Knowledge Graph-based Retrieval-Augmented Generation System for Domain-Specific Information Extraction with Glossary-Aided Responses

By

De Silva K.N.N.C.

Jayawickrama D.S.K.

Weerasingha W.G.K.M.

BICT Honours Degree **2024**

Knowledge Graph-based Retrieval-Augmented Generation System for Domain-Specific Information Extraction with Glossary-Aided Responses

Authors	De Silva K.N.N.C. Software Technology Department of Information and Communication Technology Faculty of Technology University of Sri Jayewardenepura ICT/20/826 ict20826@fot.sjp.ac.lk
	Jayawickrama D.S.K. Software Technology Department of Information and Communication Technology Faculty of Technology University of Sri Jayewardenepura ICT/20/862 ict20862@fot.sjp.ac.lk
	Weerasingha W.G.K.M. Software Technology Department of Information and Communication Technology Faculty of Technology University of Sri Jayewardenepura ICT/20/956 ict20956@fot.sjp.ac.lk

Main Supervisor	Dr. Chamara Liyanage Academic Supervisor, Department of Information and Communication Technology, University of Sri Jayewardenepura
External Supervisor	Mr. Indika Hiran Wijesinghe Assistant Director IT, Sri Lanka Tea Board

Table of Contents

List of Figures

1. Introduction

1.1 Overview

The proposed research aims to develop an AI-powered agent system that integrates domain-specific knowledge through a Retrieval-Augmented Generation (RAG) model using knowledge graph. Many organizations, especially those with longstanding histories in specific fields, rely on traditional physical files to store legal documents, which serve as key references for decision-making. However, this manual approach can be inefficient and leads to the loss of critical institutional knowledge when experts retire or leave the organization. The new system seeks to address this by extracting and organizing information from historical documents to provide users with accurate, contextually relevant insights. By simulating interactions with a domain expert, the system will streamline the retrieval and interpretation of complex historical data, thus supporting more informed decision-making processes.

The proposed RAG system will combine the capabilities of Large Language Models (LLMs) with a knowledge graph with integrating domain specific glossary to generate accurate responses to user queries. Textual data will be extracted into nodes and relationships and then stored in a graph database, enabling efficient information retrieval while clarifying relationships between different data points. This approach ensures prompt access to past decisions, facilitating deeper insights into future organizational strategies. The project also includes the development of a web-based document management portal and an intuitive user interface, allowing authorized personnel to maintain the knowledge graph and engage with the system. Additionally, this integration of domain-specific knowledge will improve response precision, making the system an essential tool for enhancing both data management and decision-making processes across organizations.

1.2 Project Scope

A modern, web-based portal will be developed to manage the organization's repository of historical documents. This portal will allow authorized users to upload, view, and validate documents, ensuring that the knowledge base remains comprehensive, up-to-date, and accessible to all personnel. This feature is designed to streamline document management processes in organizations that previously relied on manual handling of paper-based archives.

1. Development of Document Management Portal

   The project will develop a modern web-based portal for managing a vast repository of historical documents. It will allow authorized users to upload, view, and validate documents, ensuring the knowledge base remains updated and comprehensive.

2. Construction of Knowledge Graph

   A detailed knowledge graph will be created using textual content from decades of documents. Text will be divided into smaller chunks as nodes, and relationships between these nodes will be mapped to reflect interconnected information. This graph, enhanced by a domain-specific glossary, will enable accurate and context-aware information retrieval.

3. Integration of Advanced Technologies

   The system will leverage advanced technologies, including Large Language Models (LLMs), LangChain, and Neo4j, for robust information retrieval. User prompts will be analyzed to extract key entities and relationships, matched against the knowledge graph, and enhanced with glossary terms for precise query alignment, simulating expert-level responses.

4. User Interface Design

   A user-friendly interface will be developed to enable seamless interaction with the system. Users can ask questions in natural language, and the system will retrieve and

present relevant information from the RAG system, supported by LLM-generated responses. The interface will cater to users with varying technical expertise.

5. Enhancement with Domain-Specific Glossary

   The system will integrate with a novel approach of domain-specific glossary to enhance the accuracy of information retrieval. By incorporating specialized terminologies, it will ensure user queries are correctly interpreted, and contextually relevant responses are provided, tailoring the system to meet organizational needs.

1.3 Research Problem

Organizations with extensive histories face significant challenges in managing and retrieving critical information from their large archives of historical documents. These documents, often containing vital decisions and institutional knowledge, are essential for informed decision-making. However, many organizations still rely on outdated, manual methods to store and access this information. As experienced personnel retire, there is a growing risk of losing valuable domain expertise, leaving newer employees without the deep knowledge necessary for effective decision-making. The lack of an efficient knowledge transfer process further exacerbates this issue.

