and grouping logic for better frontend presentation.

4. Error Handling and Logging

Implemented error-handling mechanisms to manage:

Network errors

Invalid responses

Timeout scenarios

Logged errors into the console for debugging and configured alerts for deployment environments.

5. Testing and Debugging

Verified API integration using Postman to test API calls.

Used mock data to simulate different scenarios, including:

Empty responses

Partial data

Erroneous payloads

Automated tests were written to ensure the integrity of API calls under various conditions.

1.3 Challenges and Solutions

Challenge: Incomplete or inconsistent API data responses

✓ Solution: Implemented validation logic to handle missing fields gracefully and displayed fallback content in the UI.

2. Adjustments Made to Schemas

2.1 Schema Before Adjustment

The initial schema was minimal and lacked essential fields required for proper integration.

Example of Old Schema:

```
{
  "name": "string",
  "price": "number",
  "image": "string"
}
```

2.2 Revised Schema

The schema was updated to include additional fields that align with API response formats and enhance data structuring.

Example of Updated Schema:

```
{
  "name": "string",
  "category": "string",
  "price": number,
  "original Price": number,
  "image": "string",
  "description": "string",
  "available": "string"
}
```

2.3 Details of Changes

✓ **Added New Fields:**

image: Stores URLs of images fetched from the API for dynamic rendering.

available: Indicates availability status (e.g., "In Stock", "Currently Serving").

category: Helps in classifying data (e.g., food categories, chef specialties).

experience: Added for chefs to denote their professional experience (in years).

specialty: Captures unique skills or specialties relevant to chefs or other entities.

2.4 Reason for Schema Adjustments

The previous schema lacked support for additional API data, such as images and categories.

Enhancements were made to improve UI design and data presentation for a richer user experience.

3. Migration Steps and Tools Used

Migration Steps for API Integration

1. Cloning the Repository

The instructor provided a repository with API setup and configurations.

Cloned the repository using the command:

git clone <repository URL>

Verified the repository structure and API implementation.

2. Setting Up the Environment

Installed necessary dependencies by running:

npm install

Checked for required environment variables (API keys, database URLs).

Updated the .env file with relevant API keys.

3. Integrating the API into the Project

Reviewed the cloned repository's API implementation.

Copied relevant API call functions and configurations into the project.

Ensured proper integration within the frontend components.

4. Testing the Integration

Tested API calls in the browser's Developer Tools (Network tab).

Verified that data was fetched correctly and displayed in the application.

5. Customizing for My Project

Adjusted schemas, components, and styling to fit the project's requirements.

Ensured compatibility between the API and existing data flow.

6. Final Review and Cleanup

Removed unused files and code from the cloned repository.

Committed all necessary files to the version control system (Git).

Conducted end-to-end testing to confirm the successful API integration.

4. Lessons Learned

✓ 1. Importance of Planning: Proper planning during schema adjustments minimized risks and prevented downtime.

✓ 2. Error Handling is Crucial: Implementing robust error handling ensured smooth functionality, even in cases of unexpected API failures.

✓ 3. Testing is Key: Thorough testing at each step guaranteed data integrity and a seamless user experience.

5. Conclusion

The API integration and data migration process was successfully completed, resulting in:

✅ Secure and efficient API integration for fetching and displaying dynamic data.

