

Recipes Assessment Write-Up (Backend + Frontend)

Tech Stack: Python (FastAPI) + PostgreSQL + React (TypeScript/Vite) + Material UI

Author: Chinemelum Umealajekwu

Date: 12/29/2025

OS/Environment: Kali Linux

1. Overview

This project implements the full Recipes assessment requirements: a backend REST API and a frontend UI that consumes the API.

What the solution does

1. Parses the provided JSON dataset
2. Cleans/normalizes numeric fields (NaN or invalid → NULL)
3. Loads recipes into PostgreSQL
4. Exposes REST endpoints to list recipes and search recipes with filters
5. Provides a UI table with pagination, field-level filters, and a right-side drawer (details view)

Key Features

Backend

- Data import script with cleaning rules (NaN/invalid → NULL)
- PostgreSQL schema using JSONB for nutrients + indexes for faster queries
- FastAPI endpoints with Swagger documentation (auto-generated)
- Stable sorting for consistent pagination

Frontend

- Table UI with required columns: Title (truncated), Cuisine, Rating (stars), Total Time, Serves
- Field-level filters that call `/api/recipes/search`
- Row click opens a right-side drawer showing Title + Cuisine header, Description, Total Time expandable (prep + cook), and Nutrition table
- Server-side pagination with page size options between 15 and 50
- Fallback messages: “No results found” and “No data found”

2. Repository / Folder Structure

```
recipes_assessment/  
  code/  
    backend/
```

app/ (FastAPI application code)
scripts/ (ingestion scripts)
data/ (dataset)
schema.sql
requirements.txt
.env.example
README.md
frontend/
src/
vite.config.ts
package.json
screenshots/
documentation/

3. Prerequisites

Install system dependencies:

```
sudo apt update
```

```
sudo apt install -y python3 python3-venv python3-pip postgresql postgresql-contrib  
nodejs npm
```

4. Database Setup

4.1 Start PostgreSQL

```
sudo systemctl enable --now postgresql
```

```
sudo systemctl status postgresql --no-pager
```

4.2 Set password for postgres user

```
sudo -u postgres psql
```

Inside psql:

```
ALTER USER postgres WITH PASSWORD 'YourNewPassword!';
```

```
\q
```

Verify login:

```
psql -U postgres -h localhost -c "SELECT 1;"
```

4.3 Create database

```
psql -U postgres -h localhost -c "CREATE DATABASE recipes_db;"
```

5. Database Schema

schema.sql creates the recipes table and indexes.

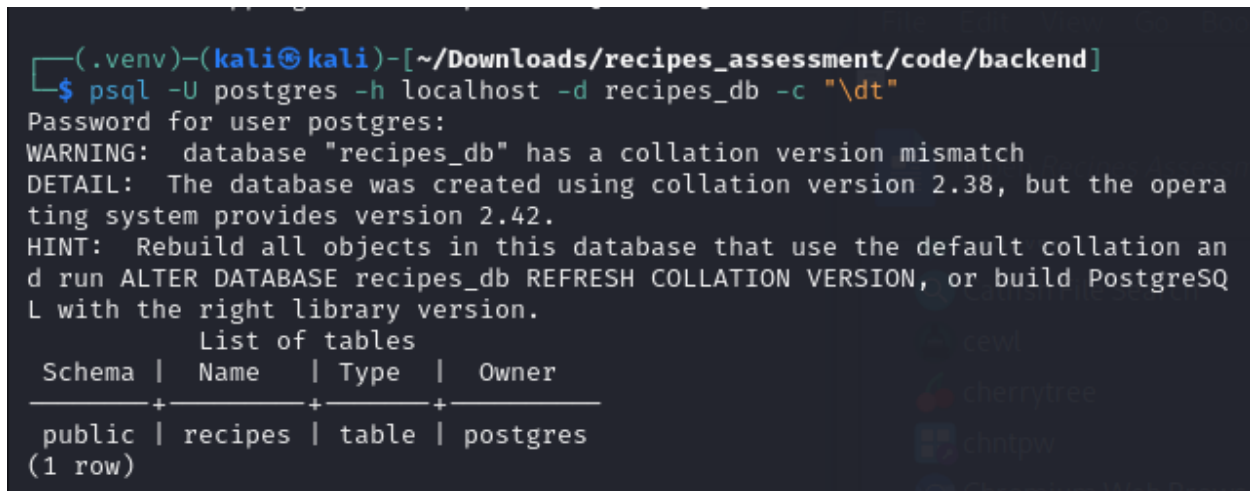
Run schema:

```
cd ~/recipes_assessment/code/backend
```

```
psql -U postgres -h localhost -d recipes_db -f schema.sql
```

