AI-Powered Micro SaaS RAG-Based Document Querying System

Software Requirements Specification (SRS)

Supervisor

Dr. Rana Nazeer Ahmad

Submitted By

Muhammad Huzaifa

F22BINFT1M01163

Dept. of Information Technology

Islamia University of Bahawalpur

Faculty of Computing

Date: 05 Dec 2025

Meeting Details

Sr No	Details	Date	Supervisor Signature
1.	Initial Proposal		
2.	SRS Review		
3.	Final Submission		

Abstract

This document constitutes a formal Software Requirements Specification (SRS) for an **AI-Powered Micro SaaS RAG-Based Document Querying System**. The primary objective of this system is to architect a scalable knowledge retrieval solution that addresses the limitations of traditional manual document search. By integrating Natural Language Processing (NLP), vector embeddings, and a Retrieval-Augmented Generation (RAG) pipeline, the platform is designed to allow users to upload diverse document formats and query them using natural language.

This SRS provides a comprehensive technical and functional blueprint, delineating the system's design philosophy, operational boundaries, and detailed requirements. It further specifies distinct user archetypes, elaborates on core backend functionalities (such as text chunking and embedding generation), defines the user interface interaction model, proposes a high-level system architecture, and provides structured use-case analyses to illustrate system behavior.

1. Introduction

1.1 Problem Statement and Motivation

Large organizations, academic institutes, and researchers store vast amounts of data across various document formats. The contemporary digital ecosystem relies heavily on manual reading and keyword-based search (Ctrl+F), which is often inefficient when dealing with large corpora of text. Users are required to manually sift through pages to find contextually relevant information. This traditional method lacks semantic understanding, making it difficult to extract precise answers from complex documents quickly.

1.2 Proposed Solution

The proposed system presents a fundamental architectural shift by utilizing a Retrieval-Augmented Generation (RAG) pipeline as its intelligent backbone. The core innovation is the deployment of a Micro SaaS platform where users can upload documents (PDF, DOCX, TXT, CSV), which are autonomously processed into vector embeddings.

These embeddings are stored in a vector database (ChromaDB), allowing for semantic search rather than simple keyword matching. When a user asks a question, the system retrieves the most contextually relevant chunks and utilizes a Large Language Model (LLM) to generate a precise, natural language answer. This model effectively automates knowledge extraction, saving time and increasing accuracy.

1.3 Purpose

The principal purpose of this development initiative is to research, design, and implement a secure, scalable, and fully automated document querying application. This purpose is operationalized through the following specific objectives:

1. To eliminate the need for manual document scanning by automating text extraction and synthesis.
2. To ensure accurate information retrieval by leveraging vector embeddings and semantic search technology.
3. To provide a user-friendly Micro SaaS interface that abstracts the complexities of AI and NLP, making advanced document interrogation viable for non-technical users.
4. To create a secure multi-user environment where document isolation and data privacy are enforced.

1.4 Scope

1.4.1 In-Scope Deliverables

The project scope is bounded to the development and deployment of the following core components:

1. **Authentication Module:** Secure Sign Up and Login functionality using JWT and encrypted storage.
2. **Document Processing Pipeline:** Automated ingestion of PDF, DOCX, TXT, and CSV files, including text extraction and chunking.
3. **Vector Embedding Engine:** Integration with SentenceTransformers to convert text chunks into vector representations.
4. **RAG Query Interface:** A chat-like interface where users can ask natural language questions and receive AI-generated answers based strictly on uploaded content.
5. **Vector Database Integration:** Implementation of ChromaDB for efficient storage and retrieval of high-dimensional vectors.
6. **Admin Dashboard:** A panel for user management and system monitoring.

1.4.2 Out-of-Scope Items

Explicitly excluded from the current project phase are:

1. **OCR Scanning:** Processing of non-selectable, image-based PDFs or handwritten documents.
2. **Real-time Collaboration:** Simultaneous editing or querying of documents by multiple users in the same session.
3. **Mobile Native Apps:** Development of iOS or Android applications (the system is a responsive web application).