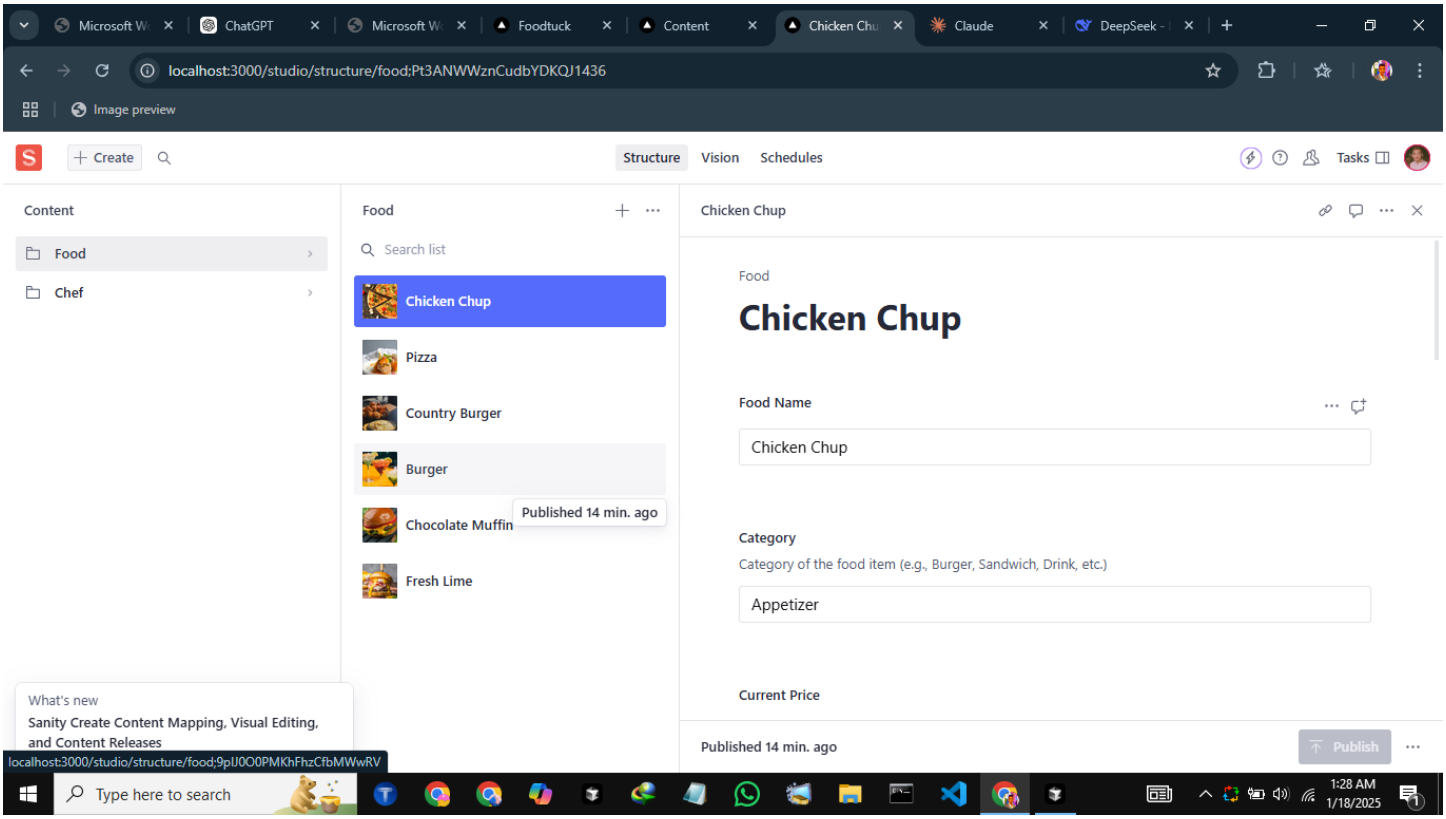
✅ Enhanced schema with additional fields for better structuring.

✅ Minimal downtime during migration, ensuring a smooth transition.