Verify table exists:

```
psql -U postgres -h localhost -d recipes_db -c "\dt"
```



```
(.venv)-(kali@kali)-[~/Downloads/recipes_assessment/code/backend]
$ psql -U postgres -h localhost -d recipes_db -c "\dt"
Password for user postgres:
WARNING: database "recipes_db" has a collation version mismatch
DETAIL: The database was created using collation version 2.38, but the operating system provides version 2.42.
HINT: Rebuild all objects in this database that use the default collation and run ALTER DATABASE recipes_db REFRESH COLLATION VERSION, or build PostgreSQL with the right library version.
      List of tables
 Schema | Name   | Type  | Owner
-----+-----+-----+-----
 public | recipes | table | postgres
(1 row)
```

- screenshots/01_psql_dt.png : output of \dt showing recipes table

6. Backend Setup (Python + Dependencies)

Go to backend directory:

```
cd ~/recipes_assessment/code/backend
```

Create virtual environment + activate:

```
python3 -m venv .venv
```

```
source .venv/bin/activate
```

Upgrade pip:

```
pip install --upgrade pip
```

Install packages:

```
pip install -r requirements.txt
```

Quick import test:

```
python -c "import fastapi, uvicorn, sqlalchemy, psycopg; print('OK')"
```

7. Environment Variables

Create .env using .env.example (do not submit .env):

```
cp .env.example .env
```

```
nano .env
```

Example (replace password with your local postgres password):

DATABASE_URL=postgresql+psycopg://postgres:YourNewPassword!@localhost:5432/recipes_db

Note: For submission, only include .env.example (never include .env, since it contains real credentials).

8. Data Import

8.1 Place dataset in the data folder

Copy US_recipes_null.Pdf.json into:

code/backend/data/

Verify:

ls -lh data/

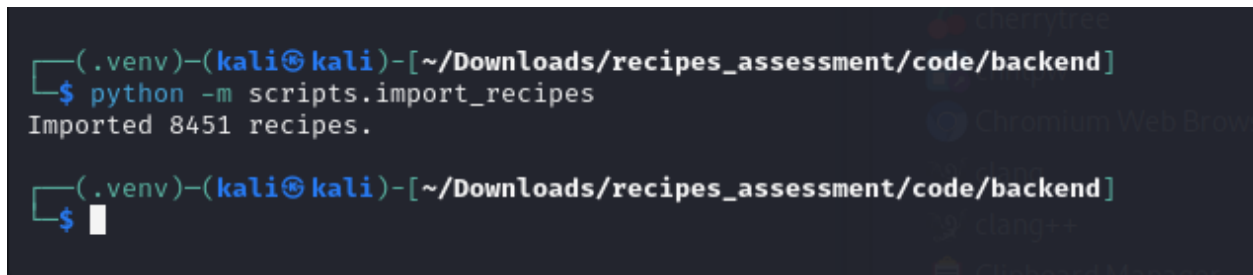
8.2 Run import script

From code/backend:

python -m scripts.import_recipes

Expected output (example):

Imported 8451 recipes.

