Ollama Chatbot

Q1: Why did you choose Ollama as your LLM provider, and what advantages does it offer over other LLM APIs?
A: I chose Ollama because it allows running LLMs locally, offering full data privacy and eliminating latency and cost associated with cloud APIs. It's ideal for environments with strict data governance or limited internet access. Additionally, Ollama supports a variety of open-source models, enabling experimentation and customization.

Q2: How does your Streamlit interface interact with the Ollama model?
A: The Streamlit interface collects user input, sends it to a backend function that queries the Ollama model via its API or Python SDK, and displays the response. The interface manages conversation history and provides a user-friendly chat experience, with real-time updates and error handling.

Q3: Explain your prompt template design and how it influences the chatbot's responses.
A: I design prompt templates to set the system's persona, context, and response style. For example, I instruct the model to be concise, factual, and empathetic. Well-crafted prompts guide the LLM to produce relevant, safe, and on-brand responses, reducing ambiguity and hallucination.

Q4: How do you handle user input validation and error handling in your app?
A: I validate user input for length, content, and appropriateness before sending it to the model. I wrap API calls in try/except blocks to catch and display errors gracefully, ensuring users receive informative feedback rather than generic failures.

Q5: What are the trade-offs of using a local LLM versus a cloud-based API in terms of latency, cost, and privacy?
A: Local LLMs offer lower latency for small models and complete data privacy, but require significant hardware resources and may lag behind cloud APIs in model size and capabilities. Cloud APIs provide access to state-of-the-art models and scalability but incur ongoing costs and potential privacy concerns.

Q6: How do you manage model parameters such as temperature and max tokens in your system?
A: I expose these parameters in the Streamlit sidebar, allowing users to adjust model creativity and response length. I set sensible defaults based on testing and document the impact of each parameter to guide user choices.

Q7: Describe how you would implement conversation history or context awareness in this chatbot.
A: I maintain a list of user and bot messages, appending each turn to the conversation. This history is included in the prompt for each new response, enabling the model to reference prior exchanges and maintain context.

Q8: How do you ensure the chatbot responses are safe and aligned with user expectations?
A: I use prompt engineering to instruct the model to avoid unsafe topics and provide disclaimers when needed. I also implement content filtering and monitor outputs for inappropriate content, iterating on prompts and filters as necessary.

Q9: What testing strategies do you use to validate the chatbot's behavior?
A: I use unit tests for backend functions, simulate user conversations to test context management, and perform manual QA to assess response quality. I also collect user feedback for continuous improvement.

Q10: How would you scale this app for multiple concurrent users?
A: I would deploy the backend as a web service with session management, ensuring each user's conversation is isolated. For heavy loads, I'd use horizontal scaling (multiple model instances) and load balancing, possibly containerizing the app for cloud deployment.