Project Title : LLM RAG Chatbot Application

Name : Prabha M

College Name : Sri Venkateshwara College of Engineering

Department : AI&DS

Register Number: 2127220502064

Year : IV

Problem Statement

The challenge was to build a **real-world chat application** that demonstrates the integration of **Large Language Models (LLMs)** with a **Retrieval-Augmented Generation (RAG) pipeline**.
The system needed to:

- Provide a **frontend chat interface** for users.
- Handle **user authentication** and **document uploads**.
- Enable LLMs to respond using **user-specific document knowledge**.
- Store **conversation history** in a database.
- Deliver a scalable and production-ready architecture.

Project Overview

My project is a fully functional RAG-powered chatbot application that meets the core requirements of the challenge. It allows users to register, log in, upload documents (PDF, DOCX, TXT), and engage in conversations where the LLM retrieves relevant information from the user's uploaded documents to generate informed responses. The application ensures data isolation by associating documents and conversations with specific user IDs, preventing cross-user data leakage.

Essential features implemented:

- **User Authentication**: Secure registration and login using JWT tokens, with password hashing via bcrypt for security.
- **Document Upload and Processing**: Users can upload files, which are ingested into a vector store for RAG. The system handles multiple file types (PDF via PyMuPDF, DOCX via docx2txt, TXT via file reading) and splits text into chunks for efficient retrieval.
- **Chat Interface**: A Streamlit-based UI for real-time chatting and supporting new conversations.

- **RAG-Powered Responses**: Queries are augmented with retrieved document chunks specific to the user, using a custom prompt to ensure responses are based on uploaded content or admit lack of information.
- **Conversation History**: Stored in a SQLite database, with messages linked to conversations and users.
- **Error Handling**: Comprehensive try-except blocks for API calls, document processing, and database operations, with user-friendly error messages in the chat.

This setup provides a seamless user experience, focusing on privacy and relevance in responses. While not all bonus features are included, the core application is robust and extensible.

Tech Stack

- **Frontend**: Streamlit for the interactive chat UI and forms.
- **Backend**: FastAPI for API endpoints, with Uvicorn as the server.
- **Database**: SQLite via SQLModel for ORM, storing users, conversations, and messages.
- **Authentication**: python-jose for JWT, passlib[bcrypt] for password hashing, and OAuth2PasswordBearer for token management.
- **LLM Integration**: langchain-openai with OpenRouter API (using GPT-3.5-turbo for cost-efficiency and reliability).
- **RAG and Embeddings**: langchain-huggingface for Sentence Transformers embeddings, langchain-community for tools, Chroma (via chromadb) as the vector database.
- **Document Processing**: PyMuPDF (fitz) for PDFs, docx2txt for Word docs.
- **Other**: python-multipart for file uploads, dotenv for environment variables.

Innovative Features and Enhancements

To address the challenge effectively, I incorporated several unique elements that enhance functionality, security, and usability beyond basic requirements:

- **User-Specific RAG Isolation**: Each document chunk is tagged with the user's ID in metadata. During retrieval, only the current user's documents are considered, ensuring personalized and secure responses. This prevents data mixing in multi-user scenarios, adding a layer of privacy not explicitly required but crucial for real-world applications.
- **Multi-Document Type Support with Robust Extraction**: While the bonus mentions "advanced RAG with multiple document types," I proactively handled PDF, DOCX, and TXT formats using specialized libraries (PyMuPDF for PDFs, docx2txt for Word docs). Text extraction is thorough, and empty or invalid files are gracefully handled with error messages.
- **Persistent Vector Store with Cleanup**: Using Chroma as the vector DB, documents are persisted across sessions. Uploaded files are automatically deleted after ingestion to manage storage, a practical optimization for deployment.
- **Debug and Maintenance Utilities**: I added two utility scripts—migrate_database.py for adding missing columns (e.g., conversation titles) to evolve the schema without data loss, and reset_database.py for clearing the database and vector store during development. These ensure maintainability, though they're not part of the runtime app.
- **Custom LLM Prompting for Reliability**: The RAG chain uses a tailored prompt to instruct the LLM to only answer based on provided context or admit limitations, reducing hallucinations and improving response accuracy.
- **Logging and Debugging**: Integrated logging throughout the backend for monitoring ingestion, retrieval, and errors, aiding in quick troubleshooting.

These additions make the project more production-ready, emphasizing security, extensibility, and user-centric design while fully covering the core RAG and chat requirements.