1. **User Data Upload / Fetch Data from Azure**
   Description: The user uploads a file (PDF, CSV, Excel) through a web interface or API. This file contains tabular medical or clinical data to be analyzed.

Or

   Description: The chatbot backend connects to an Azure database to retrieve the supply purchase data.

↓

2. **Data Parser**
   Description: Retrieved tables with data are parsed row by row and then converted into meaningful text chunks where each is marked with metadata like a column header or a row ID.

↓

3. **Embeddings Generator**
   Description: All text chunks are passed through a HuggingFace embedding model to create vector representations that will capture the semantic meaning of the data.

↓

4. **Store Embeddings (Qdrant)**
   Description: The embeddings and their metadata are stored in a vector database like Qdrant. This enables efficient similarity search for later retrieval. It allows us to perform fast similarity searches based on user queries.

5. **Embedding Refresh**
   Description: A scheduled or triggered background job checks the Azure database for new or updated rows. Any detected changes are re-parsed and re-embedded, and the vector store is updated accordingly. This ensures the chatbot stays aligned with the most current hospital supply data.

↓

6. **User-Chatbot Interaction**
   Description: The user interacts with the chatbot through a web UI, asking natural language questions about the tabular data (e.g., "What is the total amount of Supply X purchased in the last quarter?").

↓

7. **Query Embedding**
   Description: The user's question is converted into an embedding vector via the HuggingFace model. This allows for semantic similarity comparisons.

↓

8. **Context Retrieval**

Description: The system queries the vector store using the question embedding to find the most relevant chunks of table data. These chunks serve as context for answering the user's question.

↓

9. **Prompt Construction**
   Description: Retrieved chunks are combined with the user's question into a single prompt formatted for GPT-4. This prompt is carefully formatted to provide the LLM with relevant background for accurate response generation.

↓

10. **LLM (GPT-4 / OpenAI API)**
    Description: The constructed prompt is sent to GPT-4 via OpenAI's API. The LLM uses the context to generate a clear and helpful natural language answer.

↓

11. **Return Response to User**
    Description: The final answer is returned to the user in the chat interface.

↓

12. **Log Chatbot Conversations**
    Description: Each user question, the retrieved context, GPT-4 prompt, and the final response are logged into a persistent database or file system.

    Questions for the **Logging** feature:
    - How long to store logs?
    - SQLite? Or AzureSQL

Security Measures to be added:

- Track the number of user queries per user, and alert on usage spikes.