Meal Planner Project

Purpose:

This project is a calorie-aware fridge clean-out planner. The core idea is: you upload a few overview photos of your fridge, pantry, or counter, tell the system roughly how many calories you want per person and for which meal (alongside some constraints), and it turns that into a structured inventory, nutrition estimates, and a concrete recipe with an accompanying “what it might look like” dish image. Rather than trying to magically guess exact calories from pixels, the photos are used primarily for inventory discovery: identifying what ingredients you have, in what form, and in what rough quantities. The actual calories are then computed from those structured assumptions using a RAG pipeline over calorie tables (e.g., calories.info + USDA-style references), not from the image itself.

The tool is meant for people like me and my girlfriend, who are trying to lose weight but also hate wasting food. We’re already fairly comfortable cooking from recipes, and we care a lot about hitting approximate calorie targets without spending twenty minutes hunting through Google or calorie-specific blogs for something that (a) fits our macros and (b) uses what’s already in the fridge. Off-the-shelf ChatGPT can generate recipes, but it doesn’t really know what’s in our actual fridge and it doesn’t systematically prioritize perishables or leftovers. Even if we prompt it to do so, it is insufficient to perform accurate ingredient extraction, RAG-based verified calorie estimation, recipe generation, and image generation, with well-structured, trackable databases enabling user tweaks, within a single query/session. This system’s second explicit objective is “use up ingredients that are likely to spoil soon,” so the recommended meals are not just calorie-aware but also frugal, waste-minimizing, and inventory-tracking.

Under the hood, the project is implemented as an agentic workflow rather than a single prompt. A vision step first scans the photos to propose ingredient rows (name, form, weight estimate, spoilage horizon), which get stored in a SQLite pantry table. A calorie agent then uses RAG over nutrition websites to assign calories and macros to each ingredient, computing total calories based on the inventory amounts. A meal-planning agent reads both tables plus user constraints and designs a recipe that hits the target calories as closely as the available inventory allows, while explicitly favoring the constraints. Finally, an image-generation agent produces a photorealistic dish image, and everything (ingredients, nutrition, recipe, and image metadata) is kept in a trackable database so the UI can show consistent “recipe cards” and, in the future, accept tweaks (“make this higher protein,” “assume only 100g broccoli,” “give me another dish using the same peppers”).

So the contribution is not “ChatGPT, but with a picture.” It’s a coordinated pipeline that combines vision-based inventory discovery, RAG-grounded calorie estimation, perishable&constraints-aware optimization, and structured storage into one flow. The design is deliberately modular: each agent is responsible for a well-defined table and can be re-invoked when something changes, which is something a one-shot ChatGPT conversation cannot reliably do.

Accomplishments:

- Built a web app, not just a notebook
    - Built a Streamlit UI that lets users upload fridge images, set meal + calorie targets, run the full pipeline, and view tables + recipe cards.
- Search-augmented retrieval for calories
    - Integrated a RAG tool over curated websource pages and used it to populate a structured nutrition table.
- Multiple models working together
    - Combined gpt-5.1 (reasoning + vision), gpt-5-mini (utility tasks), and gpt-image-1-mini (dish images) in a coordinated workflow.
- Multi-modal: images in, images out
    - Used fridge photos to detect ingredients via vision, and generated plausible dish images tied to recipes, displayed correctly in the UI from an in-memory image store.
- Multiple APIs / data sources
    - Orchestrated OpenAI chat, OpenAI images, RAG over web calorie tables, and a local SQLite DB for persistent state.
- Multi-stage, multi-agent workflow
    - Implemented a CrewAI pipeline with distinct agents (ingredients, calories, meals, images, orchestrator) passing work through a shared database.
- Custom, domain-specific tools
    - Created a set of specialized tools (DB upsert/fetch, RAG wrapper, typed image_generation, session state) instead of relying only on generic tools.
- Calorie-aware fridge cleanout
    - Helps users turn what they actually have into recipes that target a given kcal/person while preferentially using soon-to-spoil items.
- “Use what you have” optimization
    - Recipes are explicitly designed around the existing inventory and constraints, not generic ingredients, making this more practical than one-shot ChatGPT prompts.
- Extensible for interactive tweaks
    - The architecture and tools already support ingredient, nutrition, recipe, and image feedback flows; only a chat/feedback UI layer remains to be wired up.

