Conversational Q&A Chatbot

Q1: How does your chatbot maintain and utilize chat history to provide context-aware answers?
A: My chatbot maintains chat history by storing each user and bot message in a structured format, such as a list of message objects or a database table keyed by session. This history is passed to the LLM as part of the prompt, enabling the model to generate contextually relevant responses. For multi-turn conversations, I summarize or selectively include recent exchanges to stay within the model's context window, ensuring continuity without exceeding token limits. This approach helps the chatbot understand follow-up questions and maintain coherent dialogue.

Q2: Explain the difference between using Chains and Agents in LangChain for retrieval. Why did you choose one over the other?
A: Chains in LangChain are linear pipelines where each step's output is passed to the next, ideal for predictable, repeatable tasks like retrieval-augmented generation. Agents, on the other hand, can make decisions at runtime, choosing tools or actions based on the input, which is powerful for complex, dynamic workflows. For my project, I chose Chains because the retrieval and QA process is well-defined and doesn't require dynamic tool selection. This makes the system more efficient and easier to debug.

Q3: How do you implement and optimize document chunking and embedding for efficient retrieval?
A: I use a RecursiveCharacterTextSplitter to break documents into overlapping chunks, typically 500–1000 characters with 50–100 character overlaps. This balances context preservation with retrieval granularity. Each chunk is embedded using a transformer-based model, such as all-MiniLM-L6-v2, and stored in a vector database. I empirically tuned chunk size and overlap to maximize retrieval relevance while minimizing context fragmentation, using retrieval performance metrics and spot-checking results.

Q4: What vector store did you use, and why? How do you handle persistence and scalability?
A: I used Chroma as my vector store because it offers fast similarity search, easy integration with LangChain, and supports persistent storage. For persistence, I specify a directory for Chroma to save its index, allowing the system to reload embeddings without recomputation. For scalability, Chroma supports sharding and can be deployed with distributed storage. In production, I would consider managed solutions like Pinecone or Qdrant for horizontal scaling and high availability.

Q5: How do you handle the reformulation of follow-up questions to standalone queries?
A: I use a prompt template that combines the chat history with the current user query, instructing the LLM to rewrite ambiguous follow-ups as standalone questions. This ensures the retriever receives clear, context-rich queries, improving retrieval accuracy. I also validate reformulations by comparing them to expected standalone versions in test cases.

Q6: Describe your prompt engineering approach for the retrieval-augmented generation chain.

A: I iteratively designed prompts to clearly instruct the LLM on its role, the context provided, and the expected answer format. I tested different phrasings and system messages, prioritizing clarity, factuality, and empathy (since the domain is mental health). I also included explicit instructions to only use retrieved context and to admit when information is insufficient, reducing hallucinations.

Q7: How do you manage multi-user sessions and chat histories in your system?
A: I assign each user a unique session ID and store their chat history in a backend database (e.g., SQLite, PostgreSQL). This allows concurrent conversations and ensures each user's context is isolated. Session management is handled at the API or application layer, with history retrieval and updates tied to the session.

Q8: What are the main challenges in combining retrieved documents with LLM-generated answers, and how do you mitigate hallucinations?
A: The main challenge is ensuring the LLM bases its answer strictly on retrieved context, not its internal knowledge. I mitigate hallucinations by crafting prompts that instruct the model to use only the provided context and to state when information is unavailable. I also evaluate outputs for factuality and use retrieval confidence scores to decide when to answer or defer.

Q9: Can you explain how you integrated external web content into your knowledge base?
A: I use web scrapers or APIs to fetch relevant web pages, then preprocess the text (cleaning, deduplication, chunking) before embedding and adding to the vector store. I schedule regular updates to keep the knowledge base current and use metadata tags to track source and recency.

Q10: How would you extend this system to support real-time document updates and incremental indexing?
A: I would implement a background service that monitors document sources for changes, processes new or updated files, generates embeddings for new chunks, and updates the vector store incrementally. Chroma and similar databases support dynamic addition and deletion of vectors, enabling real-time updates without full reindexing.