This situation highlights the urgent need for an advanced, automated system capable of managing, retrieving, and contextualizing historical data. By developing such a system, organizations can preserve their institutional knowledge, provide easier access to critical information, and ensure continuity in decision-making processes. This solution is particularly important as organizations adapt to modern challenges while still relying on decades-old data and insights.

2. System Requirements Specification (SRS)

The proposed Knowledge Graph based Retrieval-Augmented Generation (RAG) system design to enhance information retrieval and extraction from scanned PSDs. The system integrated cutting-edge technologies, including knowledge graphs (Neo4j), and Large Language Models (LLMs). It is tailored to ensuring accurate and relevant responses by leveraging a domain-specific glossary. The goal is to streamline document processing, improve search accuracy, and enable users to query vast amounts of data using natural language.

Functional Requirements

1. Document Management
    - Upload Document via Frontend UI - Users can upload documents using a web-based frontend interface. Supported format is scanned PDF files.
    - Storage in Azure Cloud - Uploaded documents are securely stored in an Azure database. The system ensures that document metadata (such as upload date and file name) is indexed for efficient retrieval.
2. Conversational Interaction with Documents
    - Chat with Uploaded Documents - Users can initiate a chat session for all the uploaded document using the frontend UI. Queries entered by the user are processed using the Retrieval-Augmented Generation (RAG) system, which fetches relevant content from the documents and provides a context-aware, human-like response.
    - Context Retention - The system retains the context of the conversation for the duration of the chat session, enabling follow-up questions to be answered accurately.
3. Document Search

- View Uploaded Documents - Users can access a list of all their uploaded documents through the frontend UI. The document list displays metadata such as document name, upload date, size and description.
- Search functionality - A search bar allows users to find specific documents by entering keywords or applying filters (e.g., by upload date or document type). Search results are displayed in real-time with a preview option.

4. Chat History Management
    - Access Chat History - Users can view and revisit chat histories for previously conducted conversations. Chat history is organized by document name and timestamp, allowing users to quickly locate specific sessions.
    - Export Chat History – Users can export chat histories in PDF or text format for offline references.

Non-Functional Requirements

1. Performance
    - The system should return answers to user queries with low latency, ensuring a smooth user experience. This includes optimizing both the vector search and graph traversal processes.
2. Scalability
    - The system must be scalable, allowing for the integration of more documents over time without significant performance degradation.
3. Security
    - Only authorized personnel should be able to access the Document Management Portal. All sensitive data should be securely stored and protected from unauthorized access.
4. Usability

- The web app interface must be intuitive and easy to navigate, ensuring that users with varying levels of technical expertise can interact with the system effectively.

5. Reliability
    - The system must be reliable, with minimal downtime, ensuring consistent availability for users.
6. Maintainability
    - The system should be designed with maintainability in mind, allowing for easy updates, bug fixes, and feature enhancements.

System Features

1. Document Management
    - Upload and Storage - Documents uploaded via the frontend UI are securely stored in Azure and indexed for retrieval.
2. Conversational Interaction
    - Users can query uploaded documents and receive context-aware responses.
    - The system ensures smooth conversational flow with context retention.
3. Document Search and Organization
    - View a list of uploaded documents with sorting and filtering options.
    - Search for specific documents using metadata or keywords.
4. Chat History
    - View and export chat histories for previously conducted sessions.

Constraints

- Only English-language documents are supported in the initial implementation.
- Document storage and retrieval depend on Azure cloud services.

- The system is designed for use on modern web browsers.

Assumptions and Dependencies

- Users will have reliable internet connectivity to access the system and interact with uploaded documents.
- LLM models will be hosted and managed by DeepInfra and Groq platforms, ensuring efficient utilization of computational resources, including GOUs, for fats and accurate responses. These platforms provide scalability and streamline model deployment for handling real-time queries.
- Azure cloud infrastructure will be used for document storage, ensuring high availability and security.

3. Methodology

The methodology for this project is designed to effectively achieve the development of a Knowledge Graph-based Retrieval-Augmented Generation (RAG) system. The steps below outline the key processes and technologies used:

1. Document Management Portal:
    - Develop a web-based portal using React for the frontend and Spring Boot/FastAPI for the backend.
    - Features include secure upload, view, search, and delete options for documents.
    - Ensure seamless integration of newly uploaded documents into the knowledge graph.
2. Knowledge Graph Construction:
    - Use the LLMGraphTransformer tool in LangChain to extract entities and relationships directly from documents.
    - Generate a structured knowledge graph stored in a graph database for enhanced semantic understanding.
    - Avoid reliance on traditional Named Entity Recognition (NER) and relation extraction models for efficiency.
3. Context Retrieval for Queries:
    - Perform graph traversal to retrieve relevant context for user queries based on the semantic structure of the knowledge graph.
    - Utilize the knowledge graph to handle complex, multi-layered queries effectively.
4. Domain-Specific Glossary Integration:
    - Maintain a standalone dataset for glossary definitions related to the domain.

