DAY – 57

21 October 2025

Let's fix that **properly** by ensuring:

1. Each user has an **independent chat history** (unique session).
2. Session persists correctly across requests.
3. Multiple users can chat simultaneously (Flask session isolation).

Fixed Full Working Code (Session-Persistent Multi-User Chatbot)

chatbot_logic.py

Keep your current version as-is — it's fine.
We only make **minor changes inside get_response** to ensure it works with the history passed by Flask sessions (list of messages).

Replace your get_response function (near the end of file) with this improved version

```
def get_response(user_input, lang=None, history=None):
    """
    Fully conversational PDF QA with hybrid RAG (TF-IDF + embeddings).
    Handles follow-ups like 'what are the steps of this?' using context memory.
    Works properly with Flask session-based chat history.
    """
    if history is None:
        history = []

    # 1 Ensure training is loaded
    if not questions:
        load_and_train()

    # 2 Detect language
    lang = lang or ("punjabi" if re.search(r'[\u0A00-\u0A7F]', user_input) else "english")
    query = user_input if lang == "english" else translate_text(user_input, "en")
    user_norm = normalize_text(query)

    # 3 Recall topic context
    last_topic = None
    for past_user, past_bot in reversed(history):
        if len(past_user.split()) > 2:
            key_terms = re.findall(r'\b[a-zA-Z]{4,}\b', past_user)
            if key_terms:
                last_topic = " ".join(key_terms[-2:])
                break

    # 4 Follow-up handling
    follow_pattern = r'\b(this|that|it|those|these|ਉਹ|ਇਹ|steps|how)\b'
    if re.search(follow_pattern, query, re.I) and last_topic:
        query = f"{query} about {last_topic}"
        user_norm = normalize_text(query)

    # 5 Hybrid matching
    best_answer = None
    best_score = 0.0

    # --- TF-IDF ---
    try:
        query_vec = vectorizer.transform([user_norm])
        cosine_scores = cosine_similarity(query_vec, vectorizer.transform([normalize_text(q)
for q in questions]))[0]
        idx = cosine_scores.argmax()
        if cosine_scores[idx] > 0.35:
            best_answer = answers[idx]
            best_score = cosine_scores[idx]
    except Exception as e:
        if DEBUG: print("TF-IDF error:", e)

    # --- Embeddings fallback ---
    if not best_answer:
        try:
            from langchain_community.embeddings import HuggingFaceEmbeddings
            from sklearn.metrics.pairwise import cosine_similarity
            embedder = HuggingFaceEmbeddings(model_name="sentence-transformers/all-
MiniLM-L6-v2")
            q_embs = embedder.embed_documents(questions)
            query_emb = embedder.embed_query(user_norm)
            sims = cosine_similarity([query_emb], q_embs)[0]
            idx = sims.argmax()
            if sims[idx] > 0.35:
                best_answer = answers[idx]
                best_score = sims[idx]
        except Exception as e:
            if DEBUG: print("Embedding fallback error:", e)

    # --- RAG fallback ---
    if not best_answer:
        try:
            rag_ans = get_rag_response(query, history)
            if rag_ans and not rag_ans.lower().startswith(("sorry", "couldn't")):
                best_answer = rag_ans
        except Exception as e:
            if DEBUG: print("RAG error:", e)

    # --- Default fallback ---
    if not best_answer:
        best_answer = "Sorry, I don’t have an answer for that yet. Could you please clarify?"

    # --- Translate back if Punjabi ---
    if lang == "punjabi":
        best_answer = translate_text(best_answer, "pa")

    # --- Update session history (Flask session persists it) ---
    history.append((user_input, best_answer))
    if len(history) > 50:
        history[:] = history[-50:]

    return format_answer(best_answer)
```

---

app.py (UPDATED to support multiple users properly)

Replace your existing app.py completely with this:

```
from flask import Flask, render_template, request, jsonify, session
from flask_session import Session
from chatbot_logic import get_response, check_reload_needed, load_and_train
import uuid

app = Flask(__name__)
app.secret_key = "chatbot_secret_key"
app.config["SESSION_TYPE"] = "filesystem"
app.config["SESSION_PERMANENT"] = False
Session(app)

@app.before_request
def make_unique_session():
    """Assign a unique session ID to every new visitor."""
    if "user_id" not in session:
        session["user_id"] = str(uuid.uuid4())
        session["chat_history"] = []

@app.route("/")
```

```
def index():
    if "chat_history" not in session:
        session["chat_history"] = []
    return render_template("index.html")

@app.route("/get", methods=["POST"])
def chatbot_reply():
    data = request.get_json()
    user_message = data.get("msg", "").strip()
    user_lang = data.get("lang", "english")

    if not user_message:
        return jsonify({"response": "□ Please enter a message."})

    if check_reload_needed():
        load_and_train()

    # Load user-specific chat history
    history = session.get("chat_history", [])

    # Generate chatbot response
    response = get_response(user_message, user_lang, history)

    # Save updated chat history to this specific user's session
    session["chat_history"] = history
    session.modified = True  # ensure Flask saves the session

    return jsonify({"response": response})

if __name__ == "__main__":
    load_and_train()
    app.run(debug=True)
```

What This Fix Does

Unique chat session per user
Each user gets a user_id (UUID) and isolated chat history.

Persistent chat history within same browser
Session history is stored in filesystem backend — user’s chat persists until browser/session is cleared.

Multiple users work simultaneously
Different browsers or devices get separate chat histories (Flask sessions are independent).

Session is updated correctly each turn
session.modified = True ensures Flask writes updates to the session store every time.