**Lessons Learned:*

- Passing image bytes/base64 directly in LLM prompts explodes token counts; using short handles + an image store and only encoding at vision/image tool boundaries is the safe pattern.
- Agents wouldn’t reliably call the provided tools; I solved this by doing manual Python calls (vision/image → DB), then letting agents operate on DB state instead.
- Querying the calorie RAG once per ingredient was slow; switching to a single composite query (batched) returning a JSON array of nutrition rows is faster and cleaner.

Agentic Workflow Design

Tool Stack

CrewAI: runs the multi-agent workflow using OpenAICompletion for all LLM calls.
OpenAI API: gpt-5.1 / gpt-5-mini for text + vision; gpt-image-1(-mini) for dish image generation.
Streamlit: simple one-shot UI:

- User uploads fridge/pantry images and sets: hints, meal time, target kcal/person, # of people, constraints.
- User clicks “Run full workflow”, which fires the entire pipeline once.
- Final UI shows:
    - A recipe card with an image.
    - Raw database tables (ingredients, nutrition, recipes, recipe_ingredients, images).

SQLite: structured state: ingredients, nutrition, recipes, recipe_ingredients, images, and session_state.
In-memory image store (temp; better solution is a cloud storage):

- Holds *input* fridge images and *generated* recipe images as bytes.
- The LLM only ever sees small handles like img_0 (input) or gen://… (generated); base64 bytes never go into the model context, else token overflow will definitely occur.

High-Level Flow

User submits one shot:

- Uploads one or more images.
- Provides optional hints + meal time + kcal target + # people + constraints.
- Presses <Run full workflow>.

Phase 0 – Vision extraction (manual in Python, outside CrewAI)

- run_primary_flow calls vision_detect_ingredients (another LLM flow) directly with handles for the uploaded images.
- vision model returns JSON { detected: [...], summary: ... }.
- Python writes the detected rows into the ingredients table via upsert_ingredients, and stores the text summary in session_state (ingredients_summary).

Phase 1 – Ingredients summary (CrewAI Task 1)

- Ingredients Extractor agent (now *read-only*):
    - Reads ingredients (and optionally ingredients_summary) from the DB.
    - Returns a JSON payload with:
        - ingredients: current ingredient rows (no new vision calls).
        - summary: human-readable description of what’s in the fridge.

Phase 2 – Calories from RAG (CrewAI Task 2)

- Calorie Provider agent:
    - Reads all ingredients.
    - Builds one composite query string listing each ingredient id, name, and total grams.
    - Calls calorie_rag_lookup once (CrewAI RagTool over many calories.info/... pages).
    - Parses the JSON array of nutrition objects.
    - Computes total_calories_for_inventory_amount using the inventory weight and kcal/100g.
    - Persists rows into the nutrition table via db_upsert_nutrition (either directly from the agent or in host code, depending on the run).

Phase 3 – Meal planning (CrewAI Task 3 + small host glue)

- Meal Planner agent:
    - Reads ingredients + nutrition.
    - Uses the user's meal_time, target_calories_per_person, num_people, and constraints.
    - Designs one recipe for now: (could change it to any number in the prompt; but one to save costs)
        - Favors ingredients that are more perishable (shorter spoiling_estimate_days_in_place).
        - Targets target_kcal * num_people total calories (but may under-shoot if only low-calorie items are available).
    - Returns a JSON object:
        - recipes: [{ recipe_id, name, meal_time, servings, per_person_calories, total_calories, ingredient_usages, instructions, dietary_tags }]
        - summary: explanation of calorie fit and spoilage strategy.
- Host code extracts the recipe rows and writes them to recipes and recipe_ingredients via upsert_recipes. This function:
    - Normalizes recipe_id (treats non-int ids like "new_breakfast_001" as new rows).
    - Inserts/updates the recipe.
    - Writes each {ingredient_id, amount_g_or_ml_per_serving} into recipe_ingredients.
    - Recomputes per-person and total calories from recipe_ingredients × nutrition.

Phase 4 – Dish image generation (CrewAI Task 4)

- Image Generator agent:
    - Reads the new recipe(s) from recipes / recipe_ingredients.
    - Calls image_generation once per recipe with:
        - recipe_id (int or string).
        - name.
        - key_ingredients (list of ingredient names).
        - style_notes (composition, lighting, mood).
        - Optional prompt_override and generation_parameters.