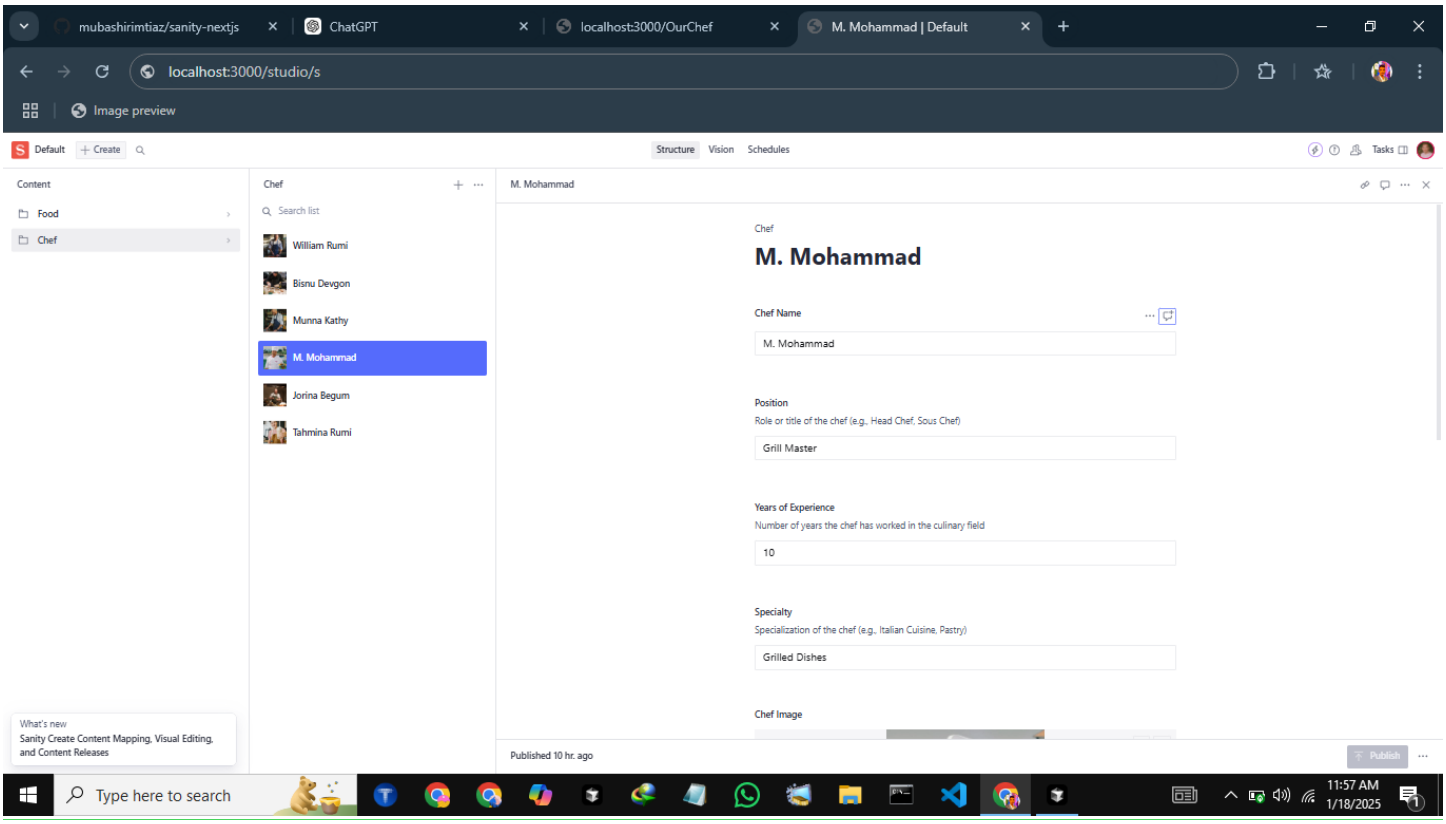
6. Screenshots and Code Snippets

📌 Screenshots:

