Clinical Knowledge Assistant – Approach and Reflection

Approach

This project implements a lightweight Retrieval-Augmented Generation (RAG) system designed to provide psychiatrists with quick, evidence-based guidance for treating Generalized Anxiety Disorder (GAD). The pipeline is intentionally modular so that each stage—ingestion, embedding, retrieval, and interaction—can be independently iterated or replaced as needed.

The ingestion and preprocessing phase is handled by `document_utils.py`, which manages URL parsing, downloading, and PDF loading using `PyPDFLoader`. Documents are chunked with `RecursiveCharacterTextSplitter`, with configurable parameters for `CHUNK_SIZE` and `CHUNK_OVERLAP`. This design keeps URL discovery separate from downloads and logs progress for long ingestion processes.

For embeddings and retrieval, document chunks are encoded using OpenAI embeddings and stored in an `InMemoryVectorStore` for simplicity. The interface design allows seamless replacement with FAISS, Chroma, or PGVector for persistence or scalability without modifying other components.

The agent layer, defined in `agent_utils.py`, provides reusable functions for retrieval and conversation. `make_retrieve_context_tool` binds the retrieval tool to a given vector store, and `make_rag_agent` returns a callable chat handler bound to both the model and store. It also keeps track of chat history. The number of messages in context can be set with `MAX_HISTORY_TURNS`. Finally, `main.py` orchestrates the full pipeline and passes a bound chat function to `gradio_app.py`, which defines a compact Gradio `ChatInterface`. Configuration values such as model names and chunk sizes are stored in environment files for easy modification.

Tools & Libraries

The implementation uses LangChain for RAG primitives (agents, loaders, vectorstores), OpenAI for chat and embedding models, and Gradio for a simple interactive chat interface. Configuration management is handled via dotenv, allowing convenient use of `.env` and `prompts.env` files for environment variables. The OpenAI model I specifically used was gpt-5-nano and the embedding model was text-embedding-3-large.

Challenges & Mitigations

A few technical challenges emerged during the process. FDA labels and clinical guidelines vary in formatting and structure, so `PyPDFLoader` was used with overlapping text chunks to improve context recall across fragmented layouts. State management was handled carefully using closures rather than module-level globals in order reduce side effects and improve testability. Lastly, the trade-off between retrieval recall and latency was addressed by making chunk size and overlap configurable, allowing for performance tuning without code changes.

Potential Improvements if Given More Time

Several extensions could enhance this prototype:

- **Structured Output:** Implement a citation formatter with token budgeting per citation and add a rule-based or model-assisted guardrail to verify clinical claims.
- **Data Layer:** Replace the in-memory vector store with FAISS, Chroma, or PGVector for larger corpora and persistence. Precompute and cache embeddings to reduce startup time.
- **Prompting & Control:** Refine role and system prompts to enforce clinical tone, citation policies, and contextual safety. Introduce UI toggles for concise vs. detailed responses or clinician vs. patient-safe modes.
- ~~**History:** Implement chat history so that follow up questions can easily be asked.~~
- **File Types and Images**: Allow for processing of images within PDFs and allow for additional files other than PDFs.
- **UI/UX:** Display retrieved snippets and citations inline, add document/source filters, and include features like streaming token output or context-based retries.
- **Evaluation & Safety:** Develop an evaluation set based on the provided questions to measure accuracy and citation correctness. Integrate explicit safety checks for black box warnings and contraindications.

How to Run

1. Create a Python 3.10 virtual environment and install dependencies:

   ```
   python3.10 -m venv .venv
   source .venv/bin/activate
   pip install -r requirements.txt
   ```

2. Set `OPENAI_API_KEY` in your `.env` file and add PDF URLs to `documents/urls.txt`

   ```
   python3 main.py
   ```

4. Go to `http://127.0.0.1:7861` in your browser