Marmara University

Faculty Of Engineering

CSE3063

OBJECT ORIENTED SOFTWARE DESIGN

TERM PROJECT ITERATION 2

Requirements Analysis Document (RAD)

DATE TO : 26 December 2025 Submitted to: Büşra Arabacı

	Dept	Student Id	Name Surname
1	CSE	150122027	Şeyma Özek
2	CSE	150122060	Fatma Zeynep Kök
3	CSE	150123069	Beyza Çoban
4	CSE	150122007	Beyza Parmak
5	CSE	150123081	Sultan Kocagöz

1. Introduction

1.1 Purpose of the System

The system is designed to provide students of the Marmara University Computer Engineering Department with fast and accurate answers to their questions regarding academic staff, course content, and university regulations. In the current phase, the system has been migrated to the Python programming language and, as part of Iteration 2, has been equipped with more advanced search and ranking capabilities.

1.2 Scope of the System

The system operates via the command-line interface (CLI). Data is stored in local JSON/YAML files; within the scope of Iteration 2, vector-based search simulation, hybrid ranking, and batch processing capabilities have been added to the system.

1.3 Objectives and Success Criteria

Determinism: Ensuring that the same input and configuration always produce the same output.

Modularity: A flexible structure compliant with SOLID and GRASP principles, utilizing the Strategy Pattern.

Extensibility: The ability to add new intents and ranking algorithms via YAML/JSON configuration without requiring code modifications.

Evaluation: Reporting the system's success through metrics such as coverage@k, simple accuracy, and latency.

1.4 References

1. Marmara University. (2025). *Computer Engineering Department Official Website*. Retrieved from https://cse-eng.marmara.edu.tr/
2. Marmara University Faculty of Engineering. (2025). *Policies, Directives, and Forms*. Retrieved from https://eng.marmara.edu.tr/idari/basili-belgelerformlar
3. Marmara University. (2025). *University Regulations and Directives*. Retrieved from https://www.marmara.edu.tr/universite/yonetim/mevzuat/

1.5 Overview

This project is a Python-based, deterministic RAG chatbot designed for university-related queries. Iteration 2 introduces a vector index, hybrid reranking, and batch CLI processing capabilities. System performance is formally measured through an evaluation report covering accuracy, coverage, and latency metrics.

2. Current System

In Iteration 1, a deterministic Java-based RAG pipeline was successfully established. University and department documents were converted into local text and JSON formats , and rule-based intent detection along with keyword-matching retrieval systems were developed. All execution steps were logged in JSONL format , and the codebase was migrated to Python for Iteration 2.

3. Proposed System

3.1 Nonfunctional Requirements

Usability & Interface: The system is strictly CLI-based, utilizing YAML/JSON for all configurations, data storage, and the newly introduced batch mode (--batch).

Reliability & Performance: All operations are deterministic and sequential, ensuring identical inputs/configurations yield identical outputs. Every pipeline step is logged in JSONL format for full transparency.

Supportability & Extensibility: Adhering to SOLID and GRASP principles, the Python-based architecture uses the Strategy Pattern to allow swapping or modifying components—such as the HybridReranker and VectorIndex—at runtime via configuration files without source code changes.

Evaluation: System performance and retrieval success are formally measured and reported through coverage@k, simple accuracy, and latency metrics.

Implementation Constraints: The system operates without a GUI, database, or external APIs; all behavior is based on local documents and deterministic stubs.

6. Glossary

- VectorIndex: A local index storing numeric vectors to enable similarity-based retrieval.
- Hybrid Reranker: A ranking strategy that merges keyword-based scores and vector similarity to reorder hits.
- Batch CLI: A command-line mode (--batch) used to process multiple queries from a single input file in one run.
- Metrics: Performance measures reported by the system, specifically coverage@k, accuracy, and latency.
- EvalHarness: A testing framework designed to automate system evaluation and generate metric reports.

UC1	Answer Question
Primary Actor	Student
Stakeholders	**Student:** Wants faster answers (via Cache) and higher accuracy (via Vector/Hybrid search) .
	Department: Wants to ensure that complex questions regarding regulations are answered consistently based on official documents.
Scope	RAG Chatbot System
Level	User Goal
Preconditions	The system is running via CLI.
	Configuration (config.yaml) is loaded with selected strategies (e.g., Hybrid Reranker, Vector Stub).
	QueryCache is active.
Postconditions	The answer and citations are displayed to the user.
	The question-answer pair is saved to QueryCache.
	A single trace event sequence is appended to the JSONL log.
Main Success Scenario	**Student** enters a question into the command line (e.g., "What are the prerequisites for CSE3063?").
	System (RagOrchestrator) hashes the question and checks the **QueryCache**.
	System does *not* find the question in the cache (Cache Miss).
	System (IntentDetector) analyzes the question and identifies the intent.
	System (QueryWriter) generates search terms for the question.
	System (Retriever) executes the search using the selected strategy (Keyword, Vector, or Hybrid) as defined in config.yaml..
	System (Reranker) reorders the retrieved hits using the selected reranking logic (e.g., Cosine, Simple or Hybrid).
	System (AnswerAgent) composes the final answer

	from the top-ranked hits and appends citations.
	System saves the result into QueryCache.
	System displays the Answer, Citations, and Latency.
	System logs the full trace to JSONL.
Extension	**2a. Cache Hit:** System finds the question hash in QueryCache. **2a.1.** System retrieves the stored answer immediately. **2a.2.** Steps 4-9 are skipped (Zero Latency). **2a.3.** System displays the answer with a note: *"Retrieved from Cache"*.
Special Requirements	**Deterministic:** Same input + config must yield exact same result.
	Latency: Cached responses should be significantly faster.
	Configurability: The pipeline steps (e.g., which Reranker to use) depend entirely on config.yaml.

UC2	**Batch Evaluation**
Primary Actor	Student
Stakeholders	**Student:** Wants to verify the system's ability to answer a list of study questions correctly without manual entry.
Scope	RAG Chatbot System
Level	User Goal
Preconditions	A JSON file containing a list of questions (e.g., eval/questions.json) exists.
	config.yaml is configured for batch processing.
Postconditions	A Batch Report (JSON/Text) is generated containing answers and metrics for all questions.
	All processed questions are added to QueryCache.
	Detailed JSONL logs are created for the entire session.

Main Success Scenario	Student runs the CLI command with the --batch argument.
	System loads the question list.
	System iterates through each question automatically.
	Loop: For each question, the system **executes the pipeline:** 4.1. Check Cache (if hit, skip to 4.5). 4.2. Detect Intent & Write Query. 4.3. Retrieve & Rerank (using Vector/Hybrid as configured). 4.4. Generate Answer & Update Cache. 4.5. Compare result with ground truth(EvalHarness).
	System calculates aggregate metrics (Total Accuracy, Avg Latency).
	System saves the results to an output file/report.
	System displays a summary message (e.g., "Batch completed: 18/20 correct")
Extension	**1a. Invalid File:** System displays error and aborts. **4a. Pipeline Error:** If a specific question causes a crash, system logs the error, records a "Fail" for that item, and proceeds to the next question.
Special Requirements	**Consistency:** The answers generated in Batch mode MUST be identical to answers generated in UC1 for the same questions and config.
	Performance: Must verify that the Cache is effectively populated during the batch run.