A terminal window screenshot with a dark background. The prompt is `(.venv)-(kali㉿kali)-[~/Downloads/recipes_assessment/code/backend]`. The user enters `$ python -m scripts.import_recipes` and the output is `Imported 8451 recipes.`. The prompt is shown again below the output.

```
(.venv)-(kali㉿kali)-[~/Downloads/recipes_assessment/code/backend]
$ python -m scripts.import_recipes
Imported 8451 recipes.

(.venv)-(kali㉿kali)-[~/Downloads/recipes_assessment/code/backend]
$
```

- screenshots/02_import_output.png: terminal output showing successful import

8.3 Verify data loaded

psql -U postgres -h localhost -d recipes_db -c "SELECT COUNT(*) FROM recipes;"

```
(.venv)-(kali@kali)-[~/Downloads/recipes_assessment/code/backend]
$ psql -U postgres -h localhost -d recipes_db -c "SELECT COUNT(*) FROM recipes;"

Password for user postgres:
WARNING: database "recipes_db" has a collation version mismatch
DETAIL: The database was created using collation version 2.38, but the operating system provides version 2.42.
HINT: Rebuild all objects in this database that use the default collation and run ALTER DATABASE recipes_db REFRESH COLLATION VERSION, or build PostgreSQL with the right library version.
 count
-----
  8451
(1 row)

(.venv)-(kali@kali)-[~/Downloads/recipes_assessment/code/backend]
$
```

- screenshots/03_count_rows.png: count(*) result in terminal

9. Run the Backend API

Start the server (from code/backend with venv active):

```
uvicorn app.main:app --reload --port 8000
```

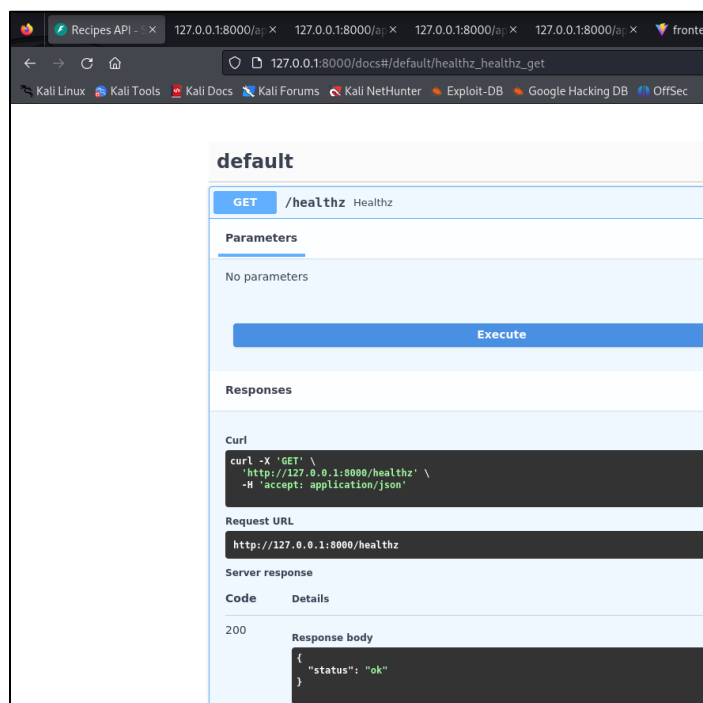
If you see “Address already in use” (port 8000 busy), free the port:

```
sudo lsof -i :8000
```

```
sudo kill -9
```

Swagger docs (open in browser):

