Project Title: Document Search Bot using OpenAI

1. Project Overview

- **Objective**: To create a document search bot that accepts a set of documents and answers user queries based on the information within those documents. The bot aims to leverage natural language processing to enhance user interaction.
- **User Roles**:
  - **Admin**: Responsible for managing documents, including uploading and deleting files.
  - **Normal User**: Can pose questions to the bot and receive answers derived from the documents.

2. Key Features and Functionalities

- **Document Management**:
  - Admins can upload documents, which are then stored in a local directory.
  - Only document metadata (title, upload date, etc.) is stored in an SQLite database.
- **User Interaction**:
  - A simple UI allows users to submit queries.
  - The bot processes the query using OpenAI's API to fetch relevant responses from the documents.
- **Response Generation**:
  - The bot matches user queries with the content of the documents and provides relevant answers, ensuring the accuracy of the information presented.

3. Technical Stack

- **Frontend**:
  - Developed using **HTML**, **JavaScript**, and **CSS**, providing a user-friendly interface for interaction.
- **Backend**:
  - Built with **Python Flask**, facilitating RESTful API creation to handle requests.
  - Integrated with **OpenAI** for natural language processing capabilities.

- **Database**:
    - Utilizes **SQLite** for storing user roles and document metadata.
    - Documents themselves are stored in a **local directory** to ensure easy access and management.
- **ORM**:
    - **SQL Alchemy** is employed to streamline database interactions and facilitate data management.

4. Workflow and Process

1. **Admin Functionality**:
    - Admin logs in and is presented with options to upload or delete documents.
    - Uploaded documents are saved in the local directory, and corresponding metadata is recorded in the SQLite database.
2. **User Queries**:
    - Normal users submit questions through the UI.
    - The backend retrieves and processes the question using the OpenAI API.
3. **Response Handling**:
    - The bot searches the relevant documents stored in the local directory and compiles the answers.
    - Responses are formatted and presented back to the user in an understandable manner.

5. Testing and Validation

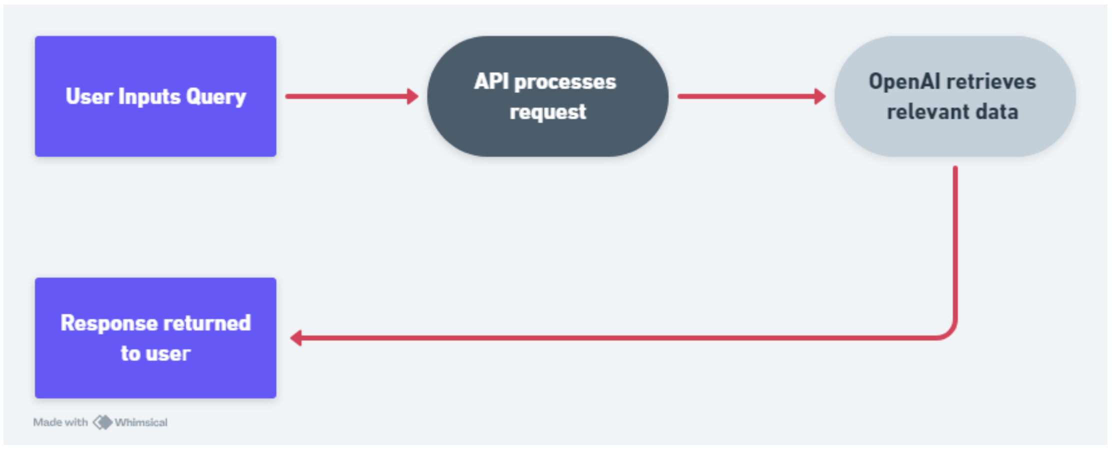
- **Testing Strategy**:
    - **Unit Tests**: Developed for individual components to ensure functionality (e.g., document upload, query handling).
    - **Integration Tests**: Assured that the components work together seamlessly (e.g., API interactions with the frontend).
    - **User Acceptance Testing (UAT)**: Conducted to validate the system against user expectations, confirming that it meets functional requirements.
- **Performance Metrics**:
    - Measured the accuracy of responses and the time taken to return results to users.

6. Tools and Deployment

- **Version Control**:
    - Source code is managed using **Git** and hosted on **GitHub** for collaboration and version tracking.
- **Development Environment**:
    - Developed in PyCharm for enhanced productivity and integrated debugging tools.
- **Deployment Options**:
    - The application can be containerized using **Docker** for deployment on cloud platforms such as AWS or GCP.

7. Design Diagram

Include a flowchart or architecture diagram:

- **User Interaction Flow**:
    - User submits query → API processes request → OpenAI retrieves relevant data → Response returned to user.
    - 

8. Conclusion

- This project demonstrates the effective use of AI for document search capabilities, showcasing an integrated solution that combines modern web technologies with powerful natural language processing. Future enhancements could include expanding document support, improving the UI/UX, and incorporating user feedback mechanisms.

ROSHAN KUNTULU

Sonata Software Ltd (23173)