DAY – 60

27 October 2025

So you'll **manually define the knowledge base** (or keep preloaded context later), but no PDF upload interface.

Here's your **cleaned and working version**

---

conversational.py — Without PDF Uploads

```
## RAG Q&A Conversation With Chat History (No PDF Upload)
import streamlit as st
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
from langchain_community.document_loaders import TextLoader
import os

from dotenv import load_dotenv
load_dotenv()

# ====== Initialize Embeddings ======
os.environ['HF_TOKEN'] = os.getenv("HF_TOKEN")
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
```

```
# ====== Streamlit UI ======
st.title("Conversational RAG Chatbot (No PDF Upload)")
st.write("Chat with built-in knowledge or preloaded documents.")

# ====== Input Groq API Key ======
api_key = st.text_input("Enter your Groq API key:", type="password")

if api_key:
    llm = ChatGroq(groq_api_key=api_key, model_name="Gemma2-9b-It", temperature=0.3)

    # ====== Static Knowledge Base (Example Content) ======
    # You can replace this text file path or add your own local knowledge data
    default_file = "data/default_knowledge.txt"
    if not os.path.exists(default_file):
        st.warning("□ No knowledge base file found. Please create
'data/default_knowledge.txt'.")
    else:
        loader = TextLoader(default_file)
        docs = loader.load()

        # Split and create embeddings
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=2000, chunk_overlap=200)
        splits = text_splitter.split_documents(docs)
        vectorstore = Chroma.from_documents(documents=splits, embedding=embeddings)
        retriever = vectorstore.as_retriever(search_kwargs={"k": 4})

        # ====== Context Reformulation Prompt ======
        contextualize_q_system_prompt = (
            "Given a chat history and the latest user question which might reference prior context,
"
            "formulate a standalone question that can be understood without chat history. "
            "Do NOT answer the question, just rephrase it if necessary."
```

```
    )
    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)

    # ====== Question Answering Prompt ======
    system_prompt = (
        "You are a helpful assistant for question-answering tasks. "
        "Use the retrieved context to answer concisely. "
        "If unsure, say you don't know.\n\n{context}"
    )
    qa_prompt = ChatPromptTemplate.from_messages([
        ("system", system_prompt),
        MessagesPlaceholder("chat_history"),
        ("human", "{input}"),
    ])

    # ====== Build Conversational Chain ======
    question_answer_chain = create_stuff_documents_chain(llm, qa_prompt)
    rag_chain = create_retrieval_chain(history_aware_retriever, question_answer_chain)

    # Manage chat history
    session_id = st.text_input("Session ID", value="default_session")

    if "store" not in st.session_state:
        st.session_state.store = {}

    def get_session_history(session: str) -> BaseChatMessageHistory:
        if session not in st.session_state.store:
```

```
            st.session_state.store[session] = ChatMessageHistory()
        return st.session_state.store[session]

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

    # ====== Chat Interface ======
    user_input = st.text_input("Ask your question:")
    if user_input:
        session_history = get_session_history(session_id)
        response = conversational_rag_chain.invoke(
            {"input": user_input},
            config={"configurable": {"session_id": session_id}},
        )
        st.write("□ Assistant:", response["answer"])
        st.write("□ Chat History:", session_history.messages)

else:
    st.warning("Please enter your Groq API key to start chatting.")
```

Key Changes

Removed:

- st.file_uploader and all PDF handling
- Temporary file saving (temp.pdf)
- Loops for multiple files

Added:

- TextLoader to read from a **fixed knowledge file** (data/default_knowledge.txt)
- Clearer sectioning for retriever, prompts, and conversational chain
- Still supports chat history per session

Folder Setup

```
project/
│
├── conversational.py
├── data/
│   └── default_knowledge.txt   ← (add your text or knowledge content)
├── .env                        ← (contains HF_TOKEN and GROQ_API_KEY)
└── requirements.txt
```