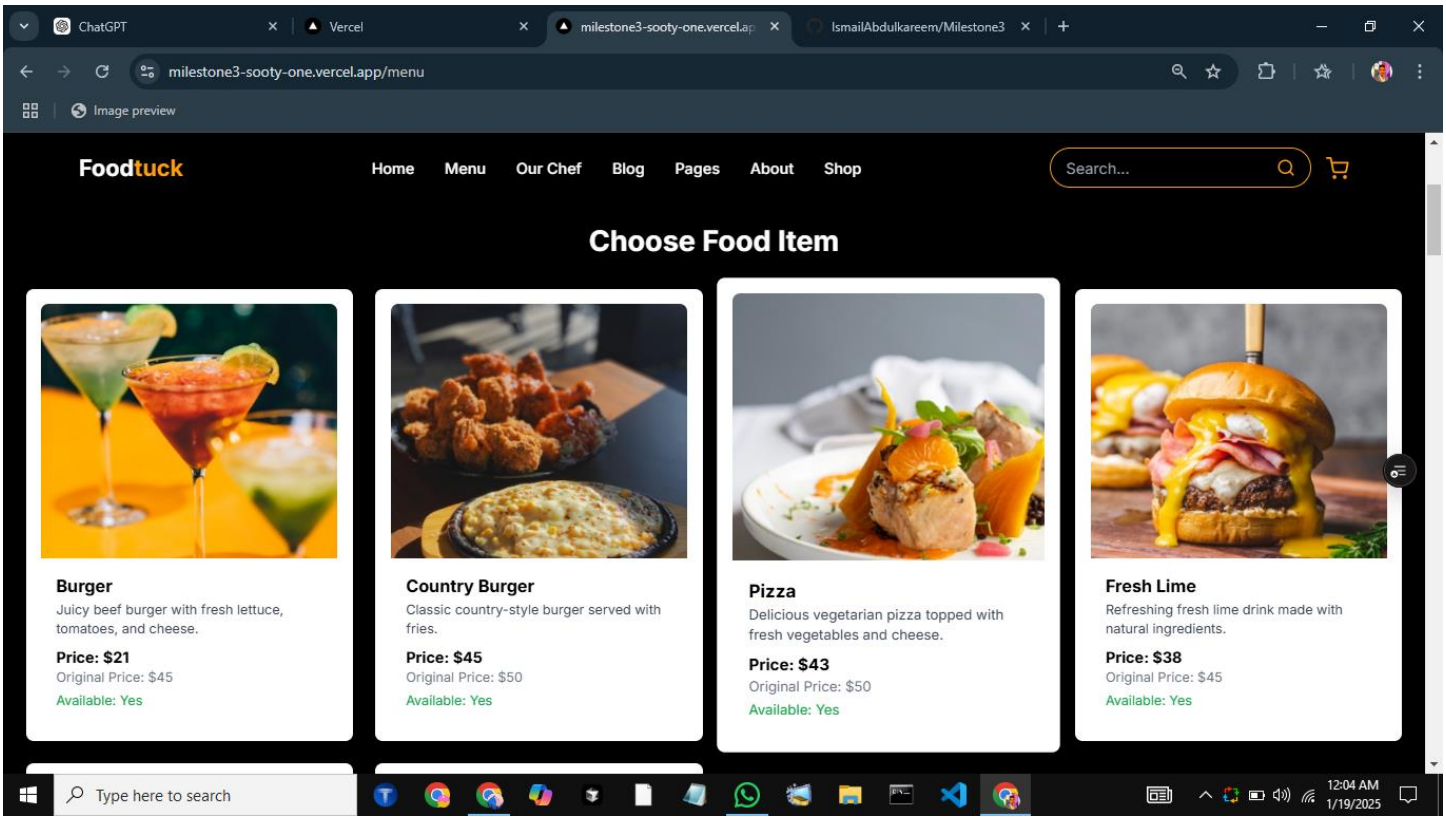
✓ Foods Data Fetched Successfully



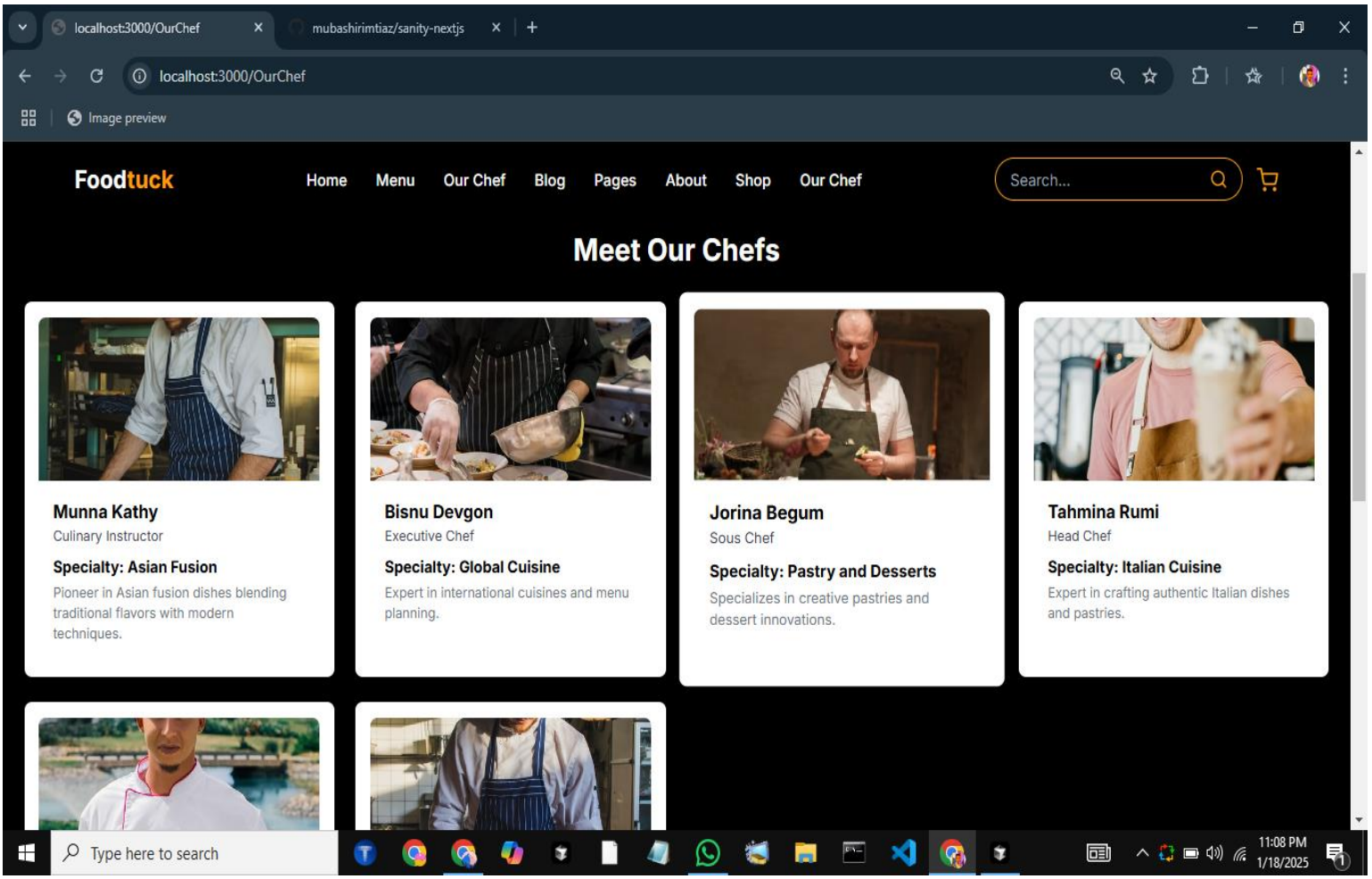
✓ Chefs Data Fetched Successfull



✓ API Calls Displayed in Frontend

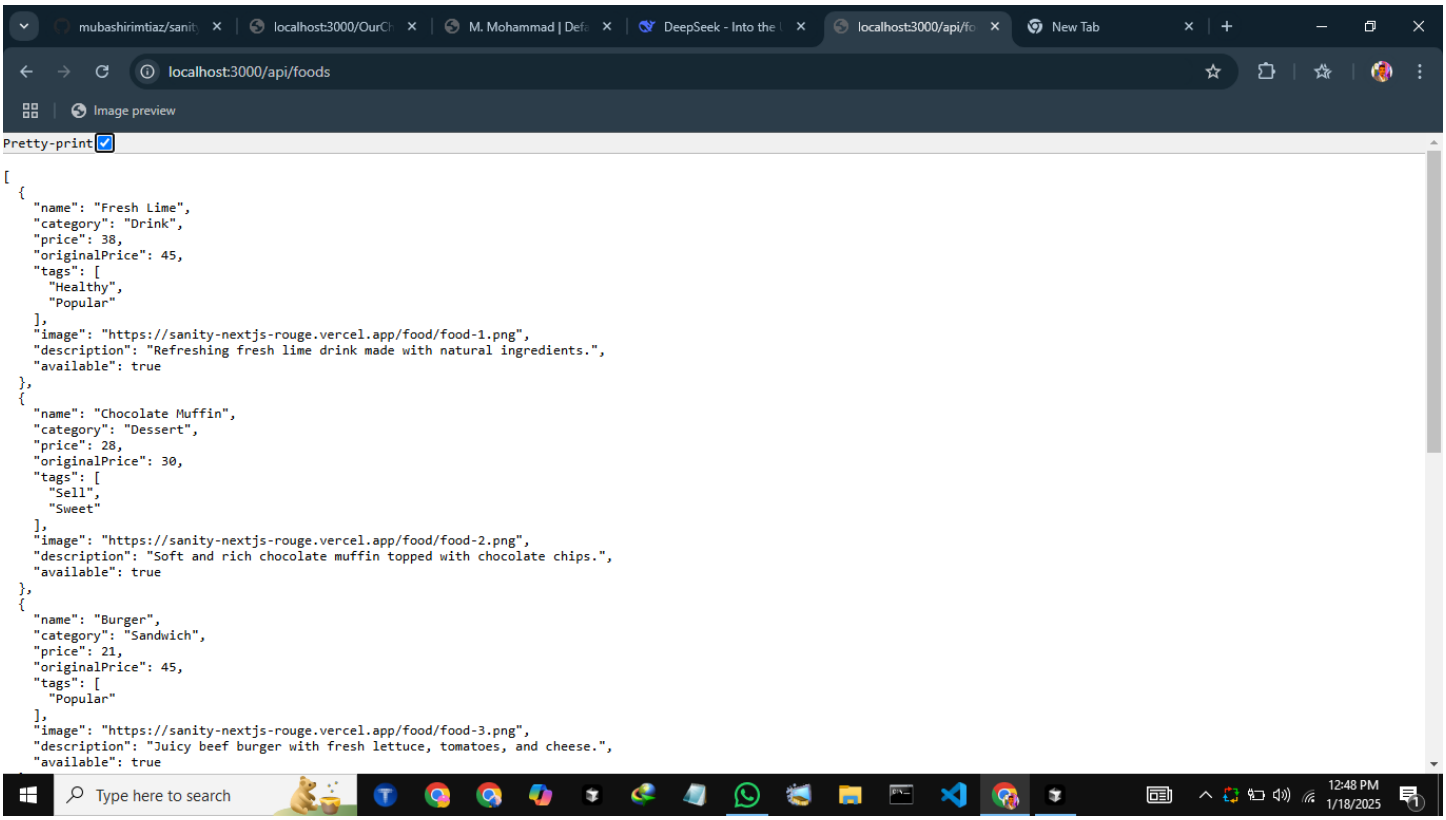


