DAY – 58, 59

23, 24 October 2025

Below are **three files** (complete) + a requirements.txt. Copy them into your project (keep the templates/ folder for index.html) and run. I made these changes to be resilient (safe fallbacks, proper session handling, clear logging) so you can verify behavior quickly.

---

1) chatbot_logic.py

- simplified/resilient load_and_train() so questions/answers are always non-empty (sample fallback Q/A).
- safe handling if vectorizer or tfidf missing.
- get_response() accepts the session history list and **updates it in-place** (so Flask session will persist changes).
- added simple logging prints to help debug.

```
# chatbot_logic.py
import os
import re
import unicodedata
import difflib
from collections import Counter

# lightweight imports for fallback behavior
try:
    import PyPDF2
except Exception:
    PyPDF2 = None

# minimal normalization
def normalize_text(t):
```

```
    if not t:
        return ""
    t = unicodedata.normalize("NFKC", t)
    t = t.lower()
    t = re.sub(r'[^a-z0-9\s]', ' ', t)
    t = re.sub(r'\s+', ' ', t).strip()
    return t

# very small question detector (used by PDF parser if present)
QUESTION_WORDS =
re.compile(r'^(?:what|how|why|when|where|which|is|are|can|should|do|does|explain|steps)',
re.I)

# Global containers
DATASET_PATH = "data/dataset.txt"
PDF_PATH = "data/PRSC.pdf"
questions = []
answers = []
vectorizer = None
tfidf_matrix = None

DEBUG = True

def robust_extract_pdf_text(pdf_path):
    if not PyPDF2:
        return ""
    if not os.path.exists(pdf_path):
        return ""
    try:
        text = []
        with open(pdf_path, "rb") as f:
            reader = PyPDF2.PdfReader(f)
            for p in reader.pages:
```

```
            t = p.extract_text() or ""
            text.append(t)
        return "\n".join(text)
    except Exception as e:
        if DEBUG: print("PDF extract error:", e)
        return ""

def parse_pdf_qa_simple(text):
    faqs = {}
    if not text:
        return faqs
    lines = [l.strip() for l in text.splitlines() if l.strip()]
    i = 0
    while i < len(lines):
        l = lines[i]
        if QUESTION_WORDS.match(l) or l.endswith("?"):
            q = l
            a_lines = []
            i += 1
            while i < len(lines) and not (QUESTION_WORDS.match(lines[i]) or
lines[i].endswith("?")):
                a_lines.append(lines[i])
                i += 1
            a = " ".join(a_lines).strip()
            if a:
                faqs[normalize_text(q)] = a
        else:
            i += 1
    return faqs

def load_and_train():
    """Load dataset.txt and PDF; ensure we have fallback QA if none present."""
    global questions, answers
```

```
faqs = {}

# 1) Load simple dataset file if exists (format: [english] then q = a)
if os.path.exists(DATASET_PATH):
    try:
        with open(DATASET_PATH, "r", encoding="utf-8") as f:
            lang = None
            for line in f:
                line = line.strip()
                if not line:
                    continue
                if line.startswith("[") and line.endswith("]"):
                    lang = line[1:-1].lower()
                    continue
                if "=" in line and lang == "english":
                    q, a = line.split("=", 1)
                    faqs[normalize_text(q)] = a.strip()
    except Exception as e:
        if DEBUG: print("dataset load error:", e)

# 2) Try extract from PDF
pdf_text = robust_extract_pdf_text(PDF_PATH)
parsed = parse_pdf_qa_simple(pdf_text)
faqs.update(parsed)

# 3) If still empty, provide fallback sample Q/A so chatbot always answers
if not faqs:
    faqs = {
        normalize_text("What is PRSC?"): "Punjab Remote Sensing Centre (PRSC) is an organisation which uses remote sensing and GIS for various applications.",
        normalize_text("How to use the user manual?"): "Open the manual, read the sections relevant to your workflow. Contact PRSC for support if unclear.",
```

normalize_text("What is e-Sinchai?"): "e-Sinchai appears to be a system described in the manual. Please provide specific page or section for more detail."
}

```
questions = list(faqs.keys())
answers = list(faqs.values())
if DEBUG:
    print(f"Loaded {len(questions)} Q/A entries.")

# Simple matching: exact normalized match -> else fuzzy match -> else default reply
def get_response(user_input, lang=None, history=None):
    """
    history: a list passed from Flask session. Example: [('hi','hello'), ...]
    This function will append (user_input, bot_answer) to history (in-place).
    """
    if history is None:
        history = []

    if not questions:
        load_and_train()

    user_norm = normalize_text(user_input)

    # 1) exact match
    if user_norm in questions:
        ans = answers[questions.index(user_norm)]
    else:
        # 2) fuzzy match on normalized keys
        matches = difflib.get_close_matches(user_norm, questions, n=1, cutoff=0.45)
        if matches:
            idx = questions.index(matches[0])
            ans = answers[idx]
        else:
```

```
        # 3) check if short follow-up pronoun and use last topic from history
        follow_pattern = r'\b(this|that|it|these|those|how|steps)\b'
        if re.search(follow_pattern, user_input, re.I) and history:
            # use last user message to build a new query attempt
            last_user = history[-1][0] if history else ""
            combined = f"{last_user} {user_input}"
            c_norm = normalize_text(combined)
            matches = difflib.get_close_matches(c_norm, questions, n=1, cutoff=0.45)
            if matches:
                idx = questions.index(matches[0])
                ans = answers[idx]
            else:
                ans = "Sorry, I don't have an exact answer. Please ask with more details or a
different phrase."
        else:
            ans = "Sorry, I don't have an exact answer. Please ask with more details or a
different phrase."

    # update session history (in-place)
    history.append((user_input, ans))
    if len(history) > 200:
        del history[:-200]

    if DEBUG:
        print("User:", user_input)
        print("Answer:", ans)
        print("History length:", len(history))

    return ans
```

