Assignment 1: Recipe Finder Single Page Application

Frontend Development using HTML, CSS, JavaScript, and TheMealDB API

1. Objectives

- Get experience with HTML, CSS, and JavaScript for building interactive web pages.
- Learn to consume RESTful APIs and parse JSON responses.
- Understand the Fetch API and async/await patterns for asynchronous programming.
- Practice DOM manipulation to dynamically update page content.
- Build a Single Page Application (SPA) without using frameworks.
- Deploy a containerized application to Google Cloud Run.

1.1 General Directions

- This assignment is **frontend only**. No backend server code (Python/Flask, Node.js) is required.
- You will use **TheMealDB API**, which is free and requires no API key.
- The assignment will be graded using the latest version of Google Chrome.
- You must deploy your application to Google Cloud Run using the provided Dockerfile.
- Refer to the grading rubric and this document while developing. Piazza clarifications are part of the assignment specification.

2. Description

In this assignment, you will build a Single Page Application (SPA) that allows users to search for recipes using TheMealDB API. The application will display search results and allow users to view detailed recipe information.

2.1 Search Form

The user opens a web page (e.g., `index.html`). The search form contains:

- **Text Input:** A text box where users enter a meal name to search for (e.g., "Chicken", "Pasta", "Arrabiata").
- **Search Button:** Clicking this button initiates the search using the TheMealDB API.

Validation: If the user clicks Search without entering a keyword, display an error message or tooltip indicating the field is required.

2.2 Displaying Search Results

When a valid search is performed, the application should display results as a grid of cards. Each card should show:

- **Meal Thumbnail:** The image of the meal from the API response.
- **Meal Name:** The name of the recipe.
- **Category:** The category of the meal (e.g., "Vegetarian", "Seafood", "Dessert").

No Results: If the API returns no matching meals, display a user-friendly message such as "No recipes found. Try a different search term."

Loading State: While waiting for the API response, display a loading indicator (e.g., "Searching..." text or a spinner).

2.3 Recipe Detail View

When the user clicks on a meal card, the application should display a detailed view of the recipe. The detail view should include:

- **Full Image:** A larger version of the meal image.
- **Meal Name:** The name of the recipe as a heading.
- **Category & Area:** The category (e.g., "Vegetarian") and cuisine area (e.g., "Italian").

- **Ingredients List:** A list of all ingredients with their measurements. The API provides ingredients in fields strIngredient1 through strIngredient20 and measurements in strMeasure1 through strMeasure20.
- **Instructions:** The cooking instructions from the strInstructions field.
- **YouTube Video Link:** If available (strYoutube field), display a link to watch the recipe video. This link should open in a new tab.
- **Back to Results Button:** A button that returns the user to the search results without losing the previous results.

2.4 TheMealDB API Endpoints

TheMealDB provides a free API that requires no authentication. Use the following endpoints:

Search by Name:

```
https://www.themealdb.com/api/json/v1/1/search.php?s={query}
```

Replace `{query}` with the user's search term (e.g., `search.php?s=Arrabiata`).

Lookup by Meal ID:

```
https://www.themealdb.com/api/json/v1/1/lookup.php?i={mealId}
```

Replace `{mealId}` with the meal's ID from the search results (e.g., `lookup.php?i=52771`).

Key Fields in API Response:

Field	Description
`idMeal`	Unique identifier for the meal
`strMeal`	Name of the meal
`strCategory`	Category (e.g., Vegetarian, Seafood)
`strArea`	Cuisine area (e.g., Italian, Mexican)
`strInstructions`	Cooking instructions
`strMealThumb`	URL to the meal thumbnail image
`strYoutube`	URL to YouTube video (if available)
`strIngredient1-20`	Ingredient names (up to 20)
`strMeasure1-20`	Ingredient measurements (up to 20)

3. Deployment to Google Cloud Run

Your application must be deployed to Google Cloud Run using Docker. Follow these steps carefully.

3.1 Prerequisites

- A Google Cloud account with an active project.
- Google Cloud SDK (gcloud CLI) installed on your computer.
- Docker installed (for local testing, optional).

3.2 Project Structure

Organize your project files as follows:

```
project/
├── index.html
├── css/
│   └── styles.css
├── js/
│   └── app.js
├── Dockerfile
└── nginx.conf
```

3.3 Create the Dockerfile

Create a file named `Dockerfile` (no extension) in your project root with the following content:

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

```
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

3.4 Create the NGINX Configuration

Create a file named `nginx.conf` in your project root with the following content:

```
server {
    listen 8080;

    location / {
        root /usr/share/nginx/html;
        index index.html;
    }
}
```

Note: Cloud Run expects your container to listen on port 8080. This configuration tells NGINX to use that port.

3.5 Deploy to Cloud Run

Open your terminal in the project directory and run the following commands:

Step 1: Set your Google Cloud project ID:

```
gcloud config set project YOUR_PROJECT_ID
```

Step 2: Enable Google APIs:

```
gcloud services enable run.googleapis.com cloudbuild.googleapis.com
```

Step 3: Build and submit the container image:

```
gcloud builds submit --tag gcr.io/YOUR_PROJECT_ID/recipe-finder
```

Step 4: Deploy to Cloud Run:

```
gcloud run deploy recipe-finder --image gcr.io/YOUR_PROJECT_ID/recipe-finder --platform
managed --region us-central1 --allow-unauthenticated
```

Step 5: After deployment completes, the CLI will display your application URL:

```
https://recipe-finder-xxxxxxxxxx.us-central1.run.app
```

This is the URL you will submit for grading.

Step 6: Redeploy:

```
gcloud run deploy recipe-finder \
  --source . \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated
```

4. Notes and Restrictions

- You **cannot** use frontend frameworks or libraries (React, Angular, Vue, jQuery, etc.).
- You **cannot** use CSS libraries (Bootstrap, Tailwind, etc.).
- You **may** use a CSS reset stylesheet. If you do, specify its source in a comment.
- All API calls must use the `fetch()` API with async/await or Promises.
- Your application must work in the latest version of Google Chrome.
- Responsive design is **not required** for this assignment.

5. Submission Instructions

Submit the following to Brightspace:

1. **Cloud Run URL:** The URL of your deployed application (e.g., https://recipe-finder-xxx-uc.a.run.app).
2. **Source Code ZIP:** A ZIP file containing all your source code (index.html, css/, js/, Dockerfile, nginx.conf).

3. **AI Log (process_log.txt):** Documentation of your AI-assisted development process. See the AI Log Rubric for format requirements.

Deadline: See Brightspace for the due date. Late submissions will use grace days as per the syllabus.

6. Academic Integrity

This is an individual assignment. You may use AI tools (ChatGPT, Claude, GitHub Copilot, etc.) to assist with development, but you must:

- Document your AI usage in the process_log.txt file.
- Understand all code you submit and be able to explain it if asked.
- Not share your code with other students or use another student's code.
- Not post your solution publicly (GitHub, etc.) until after the semester ends.

Violations of academic integrity will be reported to the Office of Academic Integrity and may result in a failing grade for the course.