**Requirement Documentation**

Medical Chatbot MVP with RAG and Voice Interface (Digital Twin Planned)

1. Overview:

This project is a **Medical Domain Chatbot MVP** that utilizes a **Retrieval-Augmented Generation (RAG)** pipeline to answer user queries based on a trusted medical knowledge base (e.g., book or document). It supports **voice-based input** for accessibility and ease of use.  
While a **Digital Twin** component was originally part of the project plan, it has been deferred due to MVP constraints. A modular placeholder has been added to facilitate future integration.

1. Objectives:

 Build a RAG-based medical chatbot using **Mistral 7B via Ollama**.

 Provide users with a **voice input interface** through Streamlit.

 Support dynamic, context-aware responses using a local **vector database** (ChromaDB).

 **Defer Digital Twin** functionality to post-MVP stage due to prioritization of core features.

1. Key Features (MVP):

| **Feature** | **Description** |
| --- | --- |
| RAG Chatbot | Accurate, document-aware medical Q&A via LLM |
| Voice Input | Convert user speech to text for natural queries |
| Streamlit UI | Simple and clean frontend interface |
| Local Deployment | Fully offline and privacy-focused |

1. Digital Twin Exclusion:

The **Digital Twin** feature—intended to simulate virtual patient profiles for more personalized interaction—was not implemented in this MVP due to the following reasons:

* **MVP Prioritization**: Focus was on building a stable RAG + voice chatbot as the core value proposition.
* **Complexity**: Implementing personalized twin logic requires dynamic profile state management, simulation, and possible reasoning, which was outside the current timeline and resource allocation.
* **Compute Constraints**: Given the hardware demands of running a 7B model locally, efforts were directed toward performance optimization and system reliability.
* **Future Integration Ready**: The system is modular and includes placeholders (e.g., digital\_twin/patient\_profile.json) for seamless future expansion.

1. System Requirements:

| **Requirement** | **Description** |
| --- | --- |
| **Python Version** | >=3.10.12 |
| **RAM** | >=16GB |
| **Model** | mistral:7b via Ollama |
| **Vector Store** | ChromaDB |
| **Frontend** | Streamlit |
| **Voice Libraries** | SpeechRecognition, PyAudio |
| **(Optional) TTS** | pyttsx3 |