- image_generation:
    - Builds a full natural-language prompt if prompt_override is empty.
    - Calls openai_client.images.generate.
    - If the service returns a URL, uses it directly.
    - If it returns b64_json, decodes it to bytes and stores in IMAGE_STORE with key like gen_1001_Sweet_Pepper_&_Green_Apple_Breakfast_Hash_1, returning a handle gen://gen_1001_Sweet_Pepper_&_Green_Apple_Breakfast_Hash_1.
    - On failure or when openai_client is unavailable, returns a stub stub://... URL with error info.
- Agent (or host glue) calls db_upsert_images with rows:
    - { recipe_id, image_url_or_blob_ref, style_notes, generation_parameters }, storing the gen://... handle in the images table.

Phase 5 – Compose recipe cards (CrewAI Task 5)

- Orchestrator agent (currently narrow role):
    - Reads recipes, recipe_ingredients, images, and nutrition via db_fetch_tables.
    - For now, only combines one recipe and its best image.

Returns a single JSON object with schema:

```
{
  "recipe_cards": [
    {
      "recipe_id": ...,
      "name": "...",
      "meal_time": "...",
      "servings": 2,
      "per_person_calories": 650,
      "total_calories": 1300,
      "key_ingredients": [...],
      "uses_soon_to_spoil": [...],
      "image_url_or_blob_ref": "gen://... or stub://... or http(s)://...",
      "short_description": "..."
    }
  ],
  "summary": "..."
}
```

Phase 6 – UI rendering

- app.py pulls ComposeRecipesAndImagesForDisplay's output from task_outputs, decodes the JSON, and displays:
    - Title + short description.
    - Image:
        - stub://... → placeholder.

        - gen://... → look up bytes in IMAGE_STORE and show.
        - http(s):// or local path → render directly.
- Below, the Streamlit app displays full DB tables (ingredients, nutrition, recipes, recipe_ingredients, images) as dataframes for inspection.

Agents

1. Ingredients Extractor
Role: Summarizes what the vision pipeline (LLM query to extract ingredients) already stored in the ingredients table. Does not call the vision tool itself.
Goal: Produce a JSON payload: { "ingredients": [...current rows...], "summary": "short inventory summary" } for the UI and downstream agents.
Backstory: You are a meticulous kitchen inventory auditor, but by the time you're called, the "junior" vision system has already logged ingredients into the database. You double-check what's there, optionally tidy naming in-memory, and summarize it for the user and for other agents. You currently operate read-only in the main flow, but separate feedback entry points (run_ingredient_feedback) exist for future tweaks.
Delegation/Tools:

- Can call: db_fetch_tables, session_get_state.
- Not currently delegating to other agents (orchestrator logic is handled in Python).

2. Calorie Provider
Role: Map each ingredient to nutrition info and keep the nutrition table in sync.
Goal: Fill nutrition with:

```
 {
  "ingredient_id": int,
  "calories_per_100g_or_serving": float,
  "standard_serving_size": "string",
  "macro_breakdown": { "protein_g": float, "fat_g": float, "carbs_g": float },
  "total_calories_for_inventory_amount": float,
  "source": "calories.info (RAG) / USDA / estimated"
}
```

Backstory: Spreadsheet-minded nutrition analyst. Uses a RAG tool over Calories.info (and inferred USDA values) to get kcal/100g and macros, then computes totals based on inventory weight.
Delegation/Tools:

- db_fetch_tables → read ingredients.
- calorie_rag_lookup → a **single** composite query over all ingredients.
- db_upsert_nutrition → persist rows.
- Feedback path (run_nutrition_feedback) exists but is not yet wired into the Streamlit UI.

3. Meal Planner

Role: Design one recipe that uses available ingredients, targets the requested calories per person and number of people, and prioritizes soon-to-spoil items.
Goal: Produce one recipe row in recipes plus its recipe_ingredients links.
Backstory: Pragmatic home-chef who looks at the fridge inventory (ingredients) and calorie spreadsheet (nutrition) and builds a single good option, rather than a menu of three. You'll favor ingredients with shorter shelf life and try to hit calorie targets while acknowledging when the inventory is too low-calorie to reach them exactly.
Delegation/Tools:

- db_fetch_tables → read ingredients + nutrition.
- Conceptual db_upsert_recipes (goal state); in practice, run_primary_flow currently parses the agent output and calls upsert_recipes itself.
- Feedback task (run_recipe_feedback) is defined but not wired into the UI yet.