1.5 Product Perspective

The system is architected as a cloud-hosted microservice. It operates within a modern client-server paradigm.

- **Frontend:** Built with React.js or Streamlit, serving as the presentation and interaction layer.
- **Backend:** A FastAPI service that handles business logic, API routing, and AI processing.
- **Database:** PostgreSQL for user data and ChromaDB for vector storage.
- **AI Layer:** Integrates with OpenAI or open-source LLMs for answer generation.

1.6 User Characteristics

The system caters to three primary user archetypes with distinct goals:

User Type	Description	Technical Prerequisite
Admin	Manages the platform, approves user accounts, and oversees system health.	Basic understanding of web administration dashboards.
Registered User	The primary user uploading documents and asking questions. Goal is to extract information efficiently.	No technical skills required; must possess valid login credentials.

Developer	Responsible for backend maintenance, API integration, and model updates.	collaborative coding skills (Python, Docker, API management).

1.7 Analysis of Comparable Systems

A comparative analysis highlights the differentiating value proposition of the proposed system:

Platform	**Key Features**	**Shortcomings**
ChatPDF	Simple PDF upload and chat interface.	Limited customization; often restricts file size/count on free tiers.
Humata AI	fast processing, good for research papers.	Proprietary and expensive for casual users; closed ecosystem.
LangChain (Raw)	Framework for building LLM apps.	Not a user-facing product; requires coding knowledge to utilize.
Proposed System	Micro SaaS architecture, custom RAG pipeline, multi-format support.	Initial reliance on external LLM APIs (latency dependent).

1.8 Proposed Technology Stack

The selection of technologies is guided by the requirements for speed, scalability, and AI support.

- **Language:** Python (3.10+)
- **Backend Framework:** FastAPI

- **Frontend:** Streamlit / React.js
- **AI/NLP:** SentenceTransformers, LangChain
- **Database:** ChromaDB (Vector), PostgreSQL (Relational)
- **Containerization:** Docker
- **Authentication:** JWT (JSON Web Tokens)

2. System Requirements

The system is designed to enable a complete, self-contained document analysis workflow. Users can register, upload files, and immediately begin querying them. A fundamental tenet is that the answer generation is grounded in the provided document context (RAG), minimizing hallucinations common in generic AI models.

2.1 Functional Requirements

Functional requirements specify what the system must do. They are enumerated below in a structured format.

FR001: User Sign Up

- **Name:** Sign Up
- **Purpose:** Allows new users to register for the platform to access document querying features.
- **Actors:** Unregistered User, Admin
- **Preconditions:** User is not currently logged in.
- **Inputs:** Name, Email, Password (8+ chars, includes number), Phone, Display Picture (Optional).
- **Process:** User submits details. The system validates input formats. The account is created with a status of "Pending" until Admin Approval.
- **Outputs:** User account created; confirmation message displayed awaiting approval.

FR002: User Login

- **Name:** Login
- **Purpose:** To authenticate users and generate a session token.
- **Actors:** Registered User
- **Preconditions:** Account exists and is approved.
- **Inputs:** Email, Password.
- **Process:** Backend verifies hash against database. Issues JWT upon success.
- **Outputs:** Redirect to Dashboard.

FR003: Document Upload & Ingestion

- **Name:** Upload Document
- **Purpose:** To allow users to add knowledge sources to the system.
- **Actors:** Registered User

- **Preconditions:** User is authenticated. File is in supported format (PDF, DOCX, CSV, TXT).
- **Inputs:** File selection via UI.
- **Process:** System accepts file, extracts text content (Text Extraction), splits text into manageable chunks (Chunking), and passes chunks to the embedding model.
- **Outputs:** Success notification; Document status updated to "Processed".