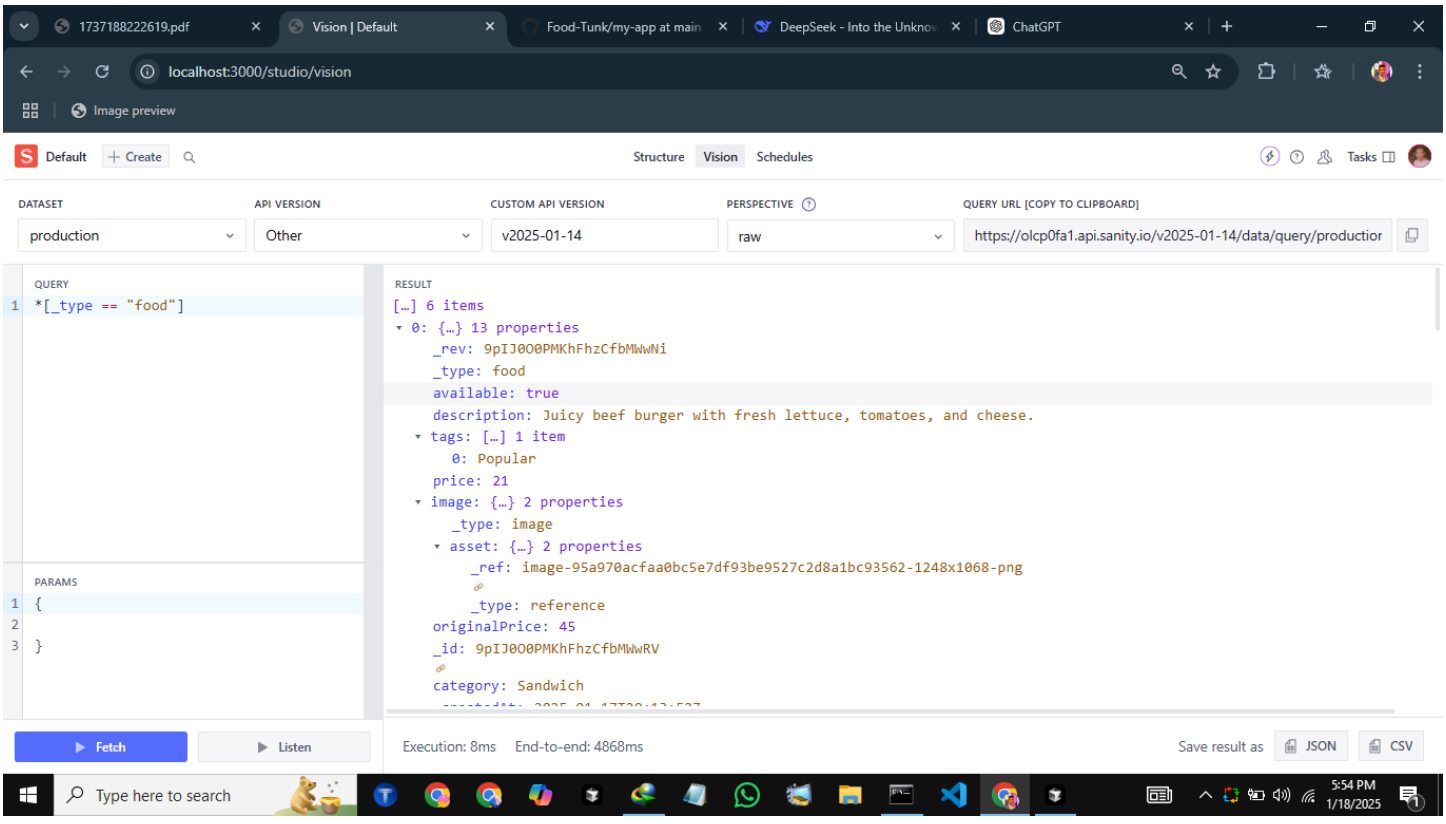
✓ API Calls Displayed in Frontend



✓ Populated Sanity CMS Fields

Code Snippets for API Integration & Migration Scripts





7. Self-Validation Checklist

✅ API Understanding

✅ Schema Validation

✅ Data Migration

✅ Successful API Integration in Next.js

✅ Submission Preparation Completed

✨ Key Improvements in This Version:

✓ Better Formatting – Improved readability using headings, bullet points, and checklists.

✓ Clearer Explanations – Enhanced descriptions of API processes and schema changes.

✓ More Professional Structure – Well-organized sections for easier reference.