Table of Contents

1. Introduction of Document

Introduction

This project aims to develop a conversational AI assistant capable of answering user questions based on a provided knowledge base. The main goal is to create an intuitive and effective way for users to retrieve information from documents. The knowledge base can be any document file, uploaded by an administrator, which the AI agent uses to answer user queries in a natural, conversational manner.

This project leverages an existing Large Language Model (LLM) to create a lightweight AI agent that integrates easily with a simple web-based user interface. Unlike generic search engines or static information retrieval systems, this agent will understand and respond to queries based on the uploaded document context, allowing for a more focused, relevant, and interactive user experience.

The final application aims to simplify information extraction, allowing users to ask questions conversationally while accessing information stored in the uploaded knowledge base files. This provides an efficient, fast, and user-friendly alternative to manual document scanning and searching.

2. Scope (of the project)

This project aims to create an AI assistant that answers user queries based on uploaded documents. The project encompasses two main user roles:

Admin and User:

Admins are responsible for managing the AI, including uploading documents that form the knowledge base and configuring settings. Users interact with the AI through a chat interface to query this knowledge base.

Key functionalities include document upload and management, where Admins can add plain text or Word documents to the knowledge base and using similarity-based search results to retrieve the most relevant content and generate responses accordingly. The AI provides one interfaces for simple queries using a basic web chat interface. An open-source LLM will process user queries, match them to the knowledge base, and generate relevant responses using the knowledge base. When the knowledge base lacks information to answer a query, the AI will display a fallback message, which the Admin can customize for a better user experience. The project will be coded fully in Python, with Flask as the web framework and SQLite for lightweight data storage. It focuses on creating a working prototype using pre-trained LLMs like GPT-4o or LLaMA. The scope excludes custom LLM training, multi-language support, complex UI, and scalability for large datasets, aiming instead at a simple and functional local deployment. Only text inputs will be supported, with English as the sole language for responses.

3. Functional Requirements Non Functional requirements

3.1 Functional Requirements

Functional requirements specify the functionality and capabilities of the AI Agent. Below are listed each of the key features:

3.1.1 Admin Functionalities

➢ **Document Upload**: Admin can upload a Word or text document that forms the knowledge base for the AI agent. This document will be analyzed and indexed for querying purposes.

➢ **Fallback Response Configuration**: Admin has the ability to set a custom fallback message that will be used when the AI agent cannot provide an answer from the uploaded knowledge base.

➢ **Basic Management Settings**: Admin can configure the interaction style of the AI, such as setting limits on the length of responses.

3.1.2 User Functionalities

➢ **Chat Interface Access**: Users can interact with the AI agent via a **web-based chat interface**.

➢ **Ask Questions**: Users can submit questions in natural language, and the agent will provide responses based on the information in the uploaded document.

➢ **Handle Unknown Queries**: If the AI agent cannot find an answer to a query, it will return a default response, as configured by the Admin, which explains the limitation clearly to the user.

3.1.3 Knowledge Base and Query Processing

➢ **Document Indexing**: Upon upload, the document will be parsed and indexed for efficient query retrieval. This indexing is important to enable quick and relevant responses during user interactions.

➢ **Query Interpretation**: The AI agent will use the selected LLM to interpret user queries, extract the key meaning, and match it against relevant information in the knowledge base.

➢ **Response Generation**: Based on the matched content, the AI agent will generate an appropriate response using the LLM capabilities, ensuring that answers are phrased conversationally.

3.1.4 Interfaces

➢**Simple UI interface:**

Initially, the chatbot will be accessible via a simple Chat User Interface for testing and prototyping.

➢ **Web-based Chat Interface**: A basic web-based interface (using Flask) will be developed where users can enter queries and receive responses in real-time.

3.2 Non-Functional Requirements

Non-functional requirements specify criteria that the system should meet beyond the basic functionality, including quality attributes, performance, usability, and other system properties.

