AI-Powered HIS Customer Service

Overview:
As part of the task, you will develop a Retrieval-Augmented Generation (RAG) system that processes an Excel sheet containing data for the Pain & Go hospital, including information on physicians, schedules, and pricelists. The system's role will be to answer user inquiries, such as the schedule of a specific doctor, the most appropriate doctor or specialty based on the question, the price of a specific service, or retrieving metadata about the hospital. The system must be implemented using Langchain along with a vector database or a Large Language Model (LLM) of your choice.

Dataset:

You can find the dataset attached with the Email.

Task Requirements:

1. **Project Setup:**
    - Set up a GitHub repository for the project.
    - The project should be modular and organized. Write clean, readable code with appropriate comments and documentation. Include a **README.md** file that explains how to set up, run the application, and any dependencies.
    - The repo should contain well-defined functions and classes.
2. **Database Creation and Data Insertion:**
    - A database has to be created for the application to store the raw data, you could use SQLite or any other relevant database.
    - Use the dummy data provided to be inserted for each worksheet in a distinct database table.
3. **Data Ingestion and Vectorization:**
    - Your task is to retrieve the data from the database, ingest the data, process it, and vectorize it to make it ready for retrieval.
    - Use **Langchain** to manage the data ingestion and the interaction with the vector database.
    - Choose a vector database such as **FAISS**, **Chroma DB**, or **Weaviate** to store and retrieve the data.
4. **Data Insertion API:**
    - Create **APIs** (e.g., using **Flask** or **FastAPI**) for each table in the database in order to insert new record or multiple records for any table.

- Each API should do two things:
    - Insert the record(s).
    - Vectorize the new records and insert in the vector database.

5. **Question-Answering API:**
    - Create an **API** (e.g., using **Flask** or **FastAPI**) that takes a user query as input and returns an answer generated by the system.
    - The query should first retrieve relevant documents from the vector database based on the query's embeddings.
    - Then, use an LLM to generate an answer using the retrieved documents. The LLM should be able to take the documents and return a natural language response to the user's query.
6. **Streamlit UI:**
    - Develop a **Streamlit interface** that allows users to input queries.
    - The UI should interact with your API to submit questions and display the returned answers.
    - The Streamlit app should provide a simple and clean user interface for easy interaction with the system.
7. **Code Quality:**
    - Write clean, well-documented code.
    - Organize your code into well-structured files and functions.
    - Ensure your code is maintainable and reusable.
    - Ensure the project has a requirements.txt file to install the prerequisites.

Deliverables:

1. **Public GitHub Repository** containing:
    - Code for setting up the API and Streamlit app.
    - Scripts for data ingestion and vectorization.
    - README.md with setup instructions.
2. **API** that answers user queries.
3. **Streamlit UI** for testing the RAG flow.
4. **Clean and readable code** with clear comments and documentation.

Notes:

- The goal of this task is not only to build a working system but to also showcase your ability to write clean, maintainable code.
- Make sure your code is modular and each component (data ingestion, vectorization, retrieval, API, UI) is separated and easy to follow.
- In your README.md, include a section on how to run the application, dependencies, and any additional information needed to understand your implementation.
- Consider adding tests (unit tests or integration tests) to ensure the correctness of your system.

Evaluation Criteria:

- **Functionality:** Does the system correctly handle data ingestion, retrieval, and question answering?
- **Code Quality:** Is the code clean, modular, and well-documented?
- **User Interface:** Is the Streamlit UI intuitive and functional?
- **Creativity:** Did you go beyond the basic requirements, e.g., by adding extra features or improving the system's performance?
- **Documentation:** Is the README clear, with setup instructions and explanations?

Deadline: [**5 Hours**]

You must submit the final project by mail: Mariam.ragy@xyrisdigital.com

To know more about us: https://www.linkedin.com/company/xyris-digital-solutions/

Sincerely,
Mariam Ragy
HRBP
Mariam.ragy@xyrisdigital.com | 01234513099