DAY – 45

1 October 2025

Features:

Uses Flask API
Maintains per-session chat history
Works with Groq LLM (Gemma2-9b-It)
Uses Chroma + HuggingFace embeddings for document retrieval (you can later add PDFs if you want)
Returns clean JSON responses for frontend (HTML/JS UI can call it easily)

File: app.py

```
from flask import Flask, request, jsonify
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
import os
from dotenv import load_dotenv

# ========== Load environment variables ==========
load_dotenv()
os.environ['HF_TOKEN'] = os.getenv("HF_TOKEN")
```

```
# ========== Initialize Flask App ==========
app = Flask(__name__)

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

# ========== Store session-based chat histories ==========
session_store = {}

# ========== Create Example Knowledge Base (You can replace this later) ==========
# Instead of PDF loader, we'll use a few static example documents.
example_docs = [
    {"page_content": "LangChain is a framework for developing applications powered by
language models."},
    {"page_content": "RAG stands for Retrieval-Augmented Generation, combining LLMs
with external knowledge sources."},
    {"page_content": "Groq API provides fast inference for models like Gemma and
Mixtral."}
]

vectorstore = Chroma.from_documents(example_docs, embedding=embeddings)
retriever = vectorstore.as_retriever()

# ========== Flask API Route ==========
@app.route('/chat', methods=['POST'])
def chat():
    data = request.json
    user_input = data.get('question')
    session_id = data.get('session_id', 'default_session')
    groq_api_key = data.get('groq_api_key')

    if not groq_api_key:
        return jsonify({"error": "Missing Groq API key"}), 400
    # Initialize LLM
    llm = ChatGroq(groq_api_key=groq_api_key, model_name="Gemma2-9b-It")

    # ===== Step 1: 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 which can be understood "
        "without the chat history. Do NOT answer the question, "
        "just reformulate it if needed and otherwise return it as is."
    )

    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)

    # ===== Step 2: Answer Question Using Context =====
    system_prompt = (
        "You are an assistant for question-answering tasks. "
        "Use the following pieces of retrieved context to answer "
        "the question. If you don't know the answer, say that you don't know. "
        "Use three sentences maximum and keep the answer concise.\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)

    # ===== Step 3: Manage Session Chat History =====
    def get_session_history(session: str) -> BaseChatMessageHistory:
        if session not in session_store:
            session_store[session] = ChatMessageHistory()
        return session_store[session]

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

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

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

# ========== Run the Flask App ==========
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Example JSON Request (Frontend or Postman)

POST → http://127.0.0.1:5000/chat
Body (JSON):

```
{
  "groq_api_key": "your_groq_api_key_here",
  "session_id": "user123",
  "question": "What is RAG?"
}
```

Response:

```
{
  "session_id": "user123",
  "answer": "RAG stands for Retrieval-Augmented Generation, a method that combines large
language models with external knowledge sources.",
  "chat_history": [
    "Human: What is RAG?",
    "AI: RAG stands for Retrieval-Augmented Generation, combining LLMs with external
knowledge sources."
  ]
}
```