🥷 Technical Assignment: AI Engineer Test Task

AI Engineer Task

Develop a service that takes one or more photos of a restaurant menu as input and returns the total sum of all vegetarian dishes. The solution must use open-source technologies and standard best practices for building AI-driven systems.

Functional Requirements

REST API

- **Endpoint:** `POST /process-menu`
- **Input:** 1 to 5 menu photos (`multipart/form-data` or `base64`).

Output: JSON with results:

```
{
  "vegetarian_items": [
    {"name": "Greek Salad", "price": 7.5},
    {"name": "Veggie Burger", "price": 9.0}
  ],
  "total_sum": 16.5
}
```

OCR (Optical Character Recognition)

- Use an open-source OCR engine (e.g., Tesseract OCR or equivalent).
- Extract text from the menu, including dish names and prices.

Data Structuring

- Transform the raw OCR text into a structured format.

Vegetarian Dish Classification

- Use an LLM for dish classification.

- Additionally, use a keyword dictionary (`vegetarian` , `veggie` , `salad` , `tofu` , etc.) as a fallback classification method.
- **Optional:** For a more advanced solution, you may implement a semantic search against a pre-defined database of vegetarian dishes to improve the classification accuracy of non-obvious items.

MCP Server

- **Calculation:** Identify all vegetarian dishes and sum their prices.
- Deploy the MCP server as a standalone service (HTTP/WebSocket).
- The REST API must not perform calculations directly; instead, it should call the MCP server via tool-calling.
- The MCP server returns the result, and the REST API returns it to the client.

RAG Memory

Use a local, open-source vector index (ingredients & dishes) and retrieve top-k evidence during classification via the MCP server; combine retrieval with the existing keyword fallback and return confidence + brief reasoning notes in the JSON.

Observability

Use Langsmith to Implement structured logs and cross-service tracing with a `request_id` , capturing OCR, parsing, MCP tool-calls, retrieval hits, token/latency stats, and the final decision.

HITL Loop (Optional)

If combined confidence is below the threshold, return an **uncertainty card** (JSON) instead of a verdict, listing suspect items with evidence. Accept corrections via `POST /review` , recompute deterministically

Non-Functional Requirements

- Code must be written in **Python** or **JS**
- Architecture must be split into separate components:
  - OCR + parsing inside the REST API.
  - Calculation logic delegated to the MCP server.
- Services must communicate over the network (e.g., REST API → HTTP request → MCP).
- The entire system must be runnable via Docker (2 microservices: API and MCP).

Deliverables

- Git monorepo with backend and MCP code.

- Docker setup (`docker-compose.yml`) to start the system locally with multiple instances.
- **README** including:
  - Instructions for running the system.
  - Architecture and design choices.
  - Testing approach.