3.2.1 Performance Requirements

➢ **Response Time:** The system should provide responses to user queries within a maximum of 3-5 seconds for documents (approximately 50-100 pages).

➢ **Scalability:** The system should be able to handle multiple document uploads from Admin users, although it will not support very large datasets (less than 100 pages) due to LLM limitations.

3.2.2 Reliability Requirements

➢ **Error Handling:** The system should provide clear error messages if it fails to process a query or upload. Admin settings should include handling unsupported document types or incorrect formats.➢ **Fallback System:** In cases where the LLM is unable to provide an answer, the default fallback response should inform the user that the information is unavailable.

3.2.3 Usability Requirements

➢ **User Interface Simplicity:** The interface (command line or web) should be user-friendly, with simple prompts and instructions so that even non-technical users can interact easily.

➢ **Web Interface Features:** The web interface should provide a basic input box for user queries and a display area for AI responses, ensuring minimal clicks and easy readability.

3.2.4 Security Requirements

➢ **Knowledge Base Access:** Only Admin users will have permission to upload, modify, or delete knowledge base documents.

➢ **Access Control:** The system should have basic role-based access control to distinguish between Admin and User roles.

3.2.5 Portability Requirements:

➢ **Platform Compatibility:** The AI agent application must be runnable on multiple platforms (Windows, Mac, Linux) with minimal setup, leveraging Python as the main language.

➢ **Deployment Ease:** The web interface must be lightweight and easily deployable using Flask, without requiring complex installation or server-side configurations.

3.2.6 Maintainability Requirements:

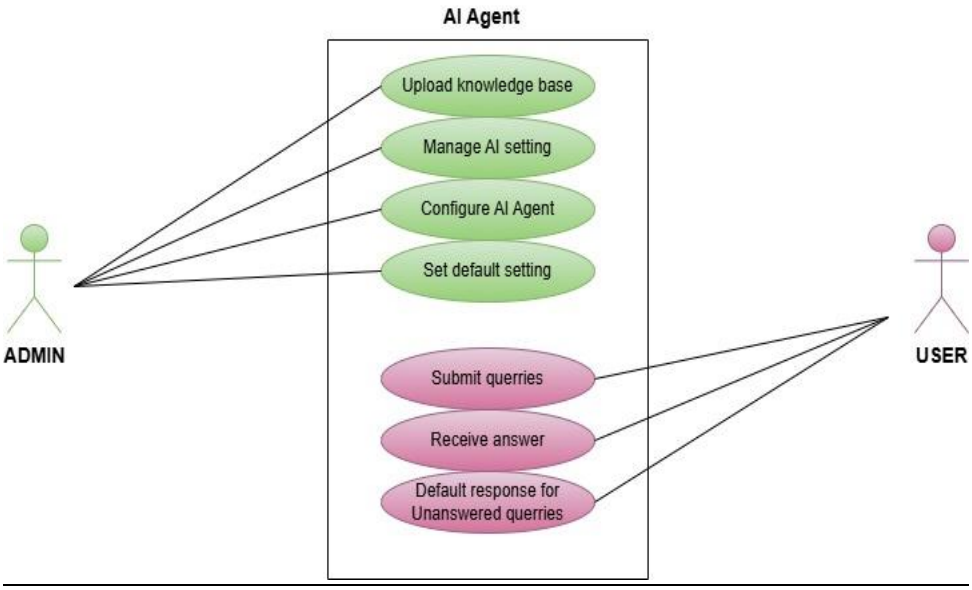
➢ **Modularity:** The application should be modular, with clear separation between the knowledge base processing, LLM query handling, and the user interface logic.