1. Dependencies:

llama-index==0.12.46

chromadb==1.0.15

llama-index-vector-stores-chroma==0.4.2

llama-index-embeddings-huggingface==0.5.5

llama-index-llms-ollama==0.6.2

sentence-transformers==5.0.0

pyttsx3==2.98

SpeechRecognition==3.14.3

pyaudio==0.2.14

streamlit==1.46.1

audio\_recorder\_streamlit==0.0.10

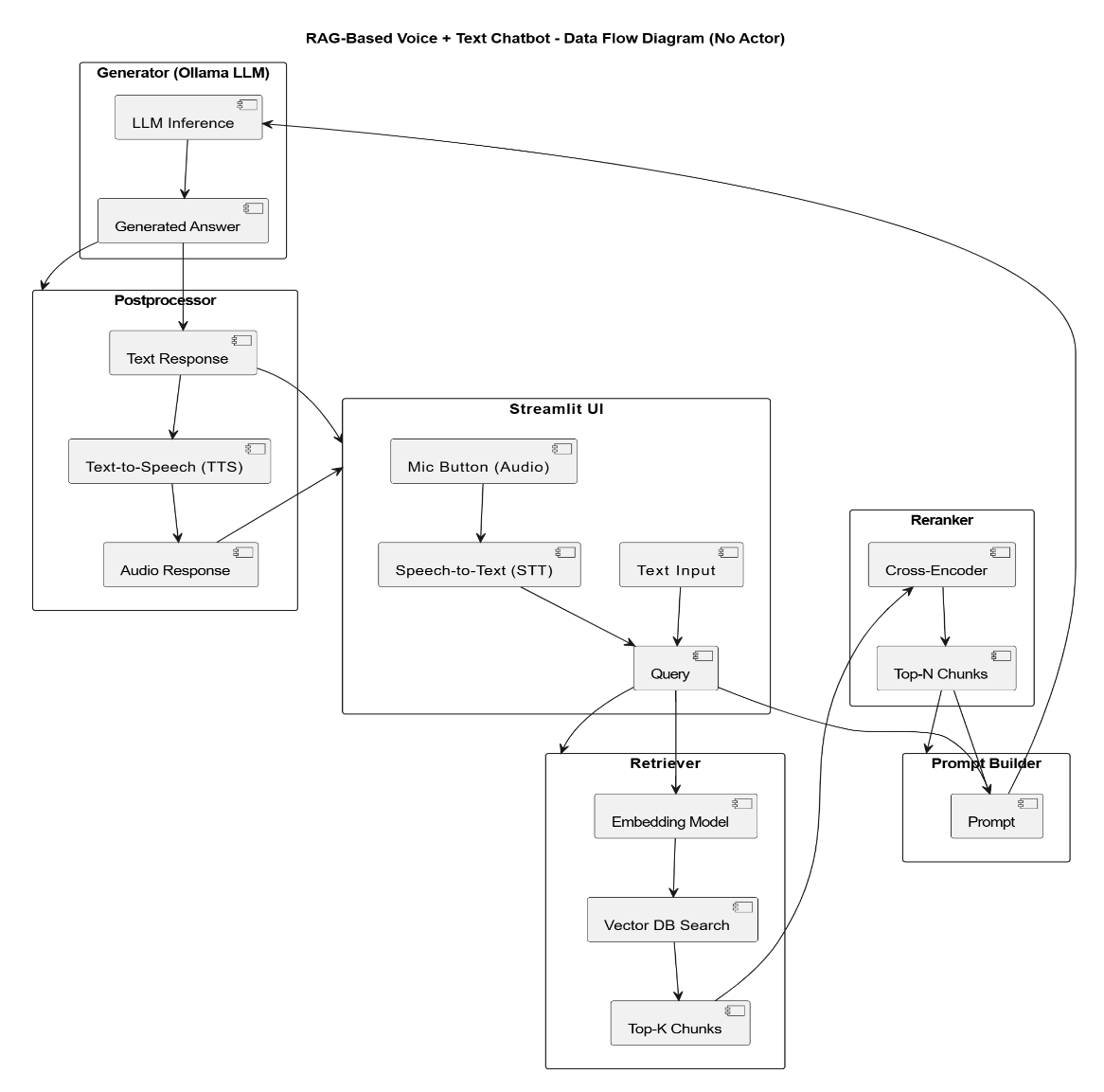
1. HLD

A diagram of a computer

AI-generated content may be incorrect.

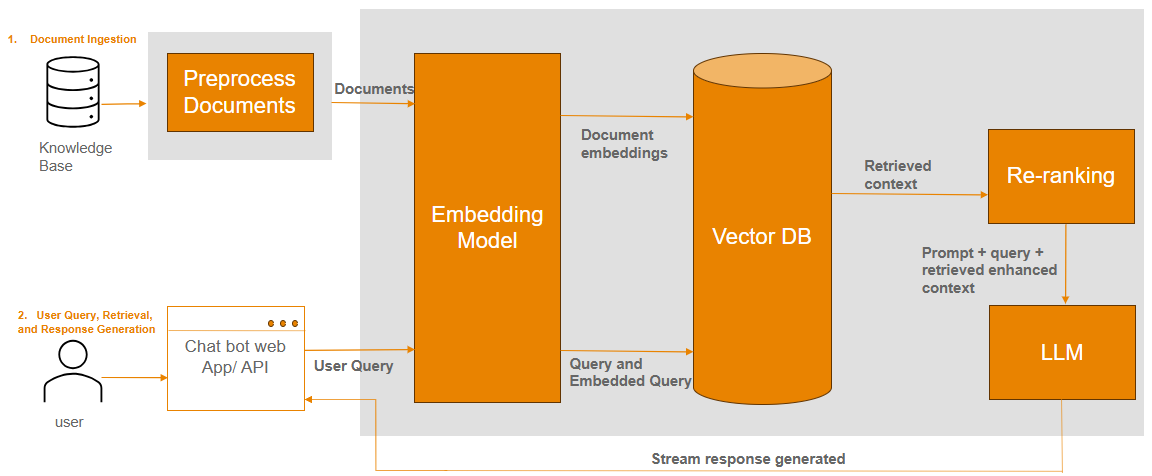
This diagram shows the high-level flow of how the **voice-based medical chatbot** works:

* The **user** interacts with the chatbot using the **Streamlit frontend**.
* If they use **voice**, it is first converted to **text** using a speech-to-text module.
* This text is sent to a **Vector Search** engine which queries the **Vector Database (ChromaDB)** for relevant medical information.
* The retrieved results are **re-ranked** to improve answer quality.
* A **prompt** is built using the re-ranked context and the user’s query.
* This prompt is then passed to the **Ollama LLM** (running mistral:7b), which generates the final response.
* The response is shown back in the **Streamlit frontend**.

1. Data Flow Diagram

This diagram breaks down the complete **data flow inside the chatbot system**, especially combining **voice and text input**:

* The **Streamlit UI** allows both **text typing** and **voice input** (mic button).
* **Voice input** is converted to text using **speech-to-text (STT)**.
* The final query is sent to the **Retriever** module:
  + First, an **embedding model** converts the query into vector format.
  + Then it searches **ChromaDB** for similar content and returns the **Top-K Chunks**.
* The **Reranker** (cross-encoder) refines those chunks and selects the best ones.
* A **Prompt Builder** forms the complete prompt using the top chunks and user query.
* This prompt is sent to the **Generator (Ollama LLM)** for inference.
* The generated answer is optionally passed through a **Text-to-Speech (TTS)** module to give an **audio response**.

1. Architecture Diagram

This diagram shows the **full architecture** of the chatbot system from document processing to user interaction:

**Phase 1: Document Ingestion**

* A **medical knowledge base (book or documents)** is preprocessed.
* The documents are passed through an **embedding model** to convert them into vector form.
* These **document embeddings** are stored in a **Vector DB (ChromaDB)**.

**Phase 2: User Query & Answer Generation**

* The **user** interacts with the chatbot through a **web app or API**.
* Their query is embedded and searched inside the **Vector DB** to find relevant information.
* The **retrieved context** is passed through a **Re-ranking** module to improve accuracy.
* Finally, the **LLM (via Ollama)** uses the enhanced prompt to generate a response, which is streamed back to the user.

**Conclusion:**

This MVP demonstrates a strong foundation for a privacy-focused, voice-enabled RAG chatbot in the medical domain. While the digital twin, caching mechanism, and account management features were not fully implemented in the current version, I have carefully planned and modularized the system to allow for seamless integration in the next phase. My commitment to building a scalable, intelligent assistant remains firm, and I am dedicated to enhancing the chatbot with personalized virtual twin logic, efficient caching for faster response, and user account handling to support real-world deployment.