FR004: Embedding Generation & Storage

- **Name:** Generate Embeddings
- **Purpose:** To convert text into machine-understandable vectors.
- **Actors:** System (Automated)
- **Inputs:** Text Chunks from FR003.
- **Process:** The SentenceTransformer model processes text chunks. Resulting vectors are stored in ChromaDB linked to the User ID.
- **Outputs:** Vectors persisted in the database.

FR005: Semantic Retrieval & Answer Generation (RAG)

- **Name:** Query Document
- **Purpose:** To allow users to ask natural language questions and receive context-aware answers.
- **Actors:** Registered User
- **Preconditions:** At least one document is uploaded and processed.
- **Inputs:** Natural language query (string).
- **Process:**
    1. The system converts a query into a vector.
    2. The system performs similarity search in ChromaDB to find relevant chunks.
    3. System sends the query + retrieved chunks to the LLM.
    4. LLM generates a response based *only* on the chunks.
- **Outputs:** Textual answer displayed to the user; references to source chunks (optional).

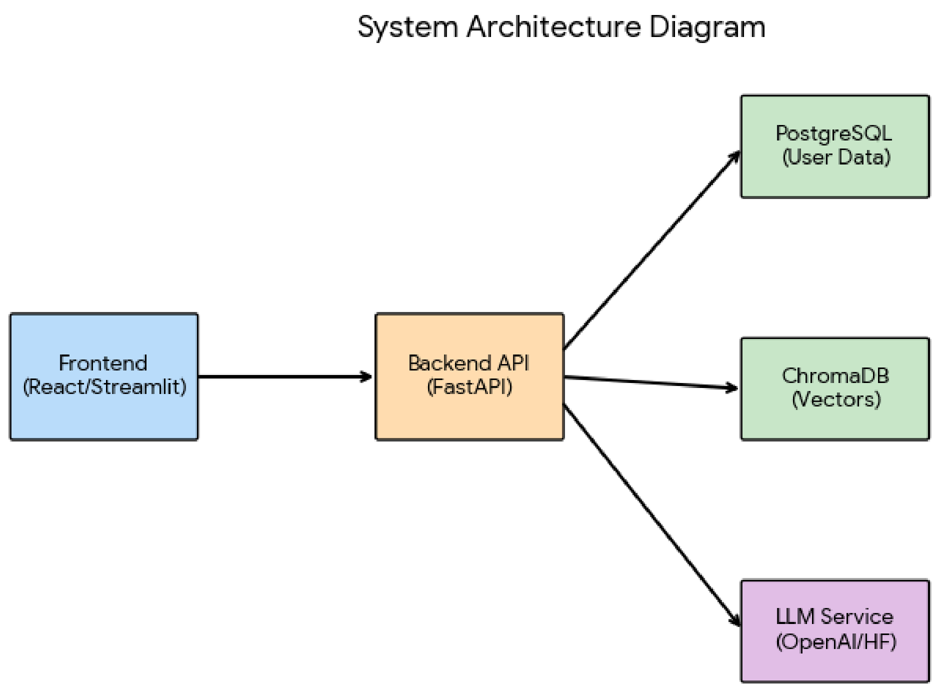
FR006: Admin Dashboard

- **Name:** Manage System
- **Purpose:** To allow the admin to view and manage users.
- **Actors:** Admin
- **Inputs:** Approval/Rejection actions.
- **Process:** Admin views list of pending users and toggles active status.
- **Outputs:** User status updated in PostgreSQL.

2.2 Non-Functional Requirements

- **Performance:** Query response time should be under 4 seconds for standard complexity queries.

- **Scalability:** The system must support a multi-user environment where distinct users can process documents concurrently.
- **Security:** All user passwords must be hashed. Data in transit must be encrypted. Authentication managed via JWT.
- **Reliability:** The system should aim for 99% uptime. Document processing failures should result in clear error messages, not system crashes.
- **Usability:** The interface must be clean, intuitive, and accessible (React/Streamlit), ensuring users know how to upload and ask questions without training.
- **Privacy:** Strict document isolation; User A must never be able to query documents uploaded by User B.