- Automatically fetch glossary definitions when domain-specific terms are detected in user queries.
- Merge glossary data with context for improved response accuracy.

5. Response Generation:
    - Combine the user query, graph-retrieved context, and glossary definitions.
    - Use advanced LLMs hosted via DeepInfra and Group for processing and generating accurate, contextually rich answers.
    - Display responses in a user-friendly format through the web app's frontend interface.
6. Technological Framework:
    - Frontend: React for responsive and intuitive user interface.
    - Backend: Python Flask, Spring Boot for efficient data processing and API management.
    - Graph Database: Store and traverse the knowledge graph to retrieve structured insights.
    - Cloud Infrastructure: Use Azure for document storage to ensure high availability, scalability and security.
    - Document Database: MongoDB document database used as document meta data and chat history storage.

3.1 Technologies and Techniques

3.1.1 Technologies

Neo4j (Graph Database)

- Neo4j is a powerful graph database used for managing and querying the knowledge graph. It will store the structured data extracted from documents in

the form of entities (nodes) and relationships (edges). Neo4j allows efficient graph traversal, making it suitable for retrieving contextual information based on the connections between entities.

LangChain

- LangChain facilitates the integration of language models (LLMs) with external data sources, such as databases and knowledge graphs. It helps manage the retrieval process, enabling the creation of knowledge graphs using the LLMGraphTransformer and passing relevant context to the LLM.

Llama 3.1 (Large Language Model)

- Llama 3.1 is the core model used for generating answers based on the user's query and retrieved context. It will analyze the input, apply natural language processing, and generate coherent and accurate answers, especially when enhanced by domain-specific terms from the glossary.

React (Frontend Library)

- React will be used to build the web application's frontend. It provides a dynamic and responsive interface for users to interact with the Document Management Portal, submit queries, and view the system's responses.

Python (Flask), SpringBoot and FastAPI (Backend Frameworks)

- These three frameworks will handle the backend logic of the application. Python will manage the overall system architecture, while FastAPI will handle interactions with the document processing pipeline and databases. This combination ensures a robust and scalable backend system.

3.1.2 Techniques

Knowledge Graph Construction

- The LLMGraphTransformer in LangChain will be used to analyze document text and automatically generate entities (nodes) and relationships (edges) for the knowledge graph. This eliminates the need for separate Named Entity Recognition (NER) and Relation Extraction models. The resulting graph will provide a structured context for queries, stored in Neo4j.

Context Retrieval from the Knowledge Graph

- When a query is input by the user, the system performs graph traversal in Neo4j to retrieve relevant entities and relationships. This ensures that structured data is leveraged to enhance the accuracy of responses.

Glossary Integration for Domain-Specific Understanding

- The system maintains a domain-specific glossary as a separate dataset. When a query contains terms from the glossary, their definitions are fetched and included in the prompt to the LLM. This ensures accurate handling of technical or specialized terminology.

LLM Prompt Engineering with Temperature Control

The LLM is instructed via prompts to only generate an answer based on the provided context. If the context is insufficient, it will inform the user that the question cannot be answered. Additionally, the temperature parameter is set to zero, minimizing the likelihood of hallucinations or irrelevant answers from the LLM.

4. System Design Overview

The system design for this project is centered around a multi-tier architecture that integrates a web-based frontend, a robust backend, and a graph database. The frontend, developed using React, provides an intuitive interface for users to interact with the system, submit queries, and manage documents. The backend, built with Python, SpringBoot and FastAPI, handles the processing of user requests, manages the knowledge graph stored in Neo4j, and interfaces with LangChain and Llama 3.1 to generate responses. The knowledge graph is the core component, where entities and relationships are extracted from documents and stored in the graph database. This architecture ensures efficient data retrieval, secure document management, and seamless integration of advanced AI techniques for natural language processing.