4. Image Generator
Role: Generate a plausible food photo for each recipe and store image_url_or_blob_ref, style_notes, and generation_parameters
Goal: Maintain images table with reproducible metadata and gen:// handles for locally stored bytes.
Backstory: You're a food stylist powered by gpt-image-1. You read the recipe (name + key ingredients + style notes) and either call the real image model and stream the result into IMAGE_STORE as bytes, returning a gen://… handle, or fall back to a stub with an error note if the image model is unavailable or blocked.
Delegation/Tools:

- image_generation → our typed tool (Image model query to generate dish images) that talks to OpenAI Images and writes to IMAGE_STORE.
- db_upsert_images → persist metadata for each recipe.

5. Orchestrator / Coordinator
Role: Assemble recipe cards (ComposeRecipesAndImagesForDisplay). **The actual orchestration of "ingredients → calories → meal → image → cards" is done in Python in run_primary_flow.*
Goal: Given DB state, output:

```
{
  "recipe_cards": [ { ... card fields ... } ],
  "summary": "brief comparison"
}
```

that Streamlit can display without further processing.
Backstory: You're the head chef at plating time. The sous-chefs (ingredients, calories, meals, images) have already done their work. You simply read the DB, pick the relevant recipe and image, and format a neat "recipe card" for the UI.
Delegation/Tools:

- db_fetch_tables → to see recipes, recipe_ingredients, images.
- No delegation to other agents; in this codebase, you're the final step in the agent chain.

Tasks (performed when the one-shot button flow is triggered)

(Phase 0; not a Task) **vision_detect_ingredients (Python)**

- Direct vision call; writes ingredients + ingredients_summary.

Task 1 – ExtractIngredientsFromImages (Ingredients agent)

- Reads ingredients + ingredients_summary.
- Returns {"ingredients": [...], "summary": "..."}.

Task 2 – FetchCaloriesForIngredients (Calorie Provider)

- Reads ingredients.
- Single composite calorie_rag_lookup query.
- Writes nutrition.

Task 3 – PlanMeals (Meal Planner)

- Reads ingredients + nutrition.
- Builds one recipe JSON.
- Host code parses and calls upsert_recipes.

Task 4 – GenerateRecipeImages (Image Generator)

- Reads recipe(s).
- Calls image_generation to produce an image handle.
- Writes images row(s).

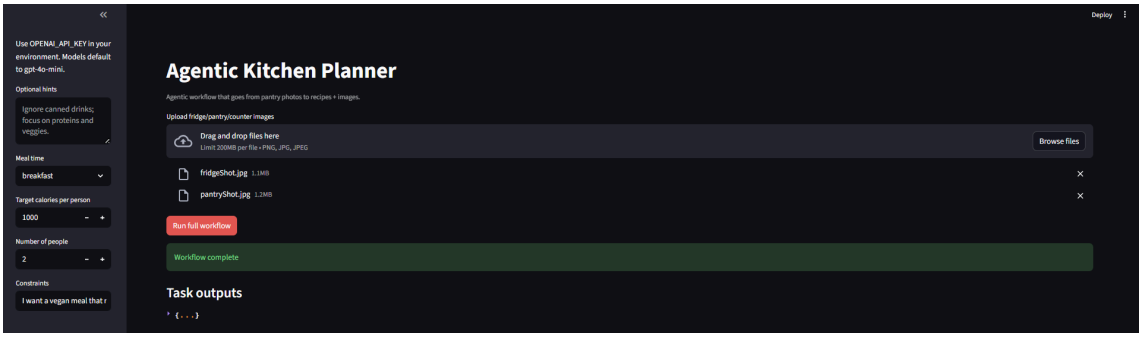
Task 5 – ComposeRecipesAndImagesForDisplay (Orchestrator)

- Reads recipes + recipe_ingredients + images.
- Returns {"recipe_cards": [...], "summary": "..."}.