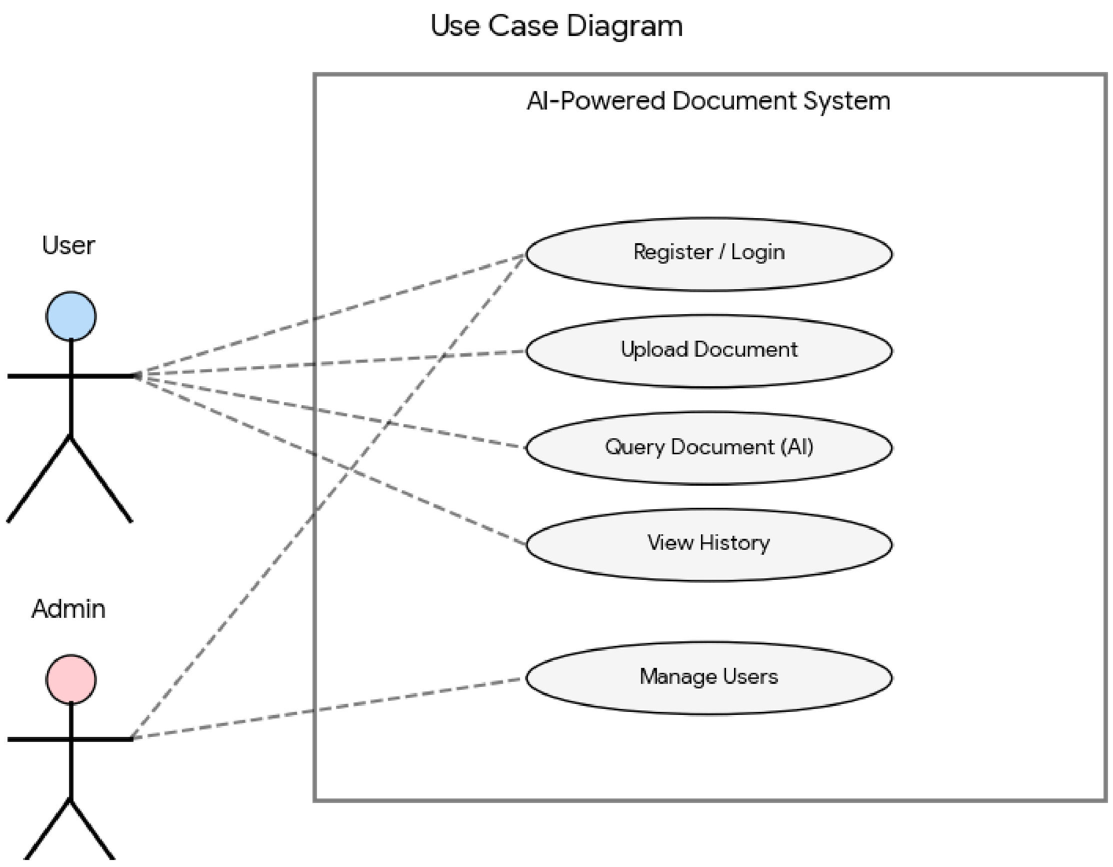
3. Use Cases and Process Flows

This section provides detailed narratives of how actors interact with the system to achieve specific goals.

3.1 Use Case Model

- **Primary Actors:** Registered User, Admin.
- **System Boundaries:** Encompasses the Streamlit/React frontend, FastAPI backend, LLM API, and Vector Database.
- **Flow Overview:** User uploads doc $\rightarrow$ system extracts text $\rightarrow$ chunks text $\rightarrow$ embeds chunks $\rightarrow$ stores in ChromaDB

$\rightarrow$ user asks query $\rightarrow$ system retrieves relevant chunks $\rightarrow$ LLM generates answer.



