Assignment Title: Multi-User Document Search and Conversational Q&A System

---

Problem Statement

A company requires a **multi-user document search system** that allows users to query a collection of text documents and retrieve relevant excerpts based on their queries. Each user has restricted access to specific documents, ensuring they can retrieve answers only from the files they are authorized to view. Additionally, the system should provide a **conversational Q&A experience**, where follow-up questions maintain context from previous interactions.

The system should include a **basic user interface (UI)** for users to interact with the solution easily.

System Context

1. **Documents**: To replicate the scenario, assume publicly available earnings call PDFs from **4–5 companies** as the document dataset. Simulate user-specific access control such that:
    - Each user (represented by dummy email IDs) is allowed to access documents for only one or two companies.
    - For example:
        - User A (userA@email.com) can access Company X's documents.
        - User B (userB@email.com) can access Company Y's documents.
        - User C (userC@email.com) can access Company X and Company Y's documents.
2. **Functionality**:
    - Users should be able to **query the system** and retrieve relevant excerpts from documents they have access to.
    - The system should provide a **conversational experience** by maintaining context from previous user queries and answers.
    - Ensure that queries and responses are **isolated per user**, meaning User A should not see answers from documents User B has access to.
    - Include a **basic UI** where users can:
        - Log in or simulate access using their email IDs.
        - Submit queries and view retrieved answers.
        - Follow up with context-aware questions.
3. **Multi-User Access**:
    - Design the system to handle **simultaneous access** by multiple users. Queries from different users should not interfere with each other.
4. **Assumptions**:

    - You may simulate user authentication with **dummy email addresses** or basic access tokens—no need for full-fledged user management.
    - Users can access the system via a simple UI, which can be a basic web app or desktop interface.

What You Need to Deliver

1. **Code Solution**: Develop the system in **Python** to implement the above functionality, including the basic user interface.
2. **Submission**:
    - Push your code to a **public GitHub repository**.
    - Include sample PDFs (e.g., earnings call PDFs) and a setup guide (in a simple README) to test the solution.
    - Share the repository link along with the video demo.
3. **Demonstration**: During the interview, you should be ready to demonstrate this implementation covering at least the following points -
    - Logging in as different users (or simulating access for multiple users).
    - Submitting queries and retrieving answers via the UI.
    - Demonstrating how query isolation works, ensuring users cannot access unauthorized documents.
    - Highlighting the conversational component (e.g., follow-up queries maintaining context).

Example Scenario

Assume the following setup:

- **Document Dataset**: 5 PDFs from different companies (e.g., earnings calls for Company A, B, C, D, and E).
- **Users**:
    - Alice (alice@email.com) has access to Company A's documents.
    - Bob (bob@email.com) has access to Company B and Company C's documents.
    - Charlie (charlie@email.com) has access to Company D and Company E's documents.

Expected System Behavior:

- Alice logs into the system through the UI and submits a query, "What was the revenue reported in Q4?" The system returns an excerpt containing the relevant revenue details from **Company A's documents only**.
- Bob asks follow-up queries about Company B's earnings, and the system maintains context to provide accurate answers.

- Charlie cannot retrieve any information about Company A or Company B as access is restricted.

Guidelines

- Focus on designing a solution that meets the functional requirements inferred from the problem statement.
- Use **Python** for development.
- Build a **simple UI** (web-based or desktop interface) to interact with the system.
- Provide clear steps for running the system and testing with sample users and documents.