Software Design Document (SDD)

AI-Powered Micro SaaS RAG-Based Document Querying System

Submitted By: *Muhammad Huzaifa (F22BINFT1M01163)*

Submitted To: *Dr. Rana Nazeer Ahmad*

Dept. of Information Technology

Islamia University of Bahawalpur

Faculty of Computing

Date: 09 Jan 2026

This Software Design Description (SDD) provides a comprehensive theoretical blueprint for the **AI-Powered Micro SaaS RAG-Based Document Querying System**. It translates the requirements established in the SRS into a detailed technical design, focusing on the architectural framework, data structures, and user interface paradigms.

1. Application Architecture

The system is architected using a **Cloud-Hosted Microservice** model, following a modern **Client-Server paradigm**. To ensure modularity and separation of concerns, the application follows an **MVC (Model-View-Controller)** inspired pattern, where the backend FastAPI service acts as the controller, managing the flow of data between the storage layers and the user interface.

1.1 Architectural Components

1.1.1 List of Proposed Models

The following models represent the data structures utilized by the system to maintain state and process document knowledge:

- **User Model:** Stores identity attributes including name, email, and encrypted password hashes.
- **Document Model:** Manages metadata for uploaded files, including file paths, formats (PDF, DOCX, TXT, CSV), and ingestion status.
- **Text Chunk Model:** Represents the segmented parts of documents created during the chunking process.
- **Vector Embedding Model:** Represents the high-dimensional numerical values stored in the vector database.
- **System Health Model:** Monitors the operational status and uptime of the RAG pipeline.

1.1.2 List of Proposed Views

The Views represent the presentation layer, built using **React.js or Streamlit**:

- **Authentication View:** Interfaces for user registration and secure login.
- **User Dashboard:** A centralized hub for managing uploaded documents and viewing history.
- **Document Interrogation View:** A chat-based interface for natural language querying.
- **Admin Management Panel:** A specialized view for managing user accounts and system status.

1.2 Controller Logic and Functionality

The controllers are implemented via **FastAPI**, serving as the intelligent routing layer that handles business logic.

- **Authentication Controller:**

    - `register_user()`: Validates input formats and creates "Pending" account entries in the relational database.
    - `authenticate_session()`: Verifies password hashes and issues **JSON Web Tokens (JWT)** for session management.
- **Document Ingestion Controller:**
    - `upload_handler()`: Receives files and triggers the extraction pipeline.
    - `process_text()`: Coordinates the "Chunking" of text into manageable segments for optimal LLM context.
- **RAG Query Controller:**
    - `vector_search()`: Executes similarity searches against **ChromaDB** to find contextually relevant text chunks.
    - `generate_response()`: Interfaces with the LLM to synthesize an answer based strictly on retrieved context.
- **Administrative Controller:**
    - `update_user_status()`: Allows administrators to approve or reject pending registrations.

1.3 External APIs and Libraries

The system relies on a specialized suite of external integrations to facilitate the RAG pipeline:

- **Sentence Transformers:** Utilized as the primary library for converting text chunks into **Vector Embeddings**.
- **LangChain:** Acts as the orchestration framework, linking the document loaders, vector stores, and LLM prompts.
- **OpenAI/HuggingFace APIs:** Provides access to high-performance **Large Language Models (LLMs)** for generating human-like answers.
- **ChromaDB:** A specialized vector database used for the storage and high-speed retrieval of high-dimensional embeddings.