3.2 Detailed Use Case Specification

Use Case ID: UC001

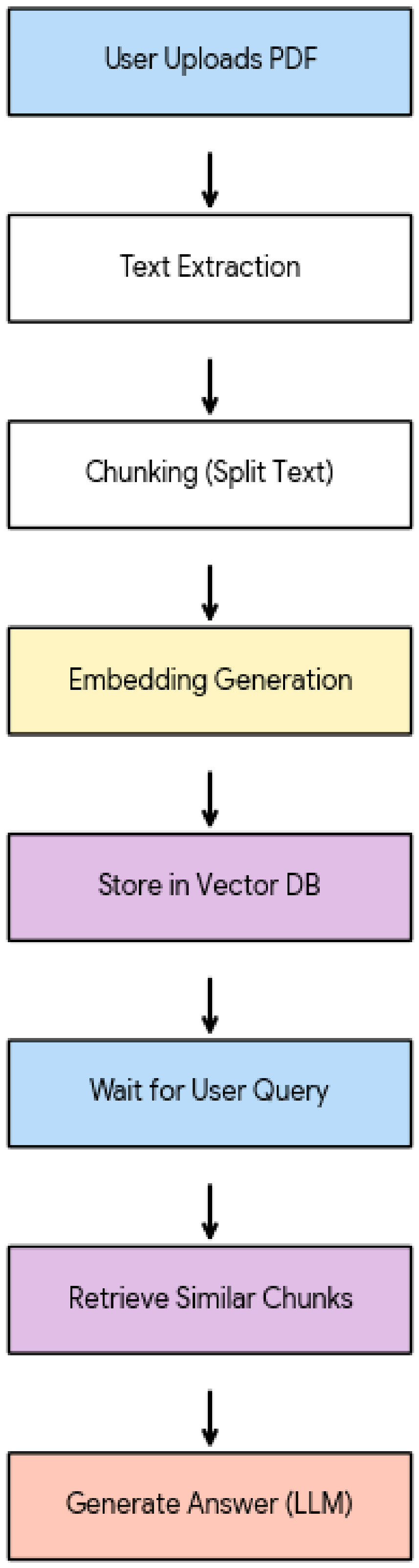
- **Use Case Name:** Sign Up (Registration)
- **Primary Actor:** Unregistered User
- **Secondary Actor:** Admin
- **Requirement:** FR001
- **Preconditions:** User has a valid email and phone number.
- **Main Success Scenario (Basic Flow):**
    - The user navigates to the Registration page.
    - The user enters Name, Email, Password, Phone.
    - System validates password complexity (8 chars + number).
    - The system creates a database entry with "Pending" status.
    - The system informs the user to wait for Admin approval.
    - Admin logs in and approves the request.
- **Alternative Flows:**

    - *Admin Manually Creates User:* Admin uses the dashboard to provision an account directly.
- **Exceptions:**
    - *Invalid Input:* System displays error messages next to relevant fields.
    - *Email Already Exists:* System prevents duplicate registration.

Use Case ID: UC002

- **Use Case Name:** Query Document (RAG Pipeline)
- **Primary Actor:** Registered User
- **Requirement:** FR007, FR008
- **Preconditions:** User is logged in; Document is successfully processed.
- **Main Success Scenario:**
    - The user selects a document or context scope.
    - User types a question (e.g., "What is the summary of section 2?").
    - The system converts questions to vectors.
    - System retrieves top 3-5 matching text chunks from ChromaDB.
    - The system prompts the LLM with context and questions.
    - The system displays the generated answer.
- **Exceptions:**
    - *No Context Found:* If the vector search returns low similarity scores, the system replies, "I cannot find the answer in the provided document."

RAG Pipeline Flowchart



4. References

1. LangChain Documentation: Introduction to RAG pipelines.
2. ChromaDB Documentation: Vector database implementation guide.
3. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.
4. OpenAI API Documentation / HuggingFace Models.
5. FastAPI Documentation.