3.2.7 Ethical and Limitations Requirements:

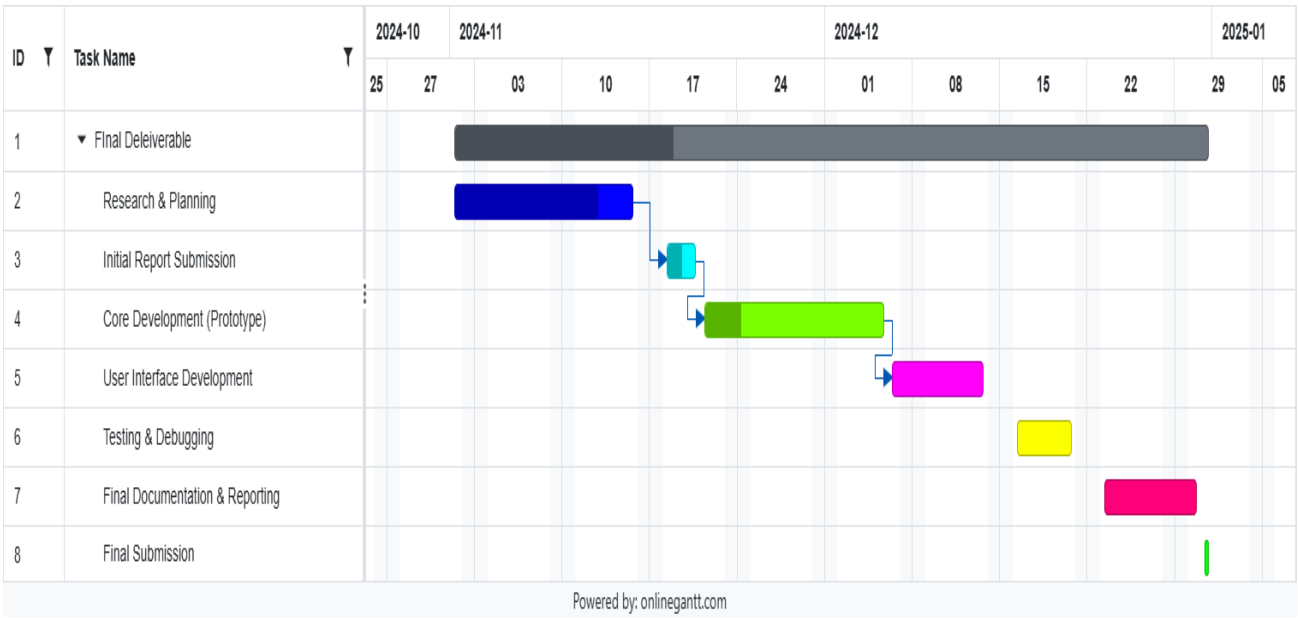
➢ **Bias and Fairness:** The LLM should be used cautiously, with fallback messages to inform users if the information is uncertain or if the model’s knowledge base is not broad enough to provide an authoritative answer.

➢ **Data Privacy:** Uploaded documents will only be used internally by the agent for generating responses, with no external data sharing or collection