1.4 UML and System Diagrams

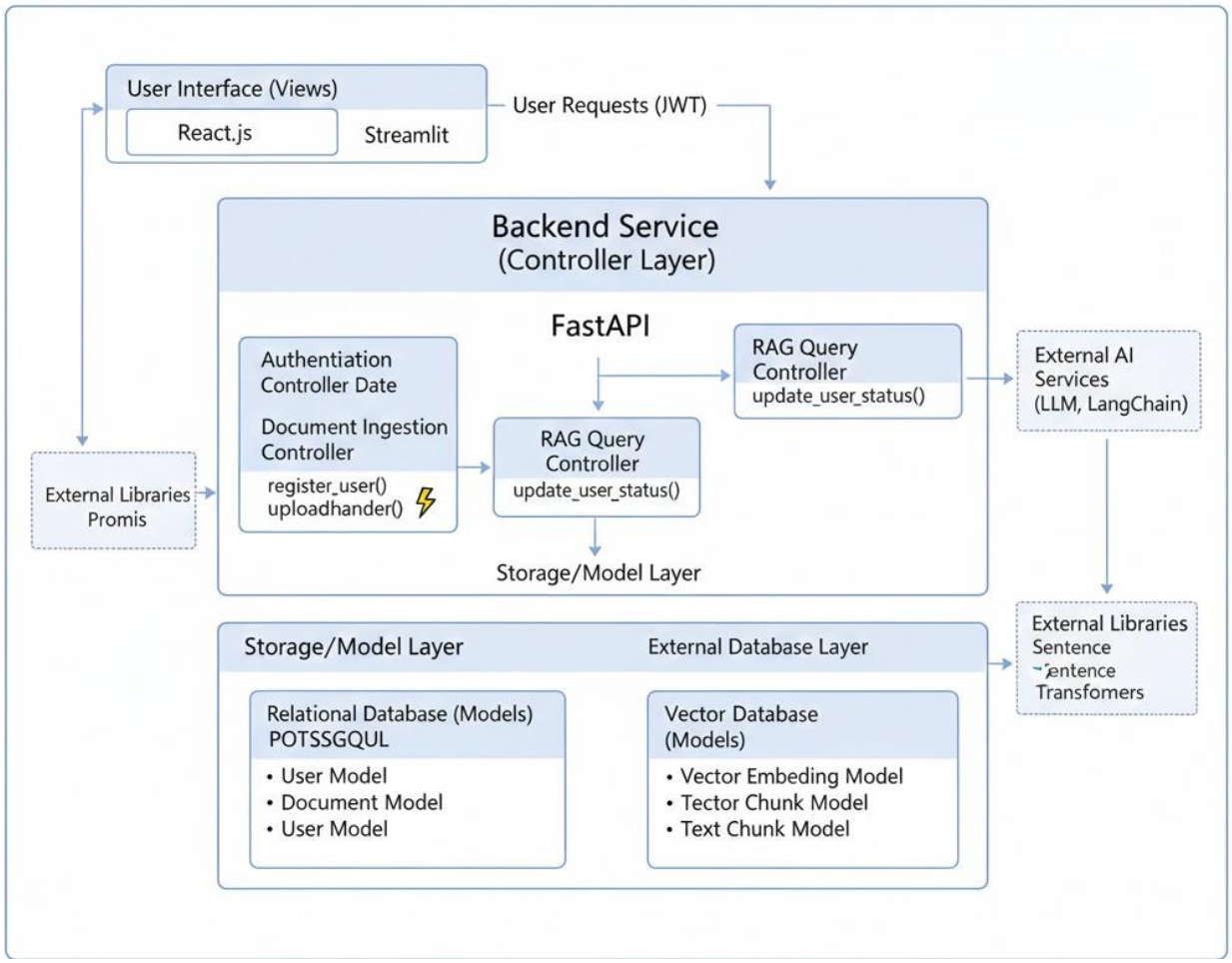
The system interaction flow begins with a user query, which is converted into a vector. This vector is compared against stored document embeddings in ChromaDB to retrieve the most similar chunks. Finally, the LLM processes these chunks to generate a response.

1.5 Coding Conventions

To maintain professional software standards, the following conventions are strictly followed:

- **Modularization:** Business logic (LLM processing) is isolated from routing logic (FastAPI endpoints).
- **Asynchronous Processing:** Long-running tasks like document embedding are handled asynchronously to prevent UI freezing.

- **Security Standards:** Implementation of **JWT** for stateless authentication and password hashing for data at rest.



2. Data Model Schema

The data architecture is dual-layered, utilizing **PostgreSQL** for structured relational data and **ChromaDB** for unstructured semantic data.