Defined but *not used* in the current Streamlit UI (future work):
UpdateIngredientsFromUserFeedback (Ingredients agent + db_apply_ingredient_patches)
UpdateCaloriesFromUserFeedback (Calorie agent + db_apply_nutrition_patches)
RefineRecipeFromUserFeedback (Meal Planner + db_apply_recipe_patches)
RegenerateOrVaryImage (Image Generator + db_upsert_images)
**They give hooks to later add a chatbox / tweak-flow without redesigning the DB or tools; just need a new UI entry point that calls the appropriate KitchenCrew.run_*_feedback method instead of run_primary_flow.*

Demo (Complex version, as mentioned in the video)

- **User Input** (pantryShot.jpg, fridgeShot.jpg, and query constraints/parameters + UI)

- 
- 
- 

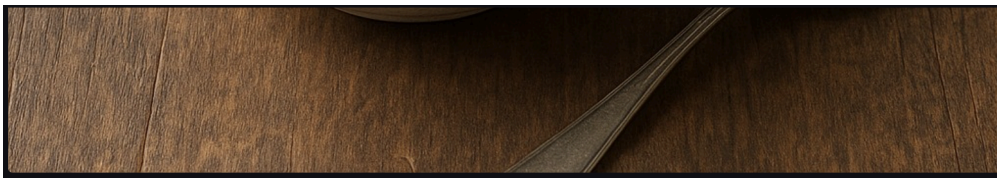
    - Zoom in to see.
    - Constraints: ”I want a vegan meal that reaches the target calories per person”
- **App Output** (generated recipe card [recipe + summary + image] and trackable DB tables for each step) *(ZOOM IN to see)*

Recipe cards

High-Calorie Vegan Rice & Berry Breakfast Bowls with Sweet Apple Juice

Warm sweetened white rice bowls topped with fresh berries and brightened with lime, served with a tall glass of chilled apple juice for a high-calorie vegan breakfast.

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `width` parameter instead.



High-Calorie Vegan Rice & Berry Breakfast Bowls with Sweet Apple Juice

Current tables

ingredients

	ingredient_id	name	form	estimated_weight_g_or_ml	observed_quantity	spoiling_estimate_days_in_place	confidence	created_at
0	1	bottled cooking oils (likely olive/vegetable oils)	bottles on left countertop	2500.0	about 4 medium bottles	120.0	0.86	2025-12-09 05:13:32
1	2	bottled vinegars (e.g., rice vinegar, other clear vinegars)	bottles on left countertop	1500.0	about 3 bottles	365.0	0.8	2025-12-09 05:13:32
2	3	soy sauce or similar dark sauces	bottles on left countertop	1000.0	about 2 bottles	365.0	0.85	2025-12-09 05:13:32
3	4	granulated white sugar	in small plastic container	200.0	1 small container	365.0	0.95	2025-12-09 05:13:32
4	5	uncooked white rice	in small plastic container	200.0	1 small container	365.0	0.9	2025-12-09 05:13:32
5	6	assorted dried spices and herbs	multiple small spice jars on right countertop rack and beside it	600.0	about 18–20 jars	365.0	0.9	2025-12-09 05:13:32
6	7	ground black pepper	tall grinder bottle on right countertop	100.0	1 bottle	365.0	0.95	2025-12-09 05:13:32
7	8	table salt	tall cylindrical container on right countertop	700.0	1 container	365.0	0.96	2025-12-09 05:13:32
8	9	bottled hot sauce / chili sauce	bottles near spice rack and in fridge door	700.0	about 3 bottles	180.0	0.8	2025-12-09 05:13:32
9	10	whipped cream	aerosol can in fridge	400.0	1 can	30.0	0.96	2025-12-09 05:13:32

- **nutrition**

	ingredient_id	calories_per_100g_or_serving	standard_serving_size	macro_protein_g	macro_fat_g	macro_carbs_g	total_calories_for_inventory_amount	source	updated_at
16	17	43.0	100 g (beer)	0.4	0.0	3.6	430.0	estimated (beer ~40-45 kcal/100ml; beverage labels/USDA)	2025-12-09 05:20:39
17	18	0.0	100 g (diet soda)	0.0	0.0	0.0	0.0	known (Diet sodas are typically non-caloric; product nutrition panels)	2025-12-09 05:20:39
18	19	40.0	100 g (soft drink - estimated	0.0	0.0	10.0	142.0	estimated (could be seltzer 0 kcal or sugar-sweetened soda ~40 kcal/100g; marked estim	2025-12-09 05:20:39
19	20	150.0	100 g	20.0	8.0	2.0	600.0	estimated (packaged sliced deli meat average ~120-170 kcal/100g; USDA/manufacturer a	2025-12-09 05:20:39
20	21	174.0	100 g	7.0	13.0	3.0	783.0	estimated (ricotta/soft cheese ~170-180 kcal/100g; USDA)	2025-12-09 05:20:39
21	22	431.0	100 g	38.0	29.0	4.0	862.0	estimated (parmesan / hard grated cheese ~431 kcal/100g; USDA)	2025-12-09 05:20:39
22	23	60.0	100 g (milk, whole-milk app	3.2	3.3	4.8	2268.0	estimated (milk ~60 kcal/100g whole milk; USDA)	2025-12-09 05:20:39
23	24	155.0	100 g (whole egg)	13.0	11.0	1.1	465.0	estimated (whole eggs ~155 kcal/100g; USDA)	2025-12-09 05:20:39
24	25	23.0	100 g (green bell pepper)	0.9	0.2	4.6	92.0	estimated (green bell pepper ~23 kcal/100g; calories.info / USDA)	2025-12-09 05:20:39
25	26	25.0	100 g (mixed fresh vegetabl	2.0	0.3	5.0	625.0	estimated (mixed vegetables average ~20-30 kcal/100g; USDA / generic averages)	2025-12-09 05:20:39

