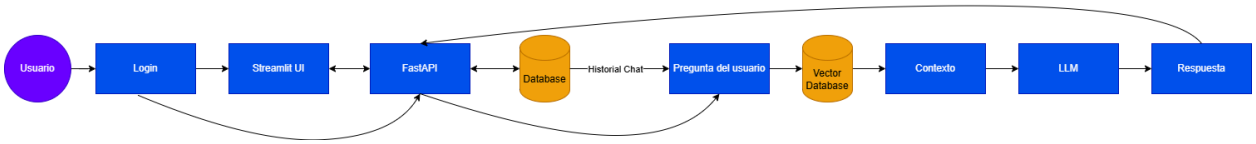
1. Diagrama simple del flujo del sistema



2. Componentes del sistema

Separé los microservicios en dos componentes uno para el Backend y otro para el FrontEnd. La elección principal de las tecnologías se basa en el conocimiento que tengo de las mismas, si bien he estado mas acostumbrado a la analitica de datos y no al desarrollo mas robusto de aplicaciones en algun momento había sido expuesto a estas tecnologías.

Microservicios

- **API Backend**: Desarrollado en FastAPI, expone los endpoints para el procesamiento de preguntas, manejo de sesiones, ingesta de documentos y autenticación de usuarios.
- **Frontend**: Aplicación Streamlit con interfaz interactiva para que el usuario pueda cargar documentos, seleccionar el modelo LLM (OpenAI o local) y realizar consultas.

Base vectorial

- **ChromaDB**: Almacena las representaciones vectoriales (embeddings) de los documentos. Es utilizada por LangChain para la recuperación semántica (RAG).

Dado que el requerimiento es para uso privativo consideré utilizar una herramienta OpenSource local para el almacenamiento de la representación del texto de los documentos. Se utilizó Sentence Transformers para la conversión a Embeddings de los fragmentos de texto. Se utilizó un modelo multilenguaje, dado que se espera que los documentos sean en español.

Sin embargo, considero que falta ahondar más en el tipo de archivos a trabajar. Dado que si estos tendrían alguna estructura particular sería más eficiente trabajar los archivos primeros y guardarlos en un formato más amigable para los LLM, ya que sin esto los chunks de texto pueden no tener coherencia si por ejemplo se trabajan tablas (columnas)

LLMs (Modelos de lenguaje)

Respecto a los modelos de lenguajes, indudablemente OpenAI es la herramienta con mayor documentación para la implementación de este tipo de sistemas. Para el desarrollo de un prototipo rápido es una buena elección.

En cuanto al modelo local, estuve evaluando diferentes modelos ligeros para poder hacer una implementación local. Consideré Phi-4 porque ha tenido buenas métricas de rendimiento,

especialmente en relación a análisis de datos. Si bien el problema no especifica qué tipo de información se cargará por el tipo de archivo .csv y .xlsx asumo que tendrá datos estructurados.

- **OpenAI (gpt-4o, gpt-4o-mini)**: Para procesamiento en la nube.
- **Modelo local Phi-4-mini-instruct**: Para entornos locales con mayor privacidad.

Capa de seguridad

No se incluye ninguna capa de seguridad en la aplicación. No es mi fuerte en el desarrollo y para este ejemplo simplemente se utilizó la contraseña cifrada para no guardar en base de datos la contraseña real. Por lo tanto se realizó una **autenticación básica con credenciales cifradas** y se generaron usuarios Dummy "quemados" en base de datos. Para el control.

3. Justificación del stack tecnológico

- **FastAPI**: Es un framework moderno de alto rendimiento. Bueno para construir APIs y generar documentación rápida.
- **Streamlit**: Sencillo de usar, ideal para prototipado rápido y pruebas rápidas.
- **LangChain**: Permite componer cadenas de RAG de forma modular y flexible.
- **ChromaDB**: Base vectorial ligera, embebible y eficiente para entornos de desarrollo locales, además OpenSource.
- **OpenAI**: Diría que el estándar para empezar cualquier prototipo, ya que hay documentación y funciones que han sido desarrolladas específicamente para estos modelos.
- **Phi-4 Mini (Microsoft)**: Alternativa local eficiente, ideal para contextos con restricción de datos.

4. Ingesta, almacenamiento y consulta

Ingesta:

- El usuario sube documentos (CSV o Excel) desde el frontend. En código inicialmente se implementaron splitters para otros formatos de archivos que se podrían ampliar a futuro.
- Los documentos son procesados en FastAPI para extraer el texto y generar embeddings mediante embeddings de HugginFace y Sentence Transformers.

Almacenamiento:

- Los datos del usuario, logs de conversaciones, y relación de documentos se agregaron una una base de datos relacional **SQLite** que se podría exportar a otros servicios locales o en nube como PostgreSQL
- Los embeddings se almacenan en **ChromaDB**.

Consulta:

- El usuario realiza una pregunta.
- El sistema convierte la pregunta en embedding y recupera documentos relevantes (retriever)
- Con ayuda de LangChain, se formula un prompt con contexto y se consulta al LLM (OpenAI o **Phi-4 Mini**).

5. Recomendaciones para escalar

- **Vector store**: Migrar de ChromaDB a otras soluciones, Pinecone o Qdrant que entiendo tienen mayores capacidades.
- **Procesamiento por batches**: Implementar una cola de procesamiento para ingestiones masivas.
- **GPU escalable**: Usar servidores con GPU o instancias en la nube con mayor capacidad para los modelos locales.
- **Caching y versionado**: Cachear respuestas frecuentes, versionar documentos y limitar la carga de embeddings redundantes.

Ever Augusto Torres Silva

Prueba para IQVIA