Software Requirements Specification

for

<MediBot>

Version 1.0 approved

Prepared by <Awanish Gupta>

Prepared by <Adarsh Raj >

<Indian Institute of Information Technology Allahabad>

<4thth May, 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.

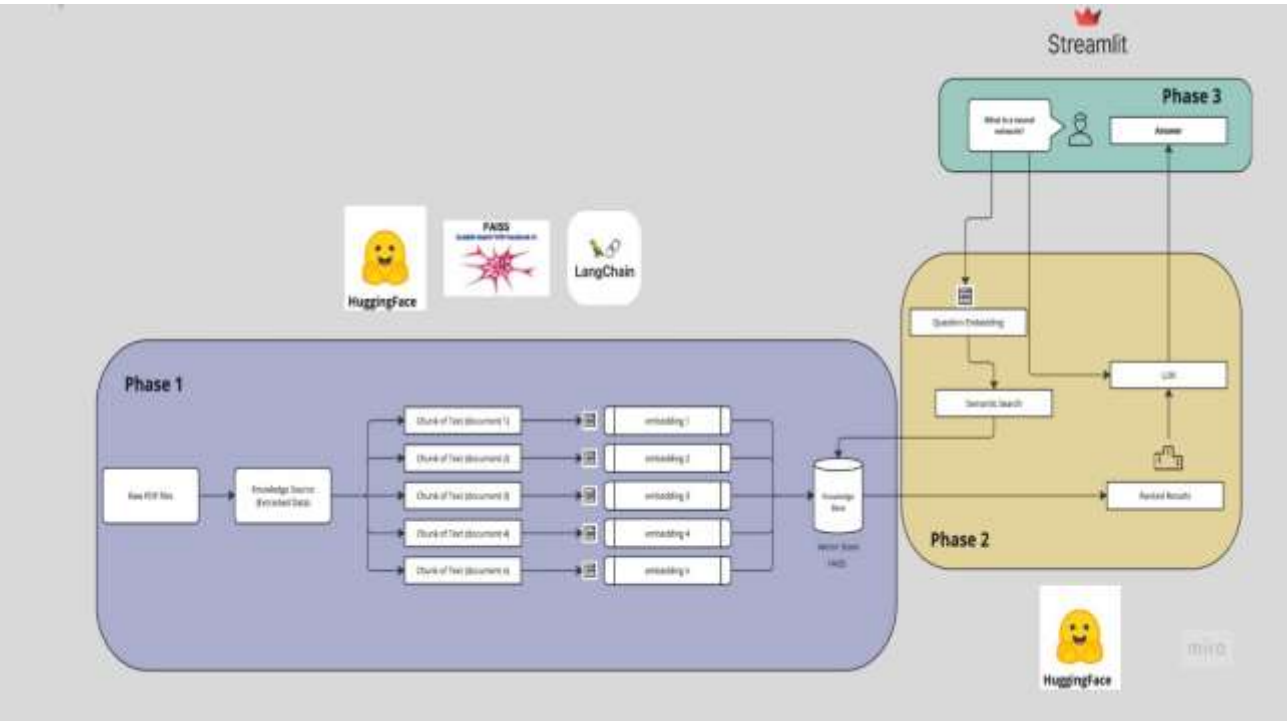
1.5 References

- 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.

4.3 Vector-Based Document Retrieval

4.3.1 Description and Priority

This feature performs the core logic behind finding the most relevant document content for any user query. It is a critical component, since accurate document retrieval directly affects the quality of chatbot responses. This is also considered a high-priority backend feature.

4.3.2 Stimulus/Response Sequences

- User sends a question to the chatbot
- System embeds the question into a vector using the same embedding model as the document chunks.
- System queries the FAISS vector store using similarity search.
- System retrieves top-k relevant document chunks as context.

4.3.3 Functional Requirements

REQ-1: Convert user query into vector representation using the same embedding model used during document preprocessing
REQ-2: Perform a top-k nearest-neighbour search in the FAISS index.
REQ-3: Return a list of k relevant document chunks sorted by similarity score.
REQ-4: Provide retrieved content to the prompt template as context.
REQ-5:Allow adjustment of search hyperparameters like k (number of results).

