Vision statement

A local food AI that provides full nutritional value information on any food and can generate complete menu proposals based on the user's specification.

The user can:
* Create an account and log in.
* Get complete nutritional value information on any food, including macro nutrients, minerals, vitamins, amino acids etc.
* Get the full nutritional value information for a given food combination, i.e. the user can enter the quantities of several foods and get a full nutritional value overview.
* Search for specific nutrient content and get a sortable list of all foods that contain this/these nutrient(s).
* Store food combinations in named and editable lists.
* Get menu proposals based on nutritional value and other food constraint input, e.g. allergies.
* Freely chat about anything related to nutrition and get competent answers.
* The AI must have a local web search tool that it will use to anonymously gather any info that it does not have in its local database.
* No user data leaves the server.
* The whole project must be in a public listed https://git.btshub.lu repo named LocalFoodAI_<your IAM> that is updated in real-time. Make sure to prevent confidential data from being uploaded. Add your teacher (id evegi144) as a collaborator. Anyone should be able to clone the repo and get his own local food AI running with ease.
* The application must be designed to run entirely on your provided Ubuntu 24.04 VM (8 vCPUs, 30 GB RAM, no dedicated GPU). You must evaluate and select appropriate lightweight, quantized local LLMs (via Ollama) and databases that fit securely within this hardware limit.

Roles

- You are the product owner and tech lead, whilst Antigravity is your development team. This means that you interview the customer to extract the vision, and then write the Taiga backlog.
- Your teacher is the customer and Scrum master. Add your teacher (gilles.everling@education.lu) as a stakeholder to your Taiga project.

Workflow

- You follow the Scrum manual.
- You agree on the meaning of "Done" with your teacher.
- The sprint time box is one week.

- All meetings are documented in the Wiki.
- In the Daily Scrum meeting you answer and document in the Wiki the three questions:
    - What has been accomplished since the last meeting?
    - What will be done before the next meeting?
    - What obstacles are in the way?
- Thursday:
    - Sprint Review meeting that lasts at most 15 minutes.
    - Sprint Retrospective meeting that lasts at most 15 minutes.
    - Sprint Planning meeting that lasts at most 30 minutes. You estimate the capacity of work, produce the Sprint Backlog and Sprint Goal.
    - Daily Scrum meeting that lasts at most 5 minutes.
- Friday:
    - Daily Scrum meeting that lasts at most 5 minutes.
- You perform continuous product backlog grooming.
- During Sprint Planning, you create User Stories in Taiga and break them down into smaller technical **Tasks**. To execute the work, you feed the overarching User Story to the agent for *context*, but you assign it the specific **Task** for *execution*.
- Before the agent writes the code, it generates an Implementation Plan (an Antigravity Artifact). You must set your Antigravity Review Policy to "Request Review" (rather than "Always Proceed"). You have to read, evaluate, and approve the AI's Python and database logic before it is committed, ensuring you actually understand the code being generated.
- You can use the Gemini CLI to quickly test database queries or prompt logic in the terminal before having the Antigravity agent implement it into the main codebase.
- All artifacts generated by Antigravity, e.g. task lists, implementation plans and browser recordings, must be attached to the corresponding task and/or user story in Taiga.
- In a real-world Agile environment, code commits must be traceable back to the business requirements (User Stories). **You must instruct your Antigravity agent(s): "*When you commit this code, use the commit message: 'TG-<Taiga task id>: <Taiga task description>'. Do not attempt to push*". If the AI reports a**

'sandbox error' or 'permission denied' when trying to push, it is not a technical problem - it is a safety feature. Perform the push manually in the terminal, then tell the AI to proceed with the next step.

- Daily Workflow with Antigravity
  You are the Tech Lead; Antigravity is your Development Team. You do not write code; you direct the Antigravity agent.
  - Planning (Thursday):
    - Create User Stories with clear story title and acceptance criteria in Taiga for your planned work. To do this you can provide Antigravity with the project vision statement and let it generate all the scrum artifacts for you and it can even feed them directly into Taiga via the Taiga API.
    - In Taiga, break the Story into specific tasks for Antigravity and note the task ids (e.g. #3).
  - Development (Thursday/Friday):
    - Orchestrate: Feed your Taiga Task to Antigravity's Agent Manager. Here's an example prompt: "Work on Taiga Task #1. Write a Python script to initialize the SQLite database. Once complete, stage, commit, and push the files with the commit message: 'TG-1: Initialize SQLite DB'."
      Pro-Tip: Create a `PROJECT_CONTEXT.md` file in the root of your repository that outlines your tech stack (Ubuntu, Ollama, SearXNG, Database, ...) and the strict "No external APIs" rule. Instruct Antigravity to *"Read PROJECT_CONTEXT.md before starting"* so it never hallucinates cloud-based solutions.
    - Review: Set Review Policy to "Request Review." Evaluate the AI's Python/Database logic before approving. Before approving the commit, verify that:
      - The agent included the TG-<ID> prefix in the commit message.
      - The code is strictly local (no external API calls).
      - The logic is correct based on your technical research.
    - Attach: Download the AI's Implementation Plan and attach it to the Taiga ticket.
  - The Golden Rule of committing:
    Every time you push code, make sure Antigravity has linked it to a Taiga Task ID in the commit message so the webhook works. If that's the case, you only need to run `git push` if not:
    ```
    git add .
    git commit -m "TG-<ID>: [Your short description]"
    git push
    ```
    (Example: `git commit -m "TG-1: Setup SQLite DB structure"`)
  - Verification
    - Taiga: Open your User Story (e.g., #1) and check the History/Git tab. You should see your commit message appear there automatically.

- Gogs: Check your Webhook "Recent Deliveries." You should see green checkmarks (204 No Content).

  Troubleshooting:

  - If you do not tell the AI to use the TG-<ID> format in the commit message, your Taiga board will not update.
  - If you see "Referenced element doesn't exist," verify your commit message has the correct TG-<ID> format matching an existing Story ID in Taiga.