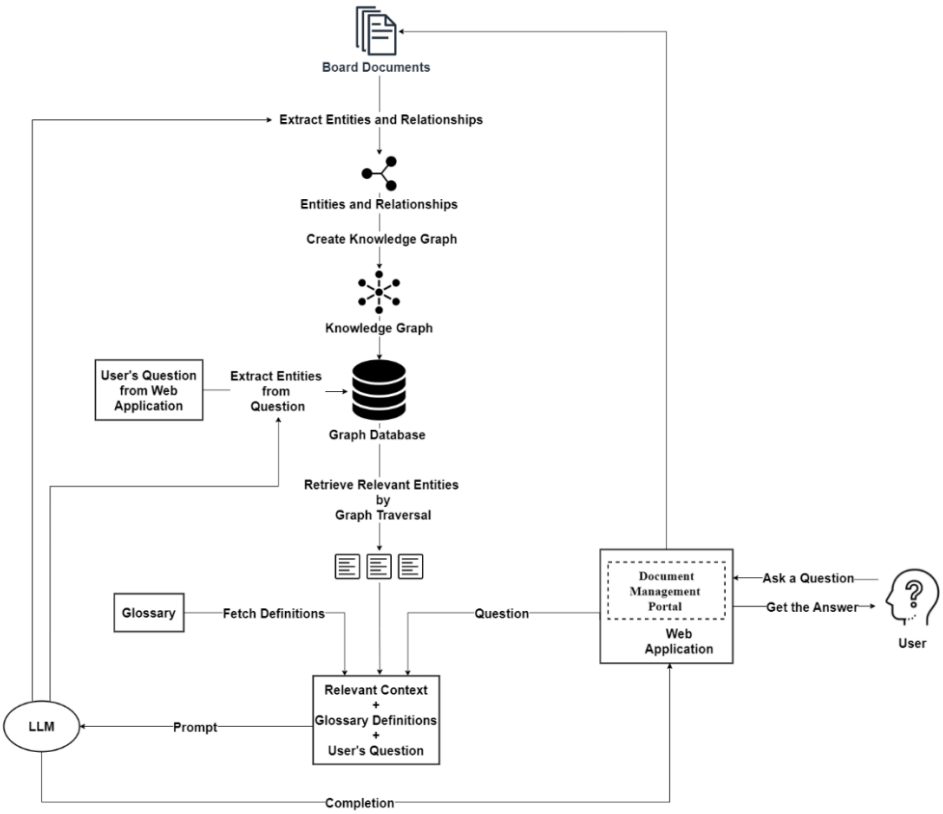

Figure 1: System Design

Frontend design

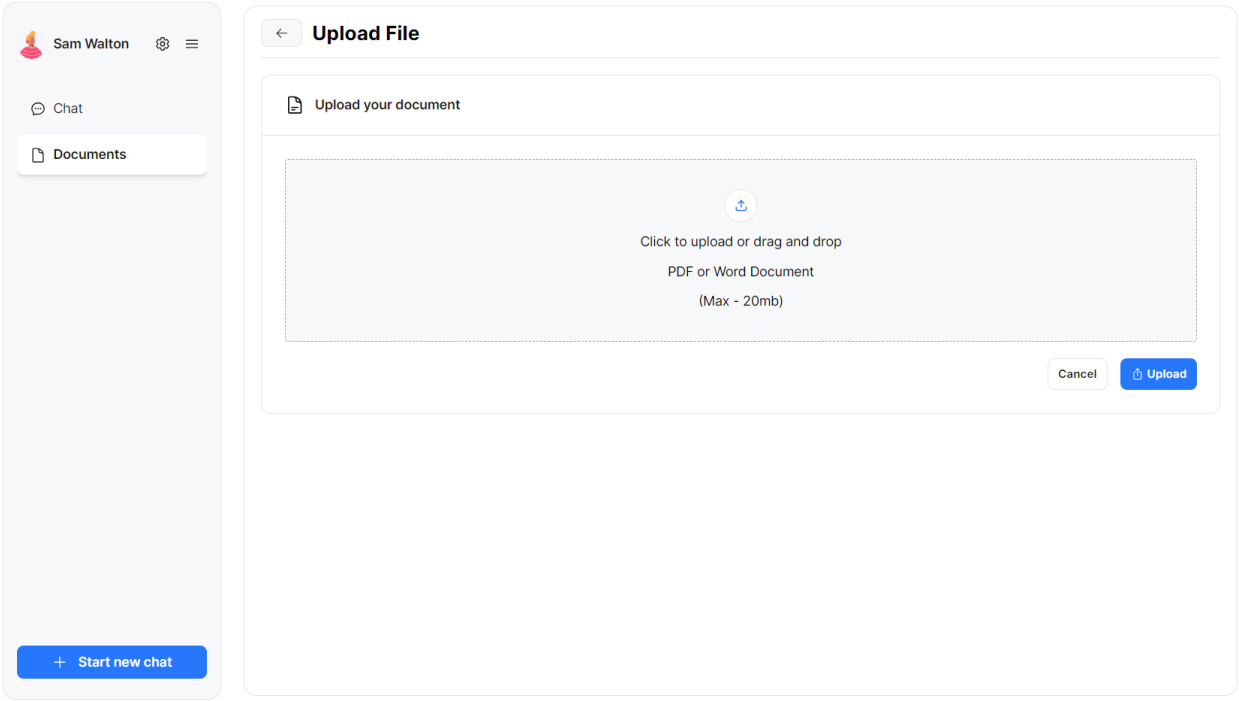

