Fully Dressed Use Case

Generate Answer via Pyhton RAG Pipeline(LLM Based)

Name: Generate LLM based answer via Pyhton RAG pipeline

Scope: Python RAG Chatbot Application

Level: User goal primary

Actor: User (student/faculty member)

Stakeholders and Interests:

- **User:** Wants an accurate answer derived strictly from university documents, delivered via a simple CLI command.
- **Developer:** Wants `rag_orchestrator.py` to coordinate components (`retriever.py`, `answer_agent.py`) without tight coupling, ensuring the Strategy pattern is respected.
- **Evaluator:** Wants the system to support a `--batch` mode for running multiple questions at once to calculate accuracy metrics using `EvalHarness`.
- **Data Engineer:** Wants the `retriever` to successfully utilize the `VectorIndex` and Embedding Stubs when configured in `config.yaml`.
- **Optimization Engineer:** Wants the system to check the `QueryCache` before processing to save resources and use `HybridReranker` for better accuracy.
- **Course Instructor:** Wants to verify that the system generates reproducible JSONL traces and supports "Batch Mode" for evaluation.

Preconditions:

- The `config.yaml` file is valid and specifies the active strategies (e.g., `retriever_type: vector`, `reranker_type: hybrid`).
- The `index_builder.py` has been executed, and the index files (JSON/Vector store) are populated in the data directory.
- The Python environment is active with required modules.

Success Guarantee (Postconditions):

- The answer is displayed on the Standard Output (stdout).
- A comprehensive execution log is appended to the JSONL file via `trace_bus.py`.
- If caching is enabled, the new query-answer pair is added to the cache.

Main Success Scenario:

1. User triggers the system via CLI: python main.py --q "What is the prerequisite for CSE3063?".
2. System (main.py) initializes config_loader.py to read settings and instantiates the TraceBus.
3. **RagOrchestrator** initializes the pipeline components (IntentDetector, Retriever, Reranker, AnswerAgent) based on the configuration.
4. **IntentDetector** analyzes the input string and classifies the intent (e.g., Course, StaffLookup).
5. **QueryWriter** reformulates the question into optimal search terms.
6. **Retriever** queries the underlying index (e.g., KeywordIndex or VectorIndex) and returns a list of Hit objects.
7. **Reranker** calculates new scores for the hits (using logic from reranker.py) and sorts them.
8. **AnswerAgent** constructs a prompt with the user question + top-ranked chunks.
9. **AnswerAgent** calls the LLM (or Stub) models defined in models.py to generate the text.
10. System displays the final answer with citations to the User.
11. **TraceBus** writes a structured JSON log entry containing inputs, outputs, and latency for all steps.

Extensions (Alternative Flows):

- **Batch Mode Execution:**
  - User runs python main.py --batch "questions.json".
  - System detects the batch flag.
  - System iterates through the JSON file, repeating steps 3-11 for every question.
  - System outputs an evaluation summary (Accuracy/Latency) instead of a single answer.
- **Query is found in Cache:**
  - **RagOrchestrator** checks the QueryCache before starting the pipeline.
  - System finds an exact match for the question string.
  - System retrieves the stored answer.
  - System skips steps 4-9 and proceeds directly to display the cached answer (Step 10).
- **Vector Index Strategy:**
  - *Context:* config.yaml is set to index_type: vector.
  - **Retriever** delegates the search to VectorIndex instead of KeywordIndex.
  - System generates embeddings (via Stub or API) for the query.
  - System performs a similarity search and returns hits.
- **Hybrid Reranking Strategy:**
  - *Context:* config.yaml is set to reranker_type: hybrid.

    - **Reranker** combines keyword scores (BM25) with semantic scores (Cosine).
    - System re-sorts the list based on the weighted average of these two scores.
- **LLM Context Limit Exceeded:**
    - **AnswerAgent** detects that the prompt length exceeds the token limit defined in models.py.
    - System removes the lowest-ranked chunk from the context.
    - System retries the generation request.
- **No Information Found:**
    - **Retriever** returns zero hits or **AnswerAgent** determines the context is insufficient.
    - System outputs: "I cannot answer this question based on the provided documents."
    - System logs a "Miss" event in trace_bus.py.

Special Requirements:

- **Determinism:** For testing purposes, if the "Stub" LLM model is used, the output must be exactly the same for the same input every time.
- **Traceability:** The trace_bus.py must record the exact doc_id and chunk_id used in the final answer to satisfy the "Citation" requirement.
- **Modularity:** Switching from SimpleReranker to HybridReranker must require **only** a change in config.yaml, not in rag_orchestrator.py code.

Technology and Data Variations List:

- **Data Storage:** Indexes are stored as local JSON or Pickle files; no external database server (SQL/NoSQL) is used.
- **Input Interface:** System accepts input via CLI Arguments (sys.argv) or a JSON file path for batch processing.

Frequency of Occurrence:

- Continuous. Triggered every time a user (or the evaluation script) asks a question.