AI Engineer Challenge - Meal Plan Generator

Introduction

The goal of this technical challenge is to evaluate your skills in building stateful AI pipelines using Large Language Models (LLMs). You will be developing an AI powered system that generates personalised meal plans for patients and automatically validates them against nutritional requirements.

All code produced during this challenge will be used solely to assess your technical capabilities.

Technologies

Your solution should be implemented using the following stack:

- **Language:** Python
- **Orchestration:** LangGraph & LangChain
- **Models:** OpenAI/Gemini

Challenge Overview

You are tasked with building an intelligent workflow that automates the creation of a **1-day meal plan** adapted to a patient profile.

You are given two files:

- `input_nutri_approval.jsonl` — different patient profiles, each with distinct caloric goals, macro targets and/or constraints.
- `input_lists.jsonl` — a set of alternative food lists. Each list contains food items and quantities such that the **macronutrients and calories of all alternatives are equivalent** within the list.

Given a patient profile, your system should:

1. Choose one or more alternative food lists for each meal.
2. Select safe and appropriate food items for the patient.
3. Remove items that are inadequate (e.g., disliked foods, incompatible with goals or restrictions).
4. Output a complete 1-day plan (5 meals) with exact nutritional totals and strict JSON formatting.

A breakfast *choice group* could look like this (illustrative only):

```
{
  "meal_type": "Breakfast",
  "time": "20:30",
  "items": [
    "1/2 porção of tapioca (50 g) (ID: 318779) OR 1 unidade de pão francês (50 g) (ID: 318778) OR 2 fatias of pão de forma de trigo integral (50 g) (ID: 318777) OR 1 porção of rap 10 (28 g) (ID: 318780) OR 3 colheres de sopa cheias of granola (33 g) (ID: 320423) OR 1 porção de cereais de flocos de milho sem adição de açúcares (30 g) (ID: 320424)",
    "1 unidade média of iogurte natural (200 g) (ID: 318955) OR 1 unidade of ovo de galinha cozido (78 g) (ID: 318951) OR 2 fatias pequenas of queijo prato (30 g) (ID: 318956) OR 2 fatias pequenas de queijo mussarela light (30 g) (ID: 320427) OR 1 colher de sopa cheia of requeijão light (30 g) (ID: 320428) OR 1 copo de bebida vegetal (200 g) (ID: 320426)",
```

```
    "1/2 unidade média of manga (60 g) (ID: 318942) OR 1/2 fatia of mamão Papaia cru (85 g) (ID: 318939) OR 1 fatia grande of melão cru (115 g) (ID: 318940) OR 1 fatia pequena of melancia crua (100 g) (ID: 318941) OR 1 unidade grande de pêssego (110 g) (ID: 318947) OR 1/2 unidade média of pêra Park crua (55 g) (ID: 318949)"
  ],
  "meal_totals": {
    "kcal": 435.3,
    "protein_g": 17.13,
    "carbs_g": 72.14,
    "fat_g": 9.47,
    "fiber_g": 5.65
  }
}
```

Core Requirements

Output Format: Strict JSON

Your final output must be valid JSON and conform **exactly** to the expected schema.

- The meal plan must include **exactly** these meals (same naming):
  - `Breakfast`, `Morning Snack`, `Lunch`, `Afternoon Snack`, `Dinner`
- Any generation that does not validate against the schema must be considered a failure and trigger regeneration.

```
{
  "daily_totals": {
    "kcal": "float",
    "protein_g": "float",
    "carbs_g": "float",
    "fat_g": "float",
    "fiber_g": "float"
  },
  "meals": [
    {
      "meal_type": "String (e.g., Breakfast, Morning Snack, Lunch)",
      "time": "String (HH:MM format)",
      "items": [
        "String containing food options separated by 'OR'. Format: '[Qty] [Unit] of [Food Name] ([Weight] g) (ID: [ID])'"
      ],
      "meal_totals": {
        "kcal": "float",
        "protein_g": "float",
        "carbs_g": "float",
        "fat_g": "float",
        "fiber_g": "float"
      }
    }
  ]
}
```

Food Item Interoperability: IDs Are Mandatory

Every chosen food item must include its Food ID (`(ID: XXXXX)`), for interoperability.

- Each meal must contain one or more *choice groups* derived from the alternatives lists.
- Each choice group must include:
  - the **selected item(s)** (final decision), each with an **ID**
  - the original **alternatives source** (or reference) to enable traceability

---

Exact Totals Consistency (No Math Drift)

The plan must be nutritionally consistent with exact totals:

- Each `meal_totals` must equal the **sum of the selected items' macros** for that meal.

- `day_totals` must equal the **sum of all meal totals**.

Patient Adaptation (Personalization)

The meal plan must be adapted to the patient profile from `input_nutri_approval.jsonl`, including:

- calorie goal and macro targets (*hard constraint*)
  - Relevant fields: `dee_goal`, `dee_goal_unit`, `macronutrient_distribution_in_grams`, `fiber_quantity_in_grams`
- diet preferences, dislikes, exclusions (*hard constraint*)
  - Relevant fields: `patient_infos.dietary_history`
- eating habits, so the patients does not need to change their habits dramatically (*soft constraint*)
  - Relevant fields: in the `patient_infos` object, inspect `dietary_history`, `eating_behaviors` and `food_diary_history_and_obs`

Safety Filtering (Conditions, Allergies, Contraindications)

If the patient profile includes **diseases/conditions/allergies/intolerances/restrictions**, the system must **filter out** unsafe or contraindicated foods. This information is present in fields, namely `food_intolerances`, `food_allergies`, `medical_history`.

- If an alternatives list includes restricted items, select a safe substitute from the same list when possible.
- If no safe option exists, the workflow must either:
  - regenerate with a different set of alternatives, **or**
  - enter a failure state with a clear validation message (and retry again)

Technical Requirements

- **Architecture Diagram:** Build and expose the architecture you chose to solve the challenge.
- **Prompt Engineering:** Use clear, system-level prompts to guide the LLM's behavior for both generation and critique.
- **Testing Script:** Provide a `main.py` or a Jupyter Notebook that runs the agent with a sample patient profile and prints the progression of the graph (e.g., "Generating..." → "Critique Failed: Too high calories" → "Regenerating..." → "Success").

Extra-Mile

These are not mandatory but will be valued during the evaluation:

- **Observability:** Integrate **LangSmith** to visualize the chain of thought and latency.
- **SML:** Iterate your solution to use a Small Language Model and compare results.

Submission Instructions

1. Submit your solution as a public Git repository (GitHub, GitLab, Bitbucket).
2. Include a `README.md` file with:
   - Setup instructions (how to install dependencies and set up `.env`).
   - How to run the agent.

- An explanation of your prompt strategy.

Evaluation Criteria

We'll assess your submission based on the following, in order of importance:

1. **Architecture:** The chosen architecture for this solution.
2. **Prompt Engineering:** Quality of the prompts used to generate and validate the output.
3. **Evaluation Strategy:** There are metrics to evaluate the output generated by the model.
4. **Robustness:** The application handles errors gracefully and produces valid JSON outputs strictly.
5. **Code Quality:** Pythonic conventions, modularity, and clean project structure.