recipes

	recipe_id	name	meal_time	servings	per_person_calories	total_calories	instructions	dietary_tags
0	1	High-Calorie Vegan Rice & Berry Breakfast Bowls with Sweet Apple Juice	breakfast	2	895.725	1791.45	["Rinse the white rice under cold water until the water runs mostly clear:", "In a medium	vegan,dairy-free,egg-free,high-calorie,nut-free,vegetarian

recipe_ingredients

	recipe_id	ingredient_id	amount_g_or_ml_per_serving
0	1	5	125.0
1	1	1	15.0
2	1	4	25.0
3	1	11	62.5
4	1	12	100.0
5	1	31	300.0
6	1	6	1.0
7	1	27	5.0

- **recipes**

	recipe_id	name	meal_time	servings	per_person_calories	total_calories	instructions	dietary_tags
0	1	High-Calorie Vegan Rice & Berry Breakfast Bowls with Sweet Apple Juice	breakfast	2	895.725	1791.45	["Rinse the white rice under cold water until the water runs mostly clear:", "In a medium pot, combine the total rice for 2 servings (about 250 g) with about 2.5\u20133 times as much water by volume. Bring to a boil, then reduce to low heat, cover, and simmer until the rice is tender and the water is absorbed (about 15\u201320 minutes). Turn off the heat and let sit covered for 5 minutes.", "While the rice cooks, rinse the fresh blueberries and strawberries. Hull and slice the strawberries into bite-size pieces.", "Fluff the cooked rice with a fork. While it is still hot, stir in the total cooking oil (about 30 g for 2 servings) and sugar (about 50 g for 2 servings). Mix well so the oil and sugar coat the grains, creating a rich, lightly sweet base.", "Add a small pinch of mixed dried spices (such as cinnamon-like spices if available, or a mild warm spice) and a tiny pinch of salt to the rice to enhance flavor. If desired, squeeze a small amount of lime juice (about 10 g total) over the rice for brightness and stir again.", "Divide the sweet, enriched rice into two bowls.", "Top each bowl with half of the prepared blueberries and strawberries (about 62.5 g blueberries and 100 g strawberries per person). You can gently fold some of the fruit into the warm rice or leave it on top.", "Pour chilled apple juice (about 300 ml per person) into glasses to serve alongside the bowls for additional calories and hydration.", "Taste and adjust sweetness by sprinkling a little extra sugar on top if needed, keeping in mind this will increase calories slightly."]	vegan,dairy-free,egg-free,high-calorie,nut-free,vegetarian

recipe_ingredients

	recipe_id	ingredient_id	amount_g_or_ml_per_serving
0	1	5	125.0
1	1	1	15.0
2	1	4	25.0
3	1	11	62.5
4	1	12	100.0
5	1	31	300.0
6	1	6	1.0
7	1	27	5.0

images

	image_id	recipe_id	image_url_or_blob_ref	style_notes	generation_parameters
0	1	new_breakfast_vegan_rice_berry_bowls_001	gen://gen_new_breakfast_vegan_rice_berry_bowls_001_High-Calorie_Vegan_Rice_I	Morning breakfast scene in warm natural window light; two ceramic bowls arranged (	{"model": "gpt-image-1-mini", "size": "1024x1024", "prompt": "High-quality foo

More Demo and Walkthrough are in the video:

https://youtu.be/tEJHou3QJIE