2.1 Entity Descriptions and Properties

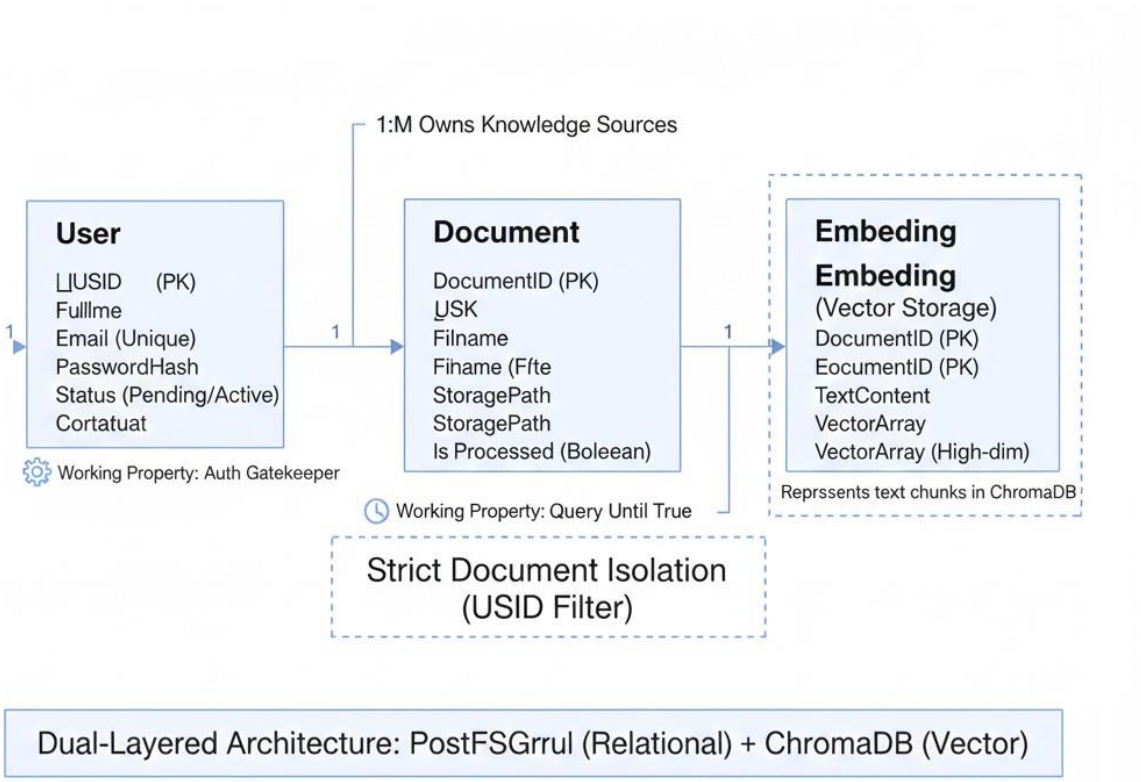
- **Entity: User**
  - **Properties:** UserID (Primary Key), Full Name, Email (Unique), Password Hash, Status (Pending/Active/Disabled), CreatedAt.
  - **Working Property:** The "Status" property acts as a gatekeeper for the `authenticate_session()` function.
- **Entity: Document**
  - **Properties:** DocumentID (Primary Key), UserID (Foreign Key), FileName, FileType, S3Path/StoragePath, IsProcessed (Boolean).

    - **Working Property:** "IsProcessed" ensures that a document is not queried until the embedding generation is complete.
- **Entity: Embedding (Vector Storage)**
    - **Properties:** EmbeddingID, DocumentID (Foreign Key), TextContent, VectorArray (High-dimensional).
    - **Relationship:** Each document is associated with multiple embedding entries (1:M relationship) due to the chunking process.

