DAY – 64, 65, 66

31 October & 3, 4 November 2025

Here's your **fully working Flask version** of the "Conversational RAG Without PDF Uploads" app

---

Flask Version: app.py

```
# ================================
# Conversational RAG (Flask Version, No PDFs)
# ================================
from flask import Flask, request, jsonify
import os
from dotenv import load_dotenv

from langchain.chains import create_history_aware_retriever, create_retrieval_chain
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_chroma import Chroma
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_groq import ChatGroq
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_text_splitters import RecursiveCharacterTextSplitter

# ============================
# Flask App Initialization
# ============================
app = Flask(__name__)

# ============================
# Environment Setup
# ==============================
load_dotenv()
os.environ["HF_TOKEN"] = os.getenv("HF_TOKEN")

embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# ==============================
# Static Knowledge Base
# ==============================
sample_docs = [
    {"page_content": "RAG (Retrieval-Augmented Generation) combines document retrieval with language generation to answer user questions accurately."},
    {"page_content": "LangChain allows developers to build modular and memory-based LLM applications by connecting chains, prompts, retrievers, and memory."},
    {"page_content": "Groq provides ultra-fast inference for large language models with the Gemma2-9b-It model, ideal for conversational AI applications."},
    {"page_content": "Chroma is a lightweight vector database for managing and searching embeddings efficiently in local or cloud environments."},
]

text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
splits = text_splitter.create_documents([doc["page_content"] for doc in sample_docs])
vectorstore = Chroma.from_documents(documents=splits, embedding=embeddings)
retriever = vectorstore.as_retriever()

# ==============================
# Global Store for Session History
# ==============================
store = {}


def get_session_history(session: str) -> BaseChatMessageHistory:
    """Maintain independent chat history per session ID."""
    if session not in store:
        store[session] = ChatMessageHistory()
    return store[session]



# ==============================
# Helper: Create RAG Chain
# ==============================
def create_rag_chain(api_key):
    llm = ChatGroq(groq_api_key=api_key, model_name="Gemma2-9b-It")

    # Contextual question reformulation
    contextualize_q_system_prompt = (
        "Given a chat history and the latest user question which might reference context "
        "in the chat history, formulate a standalone question that can be understood without "
        "the chat history. Do NOT answer the question; just reformulate it if needed."
    )
    contextualize_q_prompt = ChatPromptTemplate.from_messages(
        [
            ("system", contextualize_q_system_prompt),
            MessagesPlaceholder("chat_history"),
            ("human", "{input}"),
        ]
    )
    history_aware_retriever = create_history_aware_retriever(llm, retriever,
contextualize_q_prompt)

    # QA system
    system_prompt = (
        "You are an assistant for question-answering tasks. Use the retrieved context to answer "
        "the question. If you don't know the answer, say 'I don't know'. Keep responses short and
relevant.\n\n{context}"
```

```
    )
    qa_prompt = ChatPromptTemplate.from_messages(
        [
            ("system", system_prompt),
            MessagesPlaceholder("chat_history"),
            ("human", "{input}"),
        ]
    )

    question_answer_chain = create_stuff_documents_chain(llm, qa_prompt)
    rag_chain = create_retrieval_chain(history_aware_retriever, question_answer_chain)

    conversational_rag_chain = RunnableWithMessageHistory(
        rag_chain,
        get_session_history,
        input_messages_key="input",
        history_messages_key="chat_history",
        output_messages_key="answer",
    )

    return conversational_rag_chain


# ==============================
# Flask Route: Chat Endpoint
# ==============================
@app.route("/chat", methods=["POST"])
def chat():
    """
    POST JSON:
    {
        "api_key": "<YOUR_GROQ_API_KEY>",
        "session_id": "session123",
        "question": "What is RAG?"
    }
    """
    data = request.get_json()
    api_key = data.get("api_key")
    session_id = data.get("session_id", "default_session")
    user_input = data.get("question")

    if not api_key:
        return jsonify({"error": "Missing Groq API key"}), 400
    if not user_input:
        return jsonify({"error": "Missing user question"}), 400

    rag_chain = create_rag_chain(api_key)
    session_history = get_session_history(session_id)

    response = rag_chain.invoke(
        {"input": user_input},
        config={"configurable": {"session_id": session_id}},
    )

    return jsonify({
        "session_id": session_id,
        "question": user_input,
        "answer": response["answer"],
        "chat_history": [str(m) for m in session_history.messages],
    })


# ============================
# Root Endpoint
# ============================
@app.route("/")
```

```
def home():
    return jsonify({"message": "Conversational RAG Flask API Running"})



# ==============================
# Run the Flask App
# ==============================
if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

How It Works

- **Endpoint:** /chat
  Accepts POST requests with JSON body:
- {
-   "api_key": "your_groq_api_key",
-   "session_id": "abc123",
-   "question": "Explain RAG in simple terms"
- }
- **Response Example:**
- {
-   "session_id": "abc123",
-   "question": "Explain RAG in simple terms",
-   "answer": "RAG combines document retrieval and language generation to answer questions accurately.",
-   "chat_history": [
-     "Human: Explain RAG in simple terms",
-     "AI: RAG combines document retrieval and language generation to answer questions accurately."
-   ]
- }
- Chat history is automatically remembered per session_id using in-memory store.

Run It

1. Save file as app.py
2. Run in terminal:
3. python app.py
4. Test using **Postman** or **cURL**:
5. curl -X POST http://127.0.0.1:5000/chat \
6. -H "Content-Type: application/json" \
7. -d '{"api_key": "your_groq_api_key", "session_id": "abc123", "question": "What is RAG?"}'