Software Requirements Specification

for

<MediBot>

Version 1.0 approved

Prepared by <Awanish Gupta>

Prepared by <Adarsh Raj >

<Indian Institute of Information Technology Allahabad>

<15th April, 2025>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the overall software requirements for the MediAsh project. MediAsh is a medical chatbot powered by AI offering questioning-answering capability from a fixed set of knowledge of the chatbot. It is developed using Streamlit, Langchain, FAISS, and transformer-based LLMs. This SRS defines the backend document processing pipeline, vector-based retrieval system, and chat-based frontend.

1.2 Document Conventions

- Requirements are labeled as REQ-# (e.g. REQ-1, REQ-2).
- Major sections are numbered.
- Appendices contain supporting models and definitions.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers** who implement and maintain the system.
- **Testers** creating test cases.
- **End-users** as technical reference.

The reading flow of the document is: Start with Overall Description and System Features for a high-level understanding, followed by technical sections and appendices.

1.4 Product Scope

MediBot is intended to support users, particularly medical students and researchers to give answers from academic PDF material through the means of a chatbot interface. The architecture consists of document loading, chunking, embedding , FAISS storage, and generation through an LLM.

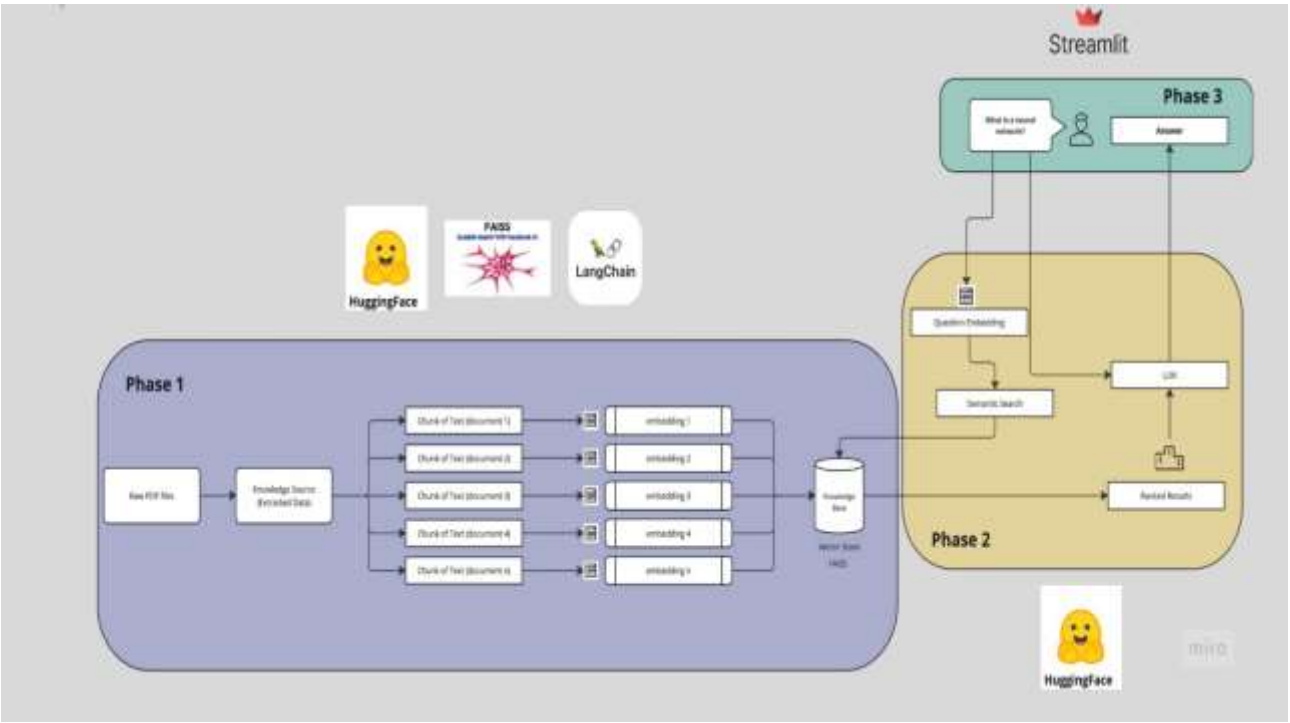
1.5 References

- Langchain Documentation
- Hugging Face API Documentation
- Streamlit Library
- FAISS (Facebook AI similarity Search)

2. Overall Description

2.1 Product Perspective

MediBot is a standalone chatbot platform that translates documents into vector embeddings and provides natural language Q&A via a Streamlit-based frontend. It is to be developed with open-source tools and for information and educational purposes.



2.2 Product Functions

- Upload and process academic PDFs
- Split text into chunks and generate embeddings
- Store vectors in FAISS for semantic search
- Accept user queries via chat interface
- Retrieve top-k relevant chunks
- Generate responses using an LLM
- Display results via Streamlit UI

2.3 User Classes and Characteristics

- **General Users:** Ask health-related questions
- **Medical Students:** Academic and research queries
- **Developers/Admins:** Maintain documents, models, and infrastructure

2.4 Operating Environment

- OS: Windows/Linux
- Tools: Python 3.11+, Streamlit, Hugging Face API, FAISS
- Requires internet for model API access

2.5 Design and Implementation Constraints

- Hugging Face API tokens and quota limitations
- PDF format required for document ingestion
- No support for live model hosting—must use Hugging Face Hub

2.6 User Documentation

- A README.md file explaining setup
- Streamlit-based UI guidance and tips
- Potential tutorial videos or user guide (TBD)

2.7 Assumptions and Dependencies

- The user has an active Hugging Face token
- Documents are already vectorized
- Internet connectivity is available

3. External Interface Requirements

3.1 User Interfaces

- Chat interface built in Streamlit
- Text input field and real-time markdown responses
- Clear labels and basic instructions

3.2 Hardware Interfaces

- No special hardware interfaces required
- Basic computing with optional GPU acceleration for local embeddings

3.3 Software Interfaces

- Hugging Face Hub
- LangChain and related chains/utilities
- FAISS (local vector database)

3.4 Communications Interfaces

- HTTPS requests to Hugging Face Hub
- Secure handling of API tokens

4. System Features

4.1 Chatbot Querying

4.1.1 Description and Priority

High priority feature for all end-users

4.1.2 Stimulus/Response Sequences

- User submits question
- System fetches relevant context
- LLM generates and displays response

4.1.3 Functional Requirements

REQ-1: Process and store document chunks as embeddings
REQ-2: Accept user input and pass it to the retrieval chain
REQ-3: Retrieve top-k matching vectors
REQ-4: Generate a response from LLM and return to UI
REQ-5: Maintain chat history for the session

4.2 Document Preprocessing and Embedding generation

4.2.1 Description and Priority

This is a core backend functionality, essential for ensuring high-quality information retrieval. This feature is of high priority because all subsequent chatbot responses depend on the successful creation of accurate document embeddings.

4.2.2 Stimulus/Response Sequences

- Admin or system initiates document
- System splits documents into smaller chunks.
- System generates vector embeddings for each chunk using a HuggingFace embedding model.
- System stores the embeddings in the FAISS vector database.

4.2.3 Functional Requirements

REQ-1: Accept .txt or .pdf documents for ingestion
REQ-2: Divide documents into manageable text chunks (e.g. using token count).
REQ-3: Convert each chunk into a high-dimensional embedding using a pre-defined HuggingFace model like sentence-tranformers/ all-MiniLM-L6-v2

REQ-4: Persist the generated embeddings into the FAISS vector store under a defined path.
REQ-5: Support re-indexing or appending new documents to the existing vector store.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>