2) app.py

- ensures session creation, sets SESSION_FILE_DIR to a local folder, marks session.modified = True.
- calls load_and_train() at startup to ensure QA loaded before first request.
- returns JSON responses so front-end can render.

```
# app.py
from flask import Flask, render_template, request, jsonify, session
from flask_session import Session
import os
import uuid
from chatbot_logic import get_response, load_and_train

app = Flask(__name__)
app.secret_key = "chatbot_secret_key_change_this"
app.config["SESSION_TYPE"] = "filesystem"
# ensure directory exists
SESSION_FILE_DIR = os.path.join(os.getcwd(), "flask_session_files")
os.makedirs(SESSION_FILE_DIR, exist_ok=True)
app.config["SESSION_FILE_DIR"] = SESSION_FILE_DIR
app.config["SESSION_PERMANENT"] = False
Session(app)

# warm start
load_and_train()

@app.before_request
def ensure_session():
    if "user_id" not in session:
        session["user_id"] = str(uuid.uuid4())
    if "chat_history" not in session:
        session["chat_history"] = []
        session.modified = True
```

```
@app.route("/")
def index():
    return render_template("index.html")

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

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

    # load user's history (this is a reference to session object)
    history = session.get("chat_history", [])
    # generate answer (function updates `history` in-place)
    answer = get_response(user_message, user_lang, history)

    # save back the modified history and mark session as modified so Flask persists it
    session["chat_history"] = history
    session.modified = True

    return jsonify({"response": answer})

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

3) templates/index.html

Simple front-end that sends messages to /get, shows chat and supports multiple messages. Use normal browser sessions or incognito to test multiple users.

```
<!-- templates/index.html -->
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Chatbot</title>
  <style>
    body { font-family: Arial, sans-serif; max-width: 800px; margin: 20px auto; }
    #chat { border: 1px solid #ddd; padding: 12px; height: 400px; overflow:auto;
background:#f9f9f9; }
    .user { color: #0b63b8; margin: 8px 0; }
    .bot { color: #222; margin: 8px 0; }
    #inputBox { width: 100%; padding: 8px; margin-top: 8px; }
    button { padding: 8px 12px; }
  </style>
</head>
<body>
  <h2>Chatbot</h2>
  <div id="chat"></div>

  <div>
    <input id="msg" placeholder="Type your message..." />
    <button onclick="send()">Send</button>
  </div>

  <script>
    async function send() {
      const msgEl = document.getElementById('msg');
      const text = msgEl.value.trim();
```

```
    if (!text) return;
    append('You: ' + text, 'user');
    msgEl.value = '';
    const res = await fetch('/get', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({msg: text, lang: 'english'})
    });
    const data = await res.json();
    append('Bot: ' + data.response, 'bot');
    // scroll
    const chat = document.getElementById('chat');
    chat.scrollTop = chat.scrollHeight;
  }

  function append(txt, cls) {
    const d = document.createElement('div');
    d.textContent = txt;
    d.className = cls;
    document.getElementById('chat').appendChild(d);
  }
 </script>
</body>
</html>
```

---

4) requirements.txt

Install with pip install -r requirements.txt.

flask
flask-session
PyPDF2

(If you want ML/text features later add scikit-learn, nltk, langchain etc. — I left them out to make the session+reply flow reliable.)

How to run

1. Put files in project:

```
project/
  app.py
  chatbot_logic.py
  templates/index.html
  requirements.txt
```

2. Create a virtualenv and install:

```
python -m venv venv
venv\Scripts\activate    # windows
# or: source venv/bin/activate
pip install -r requirements.txt
```

3. Run:

```
python app.py
```

4. Open http://127.0.0.1:5000/. Open a second browser or an incognito window to simulate a second user (each will have independent session history).

Troubleshooting checklist (if it still fails)

1. Confirm flask-session created flask_session_files directory and files inside — that proves sessions are being stored.
2. Test with two browsers (normal + incognito). They should have independent histories.
3. Add DEBUG = True in chatbot_logic.py to see printed logs in the console for each request.
4. If you use a reverse proxy or docker, ensure sticky sessions or filesystem session storage remains writable.
5. If using multiple workers (gunicorn), use Redis-based session store instead of filesystem for multi-process reliability.