Requirement Analysis Document (RAD)- Iteration 2

1. Vision

The MiniRAG Chatbot evolves into a Dual-Mode Semantic Search System. While Iteration 1 focused on architectural patterns, Iteration 2 introduces "intelligence" via Vector Embeddings (Cosine Similarity) and advanced heuristics. The system is designed to solve domain-specific retrieval challenges—such as distinguishing course definitions from prerequisites, pinpointing faculty offices, and interpreting complex regulations—by offering two distinct operation modes: **Simple** and **Cosine**.

2. Iteration 2 Goals

Vector Integration: Replace simple keyword matching with **Cosine Similarity** using embeddings to capture semantic meaning.

Dual-Mode Behavior:

Simple Mode: Returns raw, broad text chunks to provide full context.

Cosine Mode*:* Uses "Strict Matching" and "Header Lookback" to extract precise answers.

Precision Logic: Implement "Definition Priority" (boost course codes starting a line) and "Scope Penalty" (punish irrelevant documents like 'ÇAP' when searching for 'Yatay Geçiş').

Robust Testing: Achieve >80% unit test coverage, verifying edge cases like regex word boundaries and stopword filtering.

3. Non-Functional Requirements (NFRs)

Accuracy (Precision): The system must distinguish between a course's definition vs. a reference (prerequisite) with a specific boosting score (+300 points).

Contextual Integrity: When retrieving regulation clauses (e.g., "Item d"), the system must reconstruct the context by finding the parent header (e.g., "Suspension Crimes") via the Header Lookback algorithm.

Freshness: The indexing process must auto-purge the query_cache.json to prevent stale embeddings during development.

Testability: The logic must be verifiable via automated tests covering heuristic scoring, penalties, and citation formatting without external dependencies.

4. Risks

R1: Context Fragmentation: Splitting regulations into fixed-size chunks may separate a specific article from its governing header. Mitigation: Header Lookback logic in VectorAnswerAgent.

R2: False Positives (Name Collision): Partial name matches may return the wrong staff member. Mitigation: Strict multi-token matching logic requiring distinct matches for Name+Surname.

R3: Synonym Overlap: Generic sub-words (e.g., "af" in "Mustafa") triggering synonyms like "Student Amnesty". Mitigation: Regex-based word boundary (\b) checks in QueryWriter.

5. Glossary

Embedding: A vector representation of text used to calculate semantic similarity between a user query and corpus chunks.

Silver Bullet: A hardcoded scoring boost (+200 points) applied when a query token exactly matches alphanumeric patterns (e.g., "CSE3063") in the text.

Scope Penalty: A negative scoring mechanism (-50 points) applied to documents that match keywords but fail metadata/intent checks (e.g., wrong regulation type).

Header Lookback: An algorithm in VectorAnswerAgent that scans preceding lines in a chunk to recover lost context for bullet-point answers.

6. Specific Requirements (Use Cases)

6.1 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.

6.2 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.