Figure 2: User Interface 1

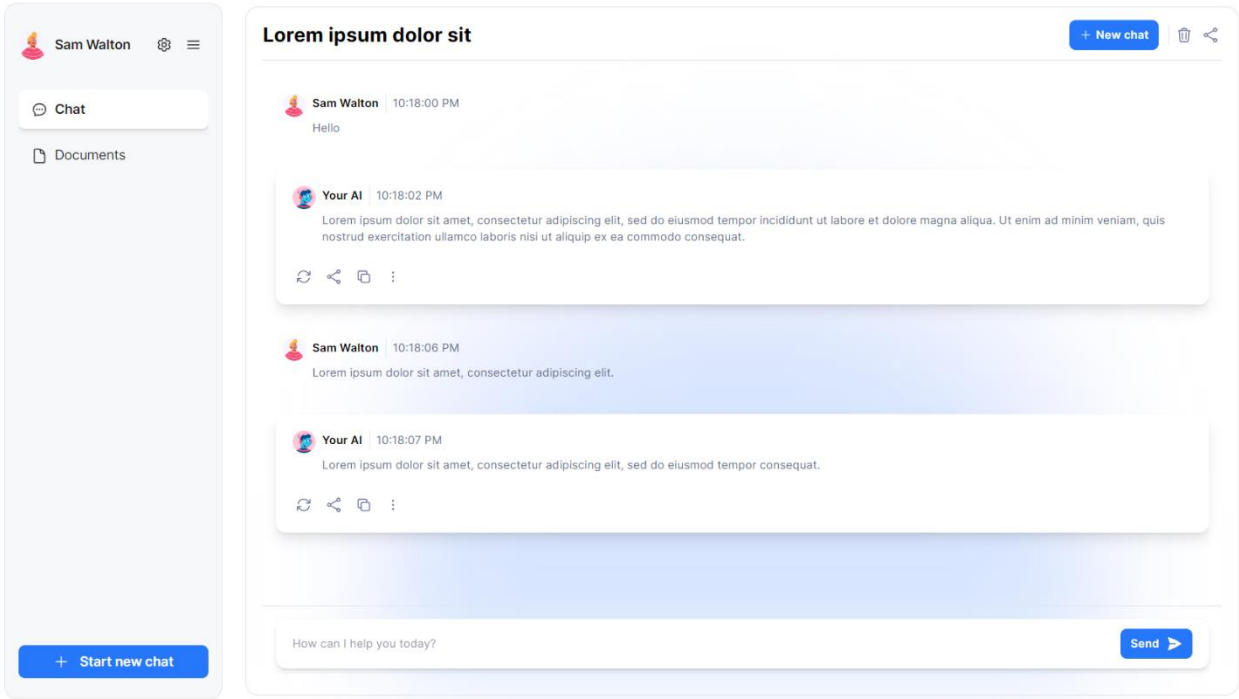

Figure 3:User Interface 2