<http://127.0.0.1:8000/docs>

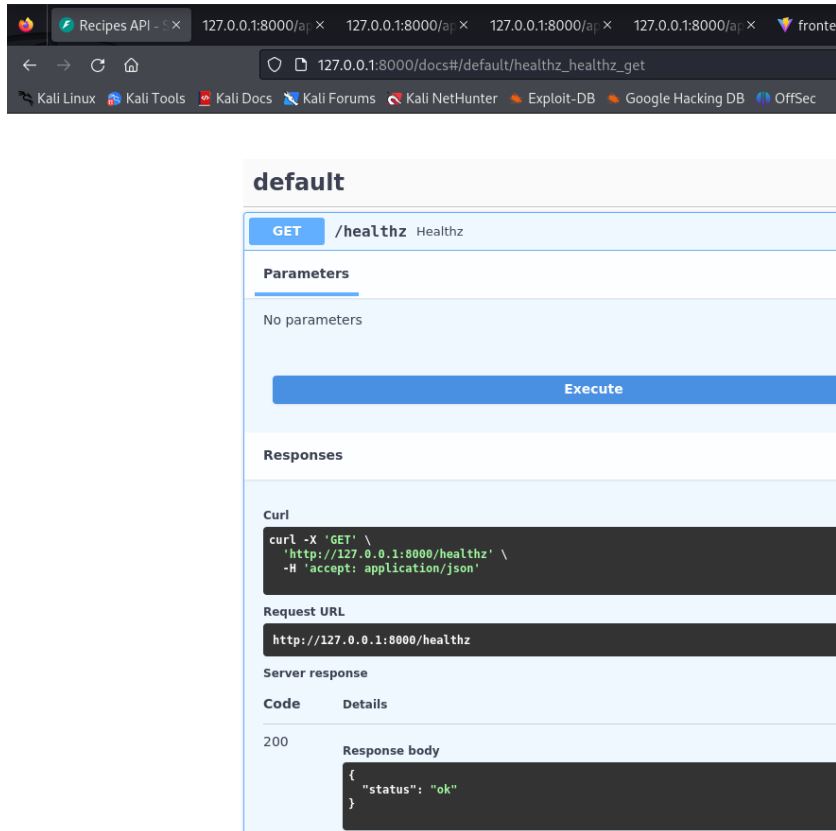


- screenshots/04_swagger_docs.png : Swagger UI loaded in browser

10. API Testing (Requests + Sample Responses)

10.1 Health check

curl "<http://127.0.0.1:8000/healthz>"



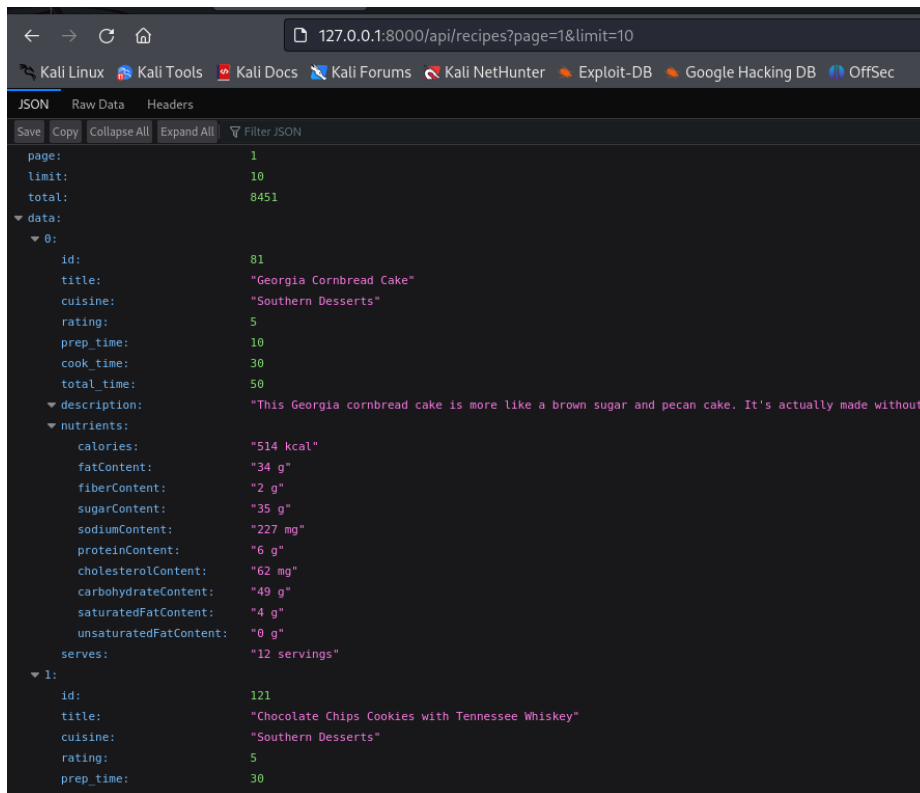
- screenshots/05_healthz.png

10.2 List recipes endpoint

Endpoint: GET /api/recipes?page=1&limit=15

Example:

curl "<http://127.0.0.1:8000/api/recipes?page=1&limit=15>"