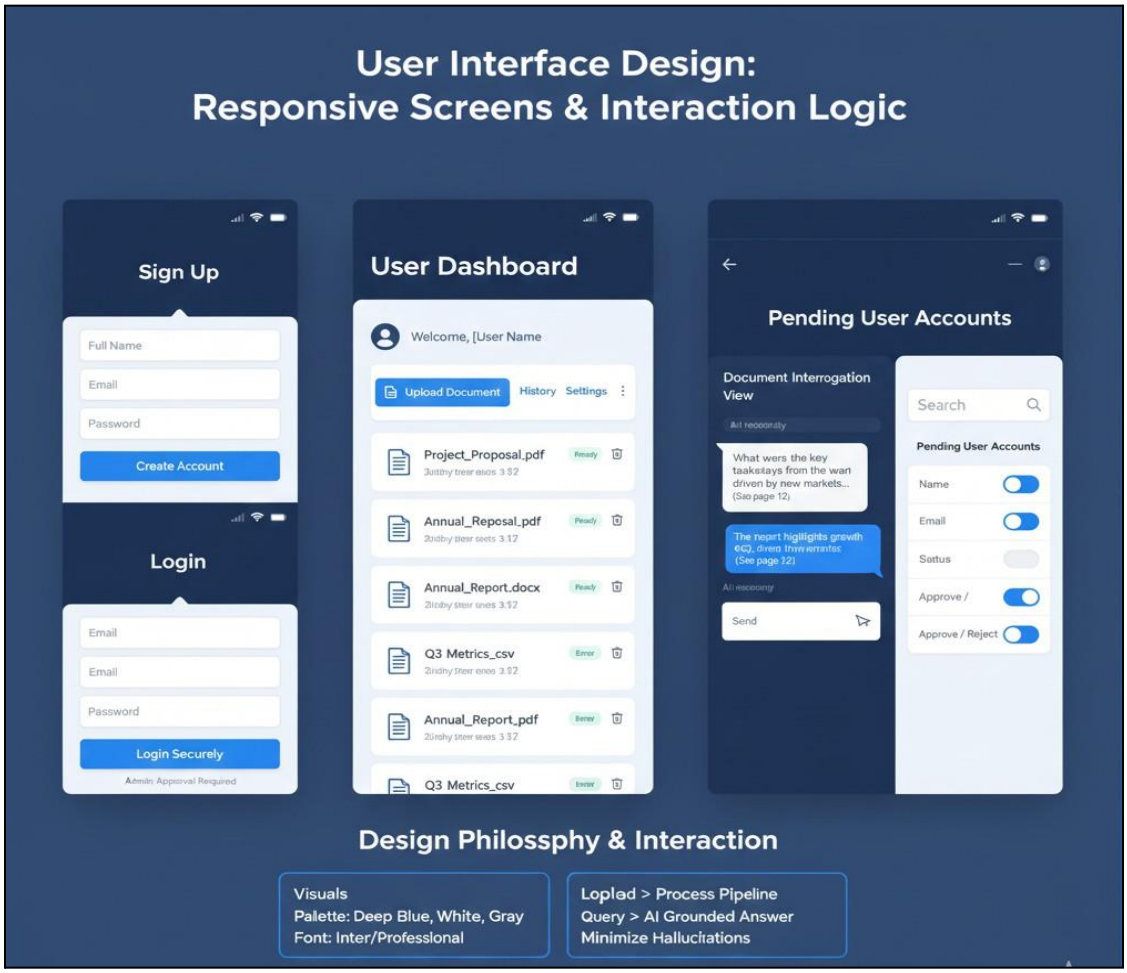
2.2 Entity Relationship (ER) Diagram

The relationship between entities is defined by strict **Document Isolation**.

- **User to Document:** A one-to-many (1:M) relationship where one user can own multiple knowledge sources.
- **Document to Embedding:** A one-to-many (1:M) relationship where a single file is decomposed into numerous vector-represented chunks.



3. User Interface



The design philosophy prioritizes **usability and accessibility**, abstracting the underlying complexities of AI for non-technical users.

3.1 Design Philosophy and Constraints

- **Color Scheme & Niche:** A professional, tech-focused palette (typically deep blues and whites) suitable for a Micro SaaS platform.
- **Responsiveness:** The system is a **Responsive Web Application**, ensuring compatibility across various screen sizes, though native mobile apps are currently out of scope.
- **Operational Boundary:** The UI is designed to minimize "hallucinations" by grounding all AI responses in the provided document context.

3.2 List of Screens and Controls

Screen Name	Data Displayed	UI Controls	Input Constraints
Sign Up	Registration fields	Textboxes, Phone Input, Submit Button	Password: 8+ chars + number
Login	Credential entry	Email Field, Password Field, Login Button	Registered and Admin-approved email
Dashboard	List of uploaded files and status	File Upload Button, Delete Icon, History Link	File formats: PDF, DOCX, CSV, TXT
Chat Interface	Query history and AI answers	Natural Language Textbox, Send Button	Textual string queries only
Admin Panel	List of pending user accounts	Approve/Reject Toggles, Search Bar	Administrative authorization required

3.3 Interface Interaction Logic

The interface is designed for immediate interaction. Once a user is authenticated, the "Upload" control triggers the backend ingestion pipeline. The "Query" control allows the user to ask a question, which then displays the generated answer along with optional references to the source chunks, ensuring transparency in how the AI reached its conclusion.