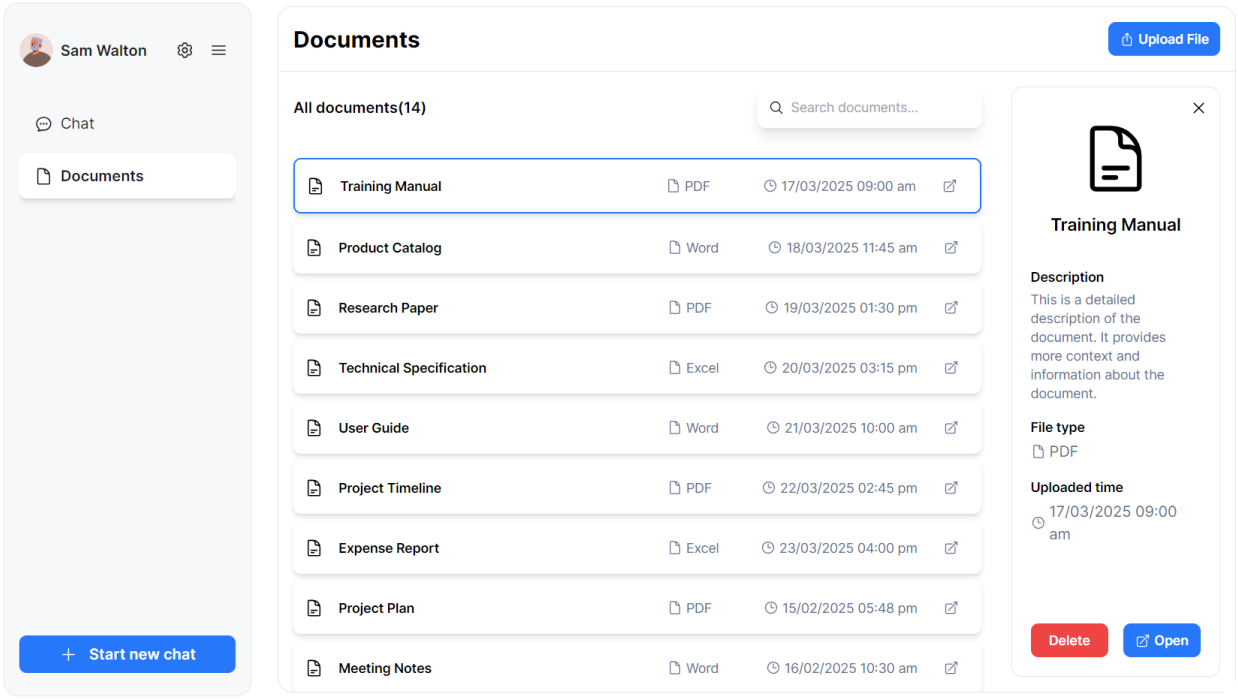

Figure 4:User Interface 3

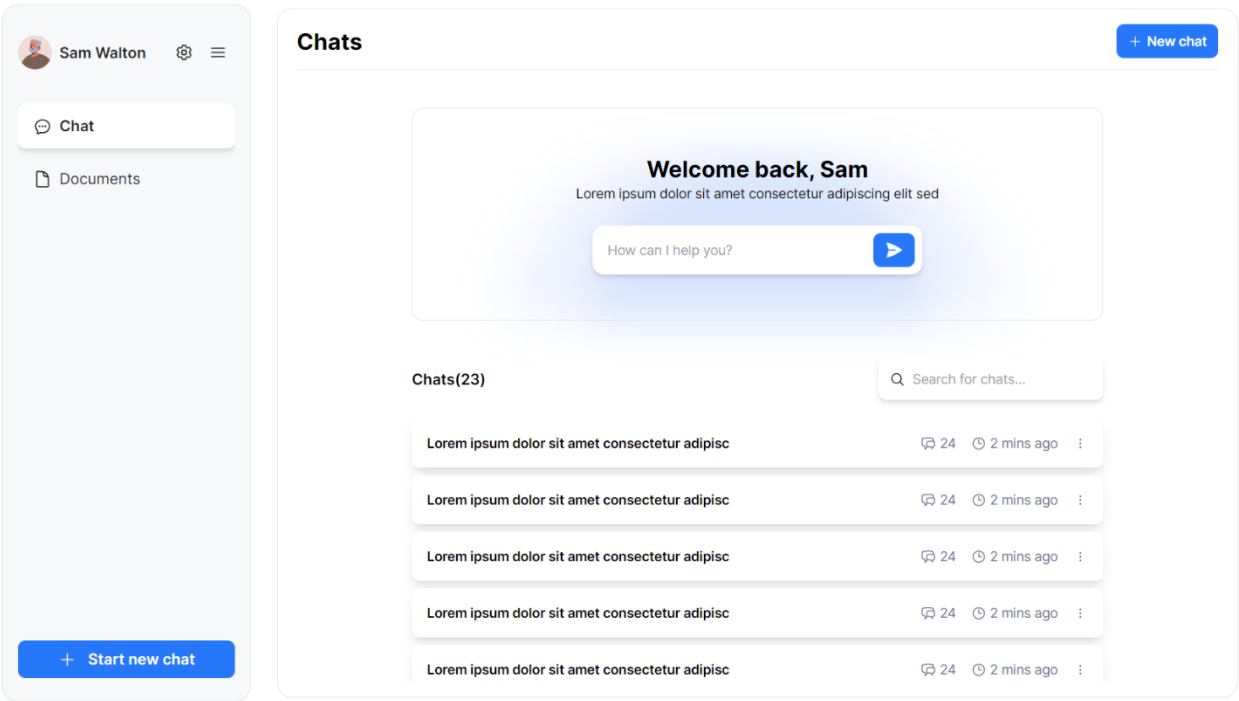

Figure 5: User Interface 4