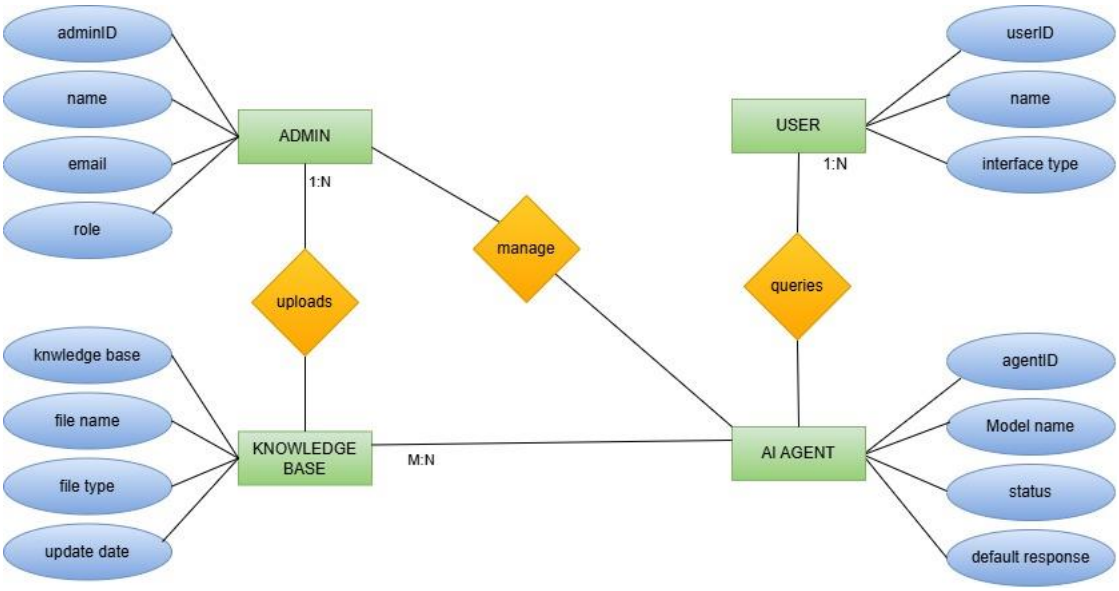
4. Use Case Diagarm



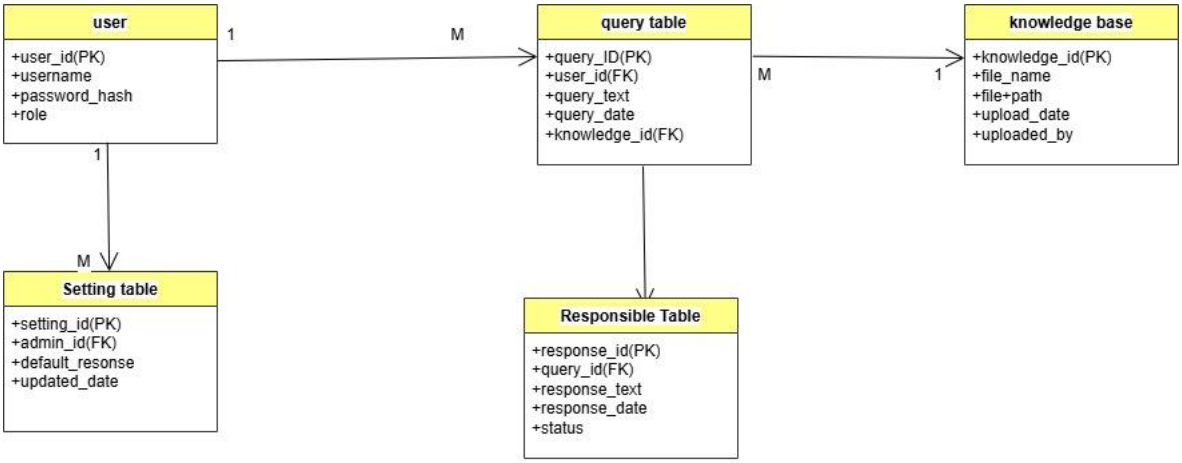
5. Work Plan



6. Entity Relationship Diagram (ERD) (To be developed using Microsoft Visio or any other drawing software of your choice)

