Reference Notes:

Use the Chat Completion approach defining the agent's system prompt & tools definition in the application side itself, instead of the AI Assistant approach of OpenAI.

Also try to setup an LLMProcessor which can dynamically connect to the different LLM provider like factory pattern; based on the LLM provider name & model, it will instantiate the LLMProcessor instance and make it singleton; so that it will not be created every time if already built.

Milvus ingestion & retriever — Practical checklist (for students)

1. Create PDFs
    - Use Copilot-like assistant to generate textual content and convert to PDFs (suggest: use Python reportlab or pdfkit).
    - Filename & metadata example: cancellation_policy_travel.pdf, topic=travel, source=generated.
2. Text extraction
    - Convert PDFs to plain text (e.g., pdfminer.six or PyMuPDF).
3. Embeddings
    - Choose an embedding model (sentence-transformers or OpenAI embeddings). Generate embeddings per chunk.
4. Milvus setup
    - Create a collection (e.g., travel_docs) with fields: id, embedding, text, metadata (topic, filename).
5. Insert vectors
    - Chunk texts (200–500 tokens), embed each, upsert into Milvus.
6. Retriever API
    - Build a tool (HTTP function or local function) that: accepts query → embeds query → searches Milvus (top_k) → returns docs & optionally a synthesized answer (using prompt + LLM).
7. Agent wiring
   In Orchestrator, detect static knowledge intent → call retriever tool. Otherwise call action agents/tools.

Suggested Milvus collection schema (example)

- id — int64, primary key
- embedding — float_vector, dim = embedding_dim (e.g., 384)
- text — long text / varchar (chunk content)
- metadata — JSON (filename, topic, source, created_at)

How agent should decide to call RAG tool (simple heuristic)

- If user query contains tokens like: policy, rules, FAQ, how to, guide, manual, terms, details, brochure → call RAG tool.
- If query includes action verbs (book, cancel, transfer, apply for leave, order) → call transactional agent → tool → API.

(Students should implement classifier/intent detector; a small rule-based fallback is acceptable.)

Example student deliverable checklist (per assignment)

- Agents implemented and orchestrator routes properly
- Mock transactional APIs implemented
- Generated 6–15 PDFs via code assistant and saved with metadata
- Extracted text and chunked content
- Embeddings generated and inserted into Milvus collection
- Retriever tool exposed (function or HTTP) and returns useful context
- Agent calls RAG tool for static queries and tools/APIs for transactional queries
- Demo script or video (2–5 minutes) showing both flows
- README explaining how to run everything locally

Evaluation rubric (suggested)

- Architecture & Design (25%) — clear agent/tool separation, orchestrator, retriever integration
- Milvus & Data Work (25%) — number of docs, proper chunking, embeddings, metadata quality

- Functionality (30%) — RAG returns relevant contexts; agents correctly route queries; mock APIs function
- Code Quality & Documentation (10%) — readable code, instructions to run
- Demo & Edge Cases (10%) — shows sample queries of both static & transactional types, handles simple failures

Quick tips / common pitfalls

- Don’t store real PII in training docs — use anonymized/example data.
- Chunk PDFs sensibly (not too large, not too small) — 200–500 words is a safe chunk size.
- Add metadata (topic) to make filtering by collection or topic easy.
- Test retrieval relevance: manually inspect top-k responses and tune embedding model or chunking if needed.
- Provide a small fallback answer if RAG returns low-similarity results.