Backend Design

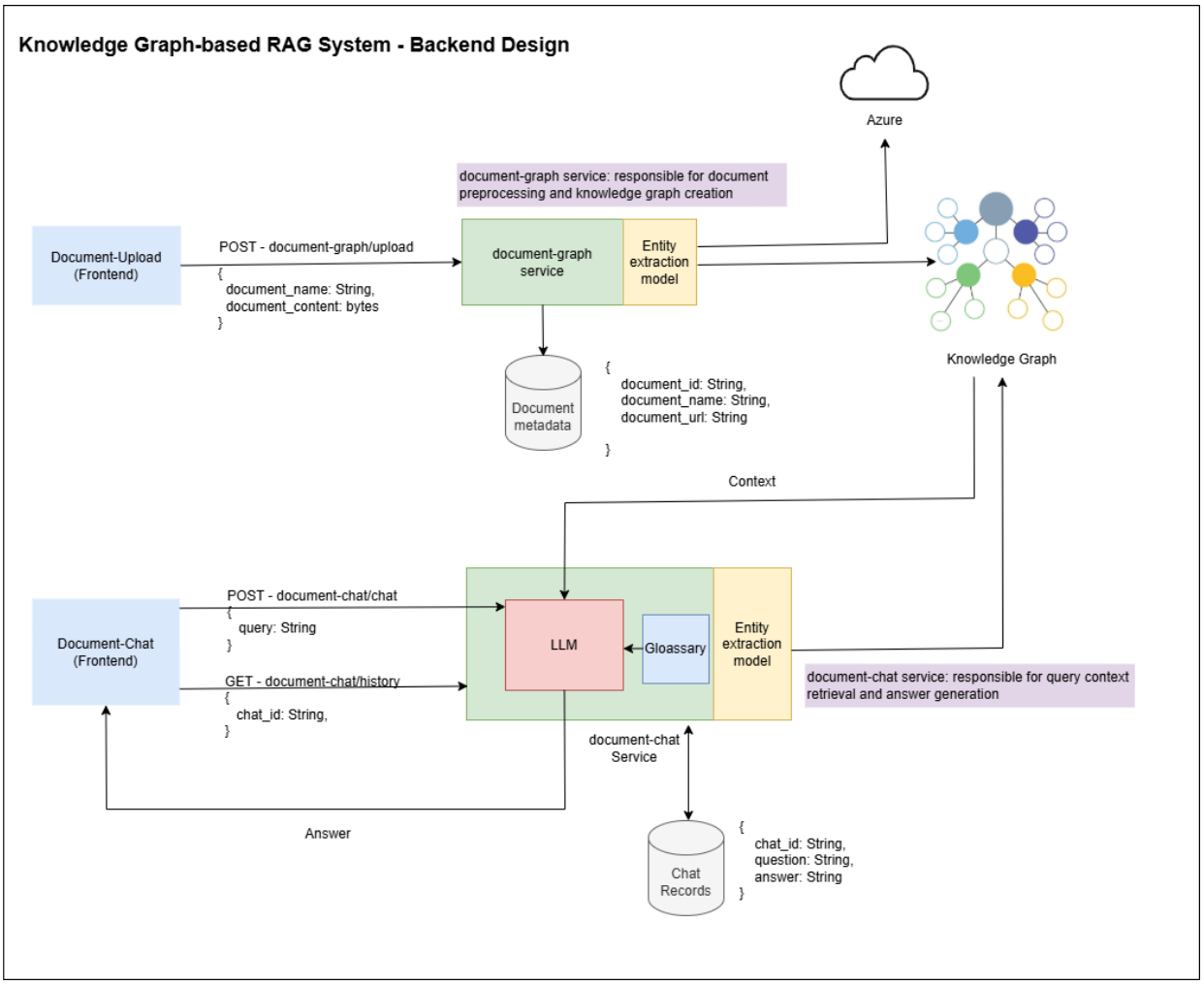

Figure 6: Backend Architecture

5. UML Diagrams