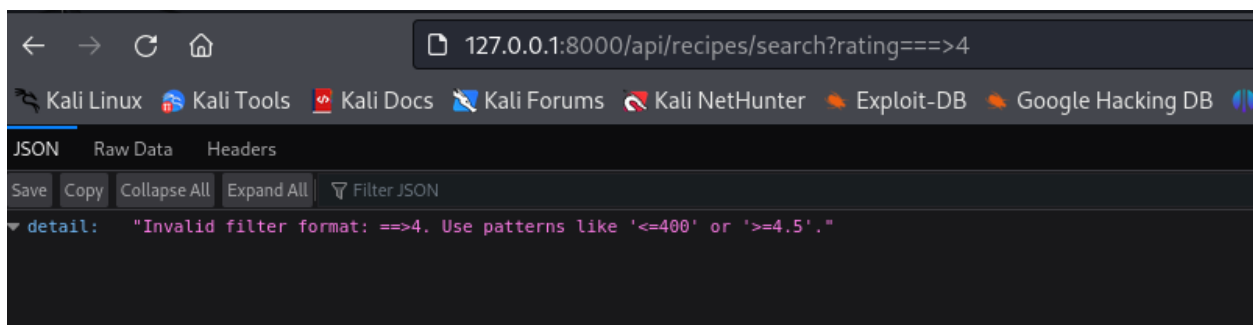


- screenshots/06_api_recipes.png

10.3 Search endpoint

Invalid filter format (returns HTTP 400)

curl "<http://127.0.0.1:8000/api/recipes/search?rating==>4>"



- screenshots/08_api_invalid_filter_400.png

11. Frontend Setup (React UI)

11.1 Install frontend dependencies

`cd ~/recipes_assessment/code/frontend`

`npm install`

11.2 Configure API proxy (to avoid CORS issues)

In code/frontend/vite.config.ts, ensure the proxy points to the backend:

Proxy /api → <http://localhost:8000>

Proxy /healthz → <http://localhost:8000>

11.3 Run the frontend

npm run dev

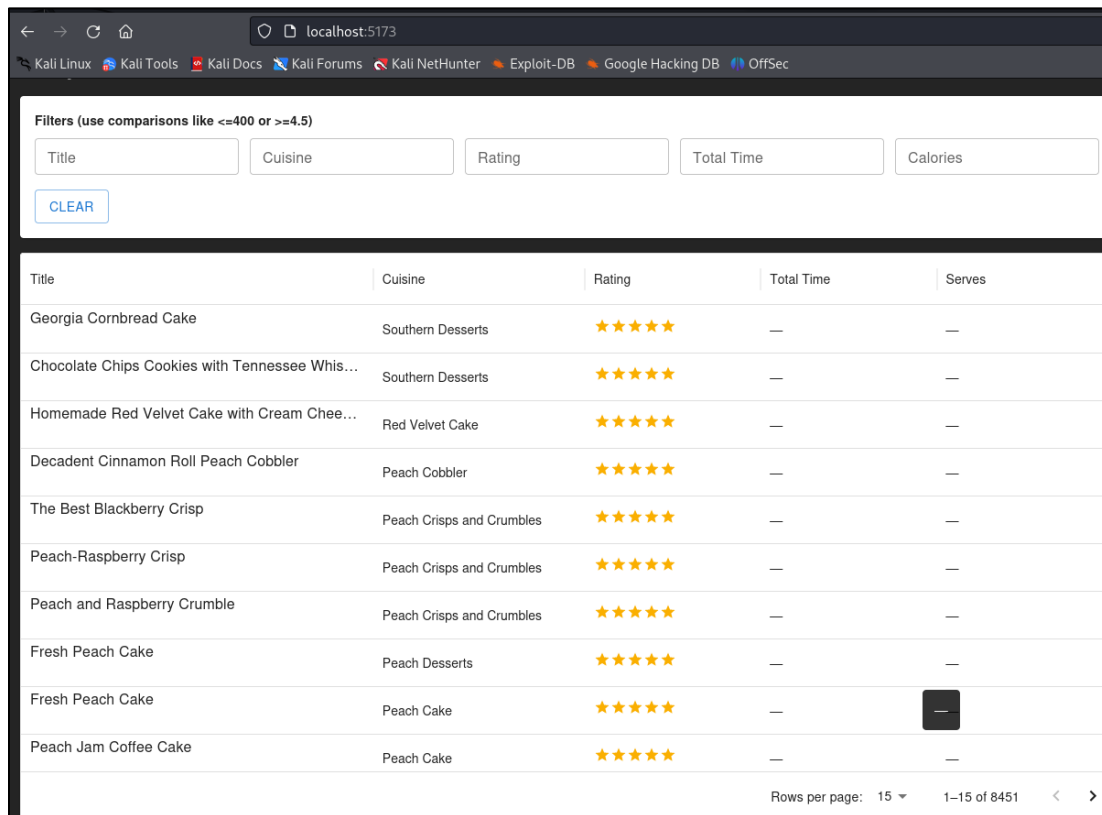
Open UI in browser:

<http://127.0.0.1:5173>

12. UI Validation Checklist (Required Screenshots)

12.1 Table loaded with required columns

Confirm the table displays: Title (truncated), Cuisine, Rating (stars), Total Time, Serves.



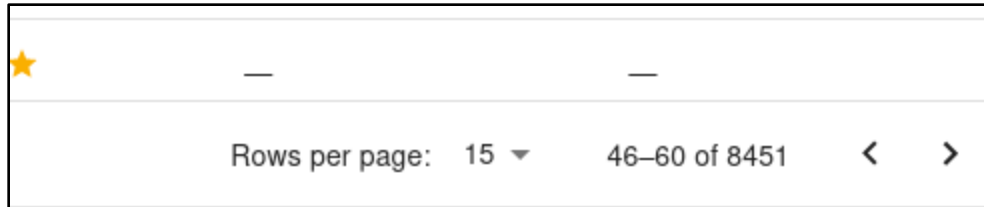
| Title | Cuisine | Rating | Total Time | Serves |
|--|---------------------------|--------|------------|--------|
| Georgia Cornbread Cake | Southern Desserts | ★★★★★ | — | — |
| Chocolate Chips Cookies with Tennessee Whis... | Southern Desserts | ★★★★★ | — | — |
| Homemade Red Velvet Cake with Cream Chee... | Red Velvet Cake | ★★★★★ | — | — |
| Decadent Cinnamon Roll Peach Cobbler | Peach Cobbler | ★★★★★ | — | — |
| The Best Blackberry Crisp | Peach Crisps and Crumbles | ★★★★★ | — | — |
| Peach-Raspberry Crisp | Peach Crisps and Crumbles | ★★★★★ | — | — |
| Peach and Raspberry Crumble | Peach Crisps and Crumbles | ★★★★★ | — | — |
| Fresh Peach Cake | Peach Desserts | ★★★★★ | — | — |
| Fresh Peach Cake | Peach Cake | ★★★★★ | — | — |
| Peach Jam Coffee Cake | Peach Cake | ★★★★★ | — | — |

Rows per page: 15 1-15 of 8451

- screenshots/09_ui_table.png: table loaded with columns visible

12.2 Pagination controls + page size 16-30

Change results per page and confirm options include values between 16 and 30.



- screenshots/10_ui_pagination_pagesize.png: pagination UI showing page size options

12.3 Field-level filters calling /api/recipes/search

Enter filters such as: Title = “pie”, Rating = “>=4.5”, Calories = “<=400”, and confirm results update.

 A screenshot of a filter UI. At the top, it says 'Filters (use comparisons like <=400 or >=4.5)'. Below this are five input fields: 'Title' with 'pie', 'Cuisine' (empty), 'Rating' with '>=4.5', 'Total Time' (empty), and 'Calories' with '<=400'. A 'CLEAR' button is to the left of the 'Calories' field. Below the filters is a table with the following data:

| Title | Cuisine | Rating | Total Time | Serves |
|---------------------------------------|------------------|--------|------------|--------|
| Peach Crumble Pie | Peach Pie | ★★★★★ | — | — |
| Spiced Peach Pie | Peach Pie | ★★★★★ | — | — |
| Banana Coconut Pudding or Pie Filling | Banana Pudding | ★★★★★ | — | — |
| Mimi's Southern Sweet Potato Pie | Sweet Potato Pie | ★★★★★ | — | — |
| Healthier Sweet Potato Pie I | | ★★★★★ | | |

