# LangChain Chatbot with Streamlit - Documentation

## **1. Overview**

This document provides a comprehensive explanation of the code used to build an interactive chatbot application using LangChain, OpenAI's GPT, and Streamlit.

## **2. Purpose**

The chatbot allows users to converse with an AI assistant in a persistent, context-aware manner. It uses:

- LangChain for conversation management and prompt templating.

- OpenAI GPT (gpt-3.5-turbo) for natural language responses.

- Streamlit for a responsive web-based chat interface.

## **3. File Structure and Requirements**

- .env: Stores the OpenAI API key.

- streamlit\_app.py: Main application file.

Dependencies:

- streamlit  
- python-dotenv  
- langchain  
- openai

Install dependencies using:

**pip install streamlit python-dotenv langchain openai**

## **4. Components Explained**

### **4.1 Environment Configuration**

load\_dotenv()  
openai\_api\_key = os.getenv("OPENAI\_API\_KEY")

- Loads the API key from a .env file to authenticate with OpenAI.

### **4.2 Streamlit Setup**

st.set\_page\_config(page\_title="Langchain Chatbot", page\_icon="🤖")

- Customizes the web page title and favicon.

## **5. Initialization and Caching**

@st.cache\_resource  
def initialize\_chatbot():

- Ensures that the chatbot components (model, memory, prompt) are only initialized once per session.

Inside initialize\_chatbot():

1. LLM Setup: Initializes ChatOpenAI with API key.

2. Memory: Uses ConversationBufferMemory to keep the chat history.

3. Prompt Template: Defines how user input and history are formatted.

4. ConversationChain: Bundles the LLM, memory, and prompt for context-aware responses.

## **6. User Interface (UI)**

Title and Description:

st.title("🤖 Langchain Chatbot")

st.markdown("Ask me anything! I'll remember our conversation.")

Display Previous Messages:

for message in st.session\_state.messages:

- Displays chat history to maintain a conversational flow.

Accept User Input:

if prompt := st.chat\_input("What would you like to ask?"):

- Captures user input in a conversational chat-style UI.

## **7. Response Handling**

ai\_response = conversation\_chain.predict(input=prompt)

- Uses the LangChain ConversationChain to generate a response.

Display and Save Response:

- User and AI messages are appended to st.session\_state.messages.

- Displayed using st.chat\_message().

Error Handling:

except Exception as e:  
 st.error("An error occurred...")

- Captures and displays any runtime errors, including API issues.

## **8. Reset Functionality**

if st.sidebar.button("Clear Chat History"):  
- Provides a button to clear chat memory and rerun the app.

## **9. Notes and Best Practices**

- API Key Security: Never hard-code keys; always use .env.

- Memory Use: Helps maintain conversation continuity.

- Caching: Prevents repeated initialization during Streamlit reruns.

- UI Simplicity: The chat interface is intuitive and minimalistic.

## **10. Possible Extensions**

- Add file upload and retrieval-augmented generation (RAG).

- Enable persona-based or tool-augmented interactions.

- Deploy to Streamlit Cloud or another platform.

## **11. Conclusion**

This chatbot implementation demonstrates how to combine LangChain's conversation capabilities with Streamlit's UI to create an intelligent, context-aware chatbot powered by OpenAI's GPT.