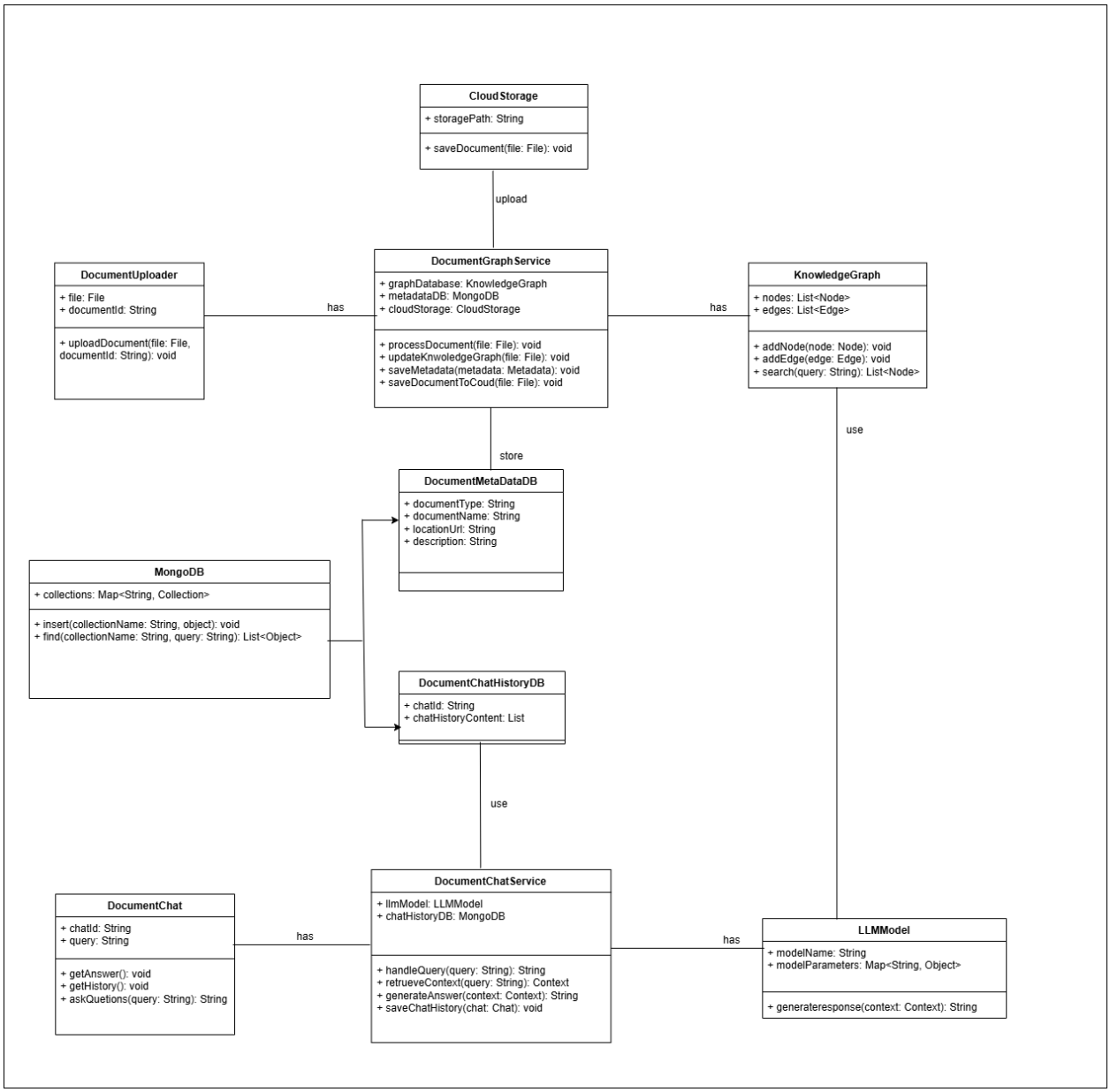

Figure 7: Class Diagram

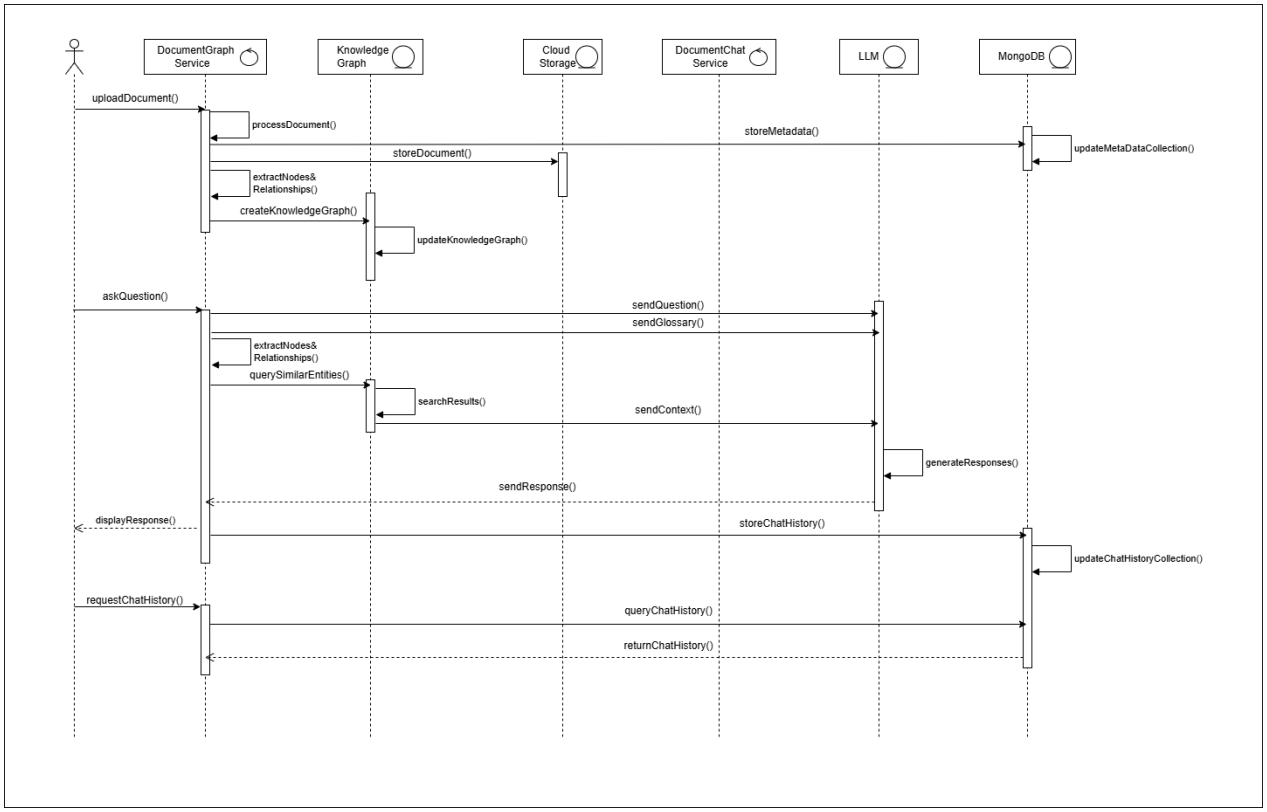

Figure 8: Sequence Diagram