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- System should respond in <5 seconds under normal conditions
- Vector search latency must remain low (near real-time)

5.2 Safety Requirements

- Chatbot will not provide real-time medical advice
- Clearly disclaim that it's for academic/informational use only

5.3 Security Requirements

- API tokens should never be hardcoded
- Token must be stored securely (in .env or secret manager)

5.4 Software Quality Attributes

- Usability: Easy chat interface
- Maintainability: Modular Python files
- Portability: Compatible with Windows/Linux
- Extensibility: Can swap models or document sources

5.5 Business Rules

- Only pre-approved academic material may be used
- The chatbot must not hallucinate or fabricate beyond context

6. Other Requirements

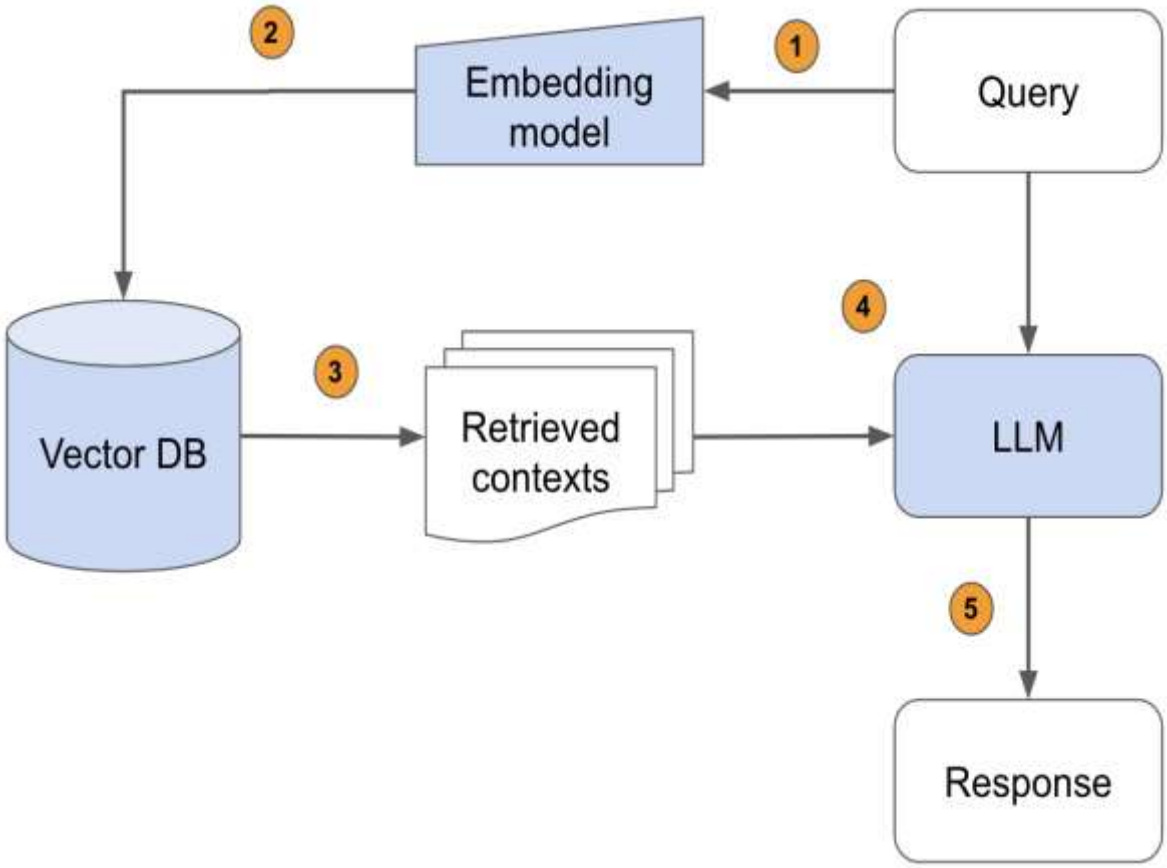
- All PDF documents must be converted to embeddings before chat interaction
- LLM fallback must be implemented if Hugging Face is unavailable

Appendix A: Glossary

- LLM: Large Language Model
- FAISS: Facebook AI Similarity Search
- Embedding: Vector representation of text
- Streamlit: Python library for interactive apps
- LangChain: Framework for LLM Pipenlies

Appendix B: Analysis Models

- Chat Workflow Diagram



Appendix C: To Be Determined List

- TBD-1: Add user authentication
- TBD-2: Implement streaming output
- TBD-3: Support for uploading documents via UI