Modular Code Flow Explanation

Overview

This application enables end-to-end processing of an audio query, retrieving relevant information from a document, and generating a spoken response using an LLM-powered RAG pipeline.

Key Components & Technologies Used

- **Frontend:** Streamlit (UI for uploading files and recording queries)
- **Speech Recognition:** Google Cloud Speech-to-Text
- **Vectorstore:** DocArrayInMemorySearch (for document retrieval)
- **Embeddings Model:** `nomic-embed-text:latest` (via Ollama)
- **Language Model:** `llama3.2:1b` (via Ollama)
- **Sentiment Analysis:** LLM-based analysis via structured prompt engineering
- **Text-to-Speech:** Google Cloud Text-to-Speech

Main Functions

1. `convert_audio_to_text(audio_bytes, client)`
   - Converts spoken input to text using Google Speech-to-Text.
2. `load_and_create_vector_db(pdf_path)`
   - Loads PDF, splits text into chunks, and stores it in an in-memory vector database.
3. `perform_sentiment_analysis(llm, text)`
   - Analyzes user sentiment and returns structured JSON output.
4. `search_vector_db(vectorstore, query)`
   - Retrieves relevant document chunks for the given query.
5. `generate_response(llm, context, query, sentiment)`

- Creates an empathetic response using the retrieved context and sentiment data.

6. `convert_text_to_speech(client, text, output_path)`
   - Converts generated text to speech and saves as an audio file.

Why These Choices?

- **Ollama for LLM & Embeddings:**
  - Runs locally, reducing latency and cloud costs.
  - `llama3.2:1b` is lightweight, making it ideal for personal/local deployments.
- **Google Cloud Speech-to-Text & Text-to-Speech:**
  - High accuracy in speech recognition and natural-sounding text-to-speech conversion.
- **DocArrayInMemorySearch Vectorstore:**
  - Simple and efficient for quick document retrieval.
  - No external database required, making it ideal for testing and small-scale applications.

End-to-End Flow

1. **User speaks a query** → Converted to text.
2. **Query analyzed for sentiment** → Sentiment & emotions extracted.
3. **Query used to fetch relevant document chunks** → Vector search performed.
4. **LLM generates an empathetic response** → Response aligned with sentiment & context.
5. **Response converted to speech** → Played back to the user.
6. **Latency measured** → Performance displayed.

This modular approach ensures that components can be swapped or improved without affecting the entire pipeline.