- screenshots/11_ui_filters_applied.png: filters visible + results updated

12.4 Row click opens right-side drawer with details

Click a row and confirm the drawer displays:

- Header: Title + Cuisine
- Description section
- Total Time with expand icon; expanded shows Prep Time and Cook Time
- Nutrition table with: calories, carbohydrateContent, cholesterolContent, fiberContent, proteinContent, saturatedFatContent, sodiumContent, sugarContent, fatContent
- Serves

Peach Crumble Pie

Peach Pie

Description

Fresh fruit and a spiced, sweet topping are the stars in this easy, baked peach crumble pie that's perfect for family gatherings.

Total Time65 min

Nutrition

| | |
|---------------------|----------|
| calories | 285 kcal |
| carbohydrateContent | 40 g |
| cholesterolContent | 15 mg |
| fiberContent | 1 g |
| proteinContent | 2 g |
| saturatedFatContent | 6 g |
| sodiumContent | 142 mg |
| sugarContent | 26 g |
| fatContent | 13 g |

Serves
8 servings

- screenshots/12_ui_drawer_open.png — drawer open with header + description

Spiced Peach Pie

Peach Pie

Description

Baked in a homemade crust with tons of warm and comforting flavor, this spiced peach pie is perfect for impressing guests.

Total Time205 min

Prep Time: 40 min
Cook Time: 45 min

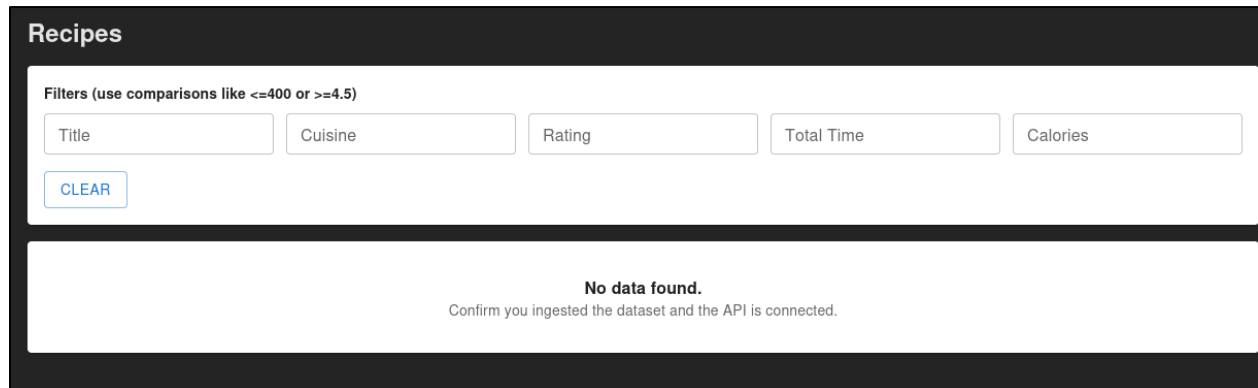
Nutrition

| | |
|---------------------|----------|
| calories | 349 kcal |
| carbohydrateContent | 50 g |
| cholesterolContent | 32 mg |
| fiberContent | 2 g |

- screenshots/13_ui_drawer_expanded_time.png: total time expanded showing prep + cook

12.5 No data fallback message (nice-to-have)

Stop the backend and refresh the UI (or temporarily point proxy to wrong port) and confirm “No data found.”



The screenshot shows a web application titled "Recipes" with a dark header. Below the header is a filter section with the text "Filters (use comparisons like <=400 or >=4.5)". There are five input fields: "Title", "Cuisine", "Rating", "Total Time", and "Calories". A "CLEAR" button is located below the "Title" field. Below the filter section is a large white box containing the message "No data found." in bold, followed by the text "Confirm you ingested the dataset and the API is connected." in a smaller font.

- screenshots/14_ui_no_data.png

Appendix: Code Reference

Backend entrypoint: `code/backend/app/main.py`

Ingestion script: `code/backend/scripts/import_recipes.py`

Frontend source: `code/frontend/src/` (components + API client)