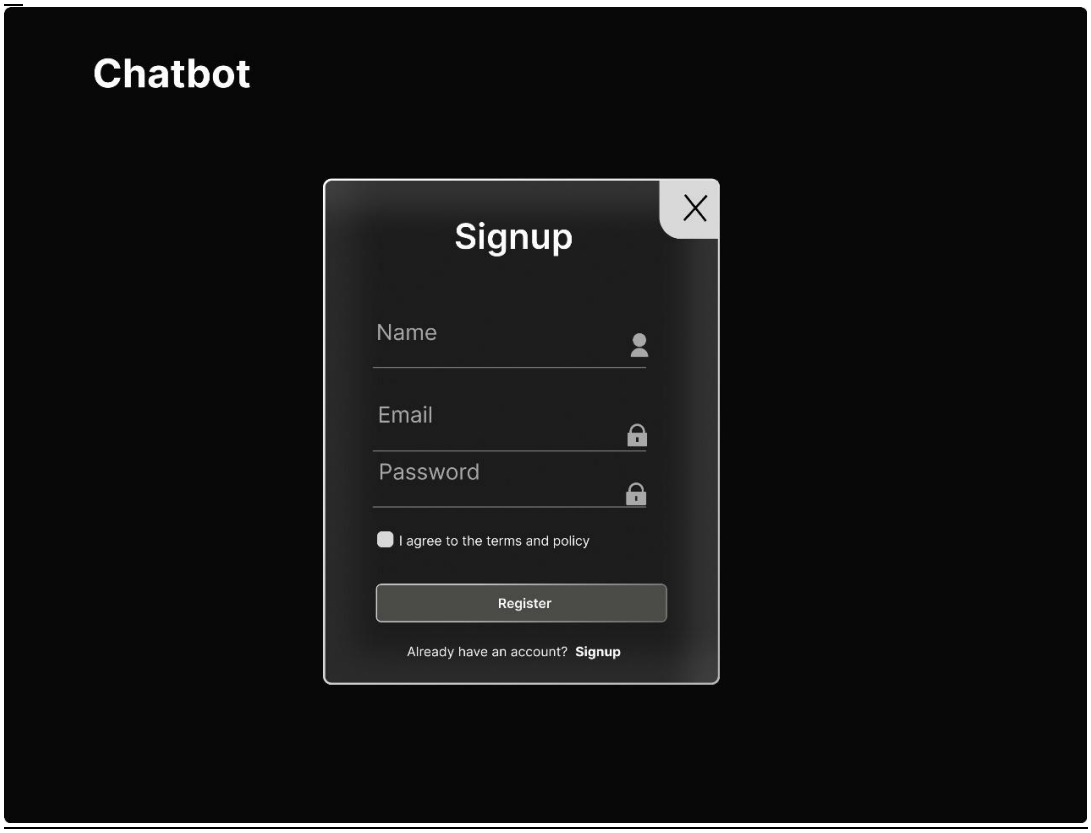


7. Database Design

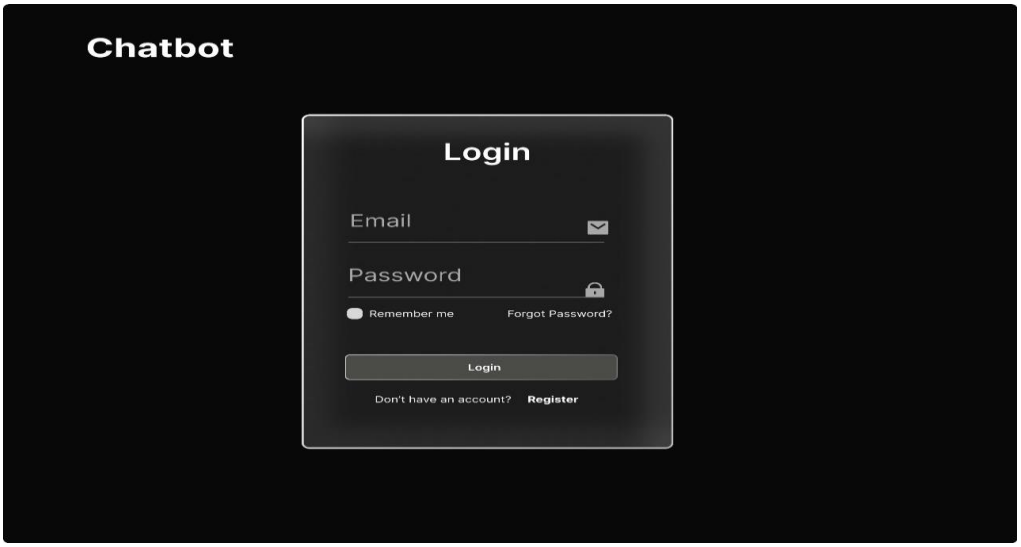


8. Interface Design

1



2.



3.