Requirement Analysis Document (RAD)

1. Vision

The MiniRAG Chatbot is a modular, command-line based Retrieval-Augmented Generation system designed to answer inquiries regarding the Marmara University Computer Engineering department, Faculty of Engineering, and general university policies. Unlike traditional chatbots reliant on external APIs, this system operates on a **local, ethical corpus** of documents using deterministic algorithms. The primary focus is to demonstrate high-quality Object-Oriented Analysis and Design (OOAD) using GRASP and SOLID principles, ensuring traceability from use cases to code.

2. Iteration 1 Goals

- **Core Pipeline:** Implement a sequential RAG pipeline (Intent Detection -> Query Writing -> Retrieval -> Reranking -> Answer Generation) without concurrency.
- **Determinism:** Ensure that identical inputs and configurations yield identical outputs and citations, replacing real LLMs/Vector DBs with deterministic stubs.
- **Design Patterns:** Apply **Strategy** for swappable logic (e.g., switching Rerankers), **Template Method** for the pipeline orchestration, and **Observer** for tracing execution logs.
- **Data Management:** Store data solely in local JSON/YAML files (no GUI or Database).

3. Non-Functional Requirements (NFRs)

- **Modularity (SOLID):** The system must adhere to the Open/Closed Principle; adding a new Reranker or IntentDetector should happen via configuration without modifying the RagOrchestrator code.
- **Traceability:** Every main class and method must be traceable to a specific use case step.
- **Testability:** The logic must be verifiable through at least 26 unit tests covering rules, stopwords, and ranking mathematics.
- **Configurability:** The system behavior (algorithms used, constants) must be controlled entirely via a config.yaml file.
- **Reproducibility:** Execution steps must be logged to a JSONL trace file for debugging and analysis.

4. Risks

- **Data Quality:** Since document conversion to text is manual, poor formatting or encoding issues may degrade retrieval accuracy using simple keyword matching.
- **Heuristic Limitations:** Without real semantic understanding (LLMs), the rule-based IntentDetector and keyword Retriever may fail on complex or ambiguous user queries.

5. Glossary

- **Chunk:** A fixed-size text segment (300-600 tokens) derived from a source document, acting as the atomic unit of retrieval.
- **Orchestrator:** The GRASP Controller that coordinates the flow of data between pipeline stages.

- **Strategy Pattern:** A behavioral design pattern enabling the selection of algorithms (e.g., SimpleRerankervs. NoOpReranker) at runtime via configuration.
- **Intent:** The classification of what the user wants (e.g., StaffLookup, PolicyFAQ).
- **Booster:** A mechanism to inject domain-specific terms (e.g., "office", "email") into a query when a specific Intent is detected.

4. Specific Requirements (Use Cases)

4.1 Use Case 01: Query Regulation Information

- **Primary Actor:** User
- **Level:** User-goal
- **Preconditions:**
  - The local document corpus has been pre-processed and indexed.
  - The system's configuration file (config.yaml) is present.
- **Postconditions (Success Guarantee):**
  - An answer is generated and printed to stdout.
  - The answer includes at least one valid citation referencing the source chunk.

Main Success Scenario:

1. User starts the program from the CLI, providing a configuration file and a policy-related question.
2. System reads the configuration file to load its settings.
3. System recognizes (detects intent) that the question is about university regulations.
4. System analyzes the question to understand the core topic.
5. System retrieves relevant content from its indexed documents based on the topic.
6. System determines which part of the retrieved content best answers the question.
7. System selects the best information from the most relevant content to compose a final answer.
8. System formats the final answer and attaches a citation from the source document's metadata.
9. System prints the final answer to stdout.

Extensions:

- **3a. Intent is "Unknown":** System prints a default failure message.
- **5a. No Chunks Found:** System finds no matching documents and generates a "No information found" response.

4.2 Use Case 02: Query Staff Information

- **Primary Actor:** User
- **Level:** User-goal
- **Preconditions:**
    - The local document corpus (including staff directories) is available and indexed.
- **Postconditions (Success Guarantee):**
    - An answer regarding a staff member (e.g., office location) is printed to stdout with a citation.

Main Success Scenario:

1. User starts the program from the CLI with a staff-related question.
2. System reads the configuration file.
3. System recognizes that the question is about academic staff (STAFF_LOOKUP).
4. System analyzes the question and enhances the query with additional staff-related terms (Boosters) according to configuration.
5. System retrieves relevant content using the enhanced query.
6. System determines which part of the retrieved content best answers the question.
7. System selects the best information (e.g., "Dr. Ganiz's office is...") to compose an answer.
8. System formats the answer and attaches the correct Citation.
9. System prints the final answer to stdout.

Extensions:

- **5a. No Staff Found:** System finds no chunks matching the name and generates a "Staff member not found" response.