backend/chat_api.py

```
# backend/chat_api.py
import os, json, time, uuid
from flask import Flask, request, jsonify
from flask_cors import CORS
from dotenv import load_dotenv

load_dotenv()

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
OPENAI_MODEL = os.getenv("OPENAI_MODEL", "gpt-4o-mini")
USE_OPENAI = bool(OPENAI_API_KEY)

if USE_OPENAI:
    import openai
    openai.api_key = OPENAI_API_KEY

app = Flask(__name__)
CORS(app)

# Persistent sessions stored in a JSON file (simple)
SESSION_DB = os.getenv("CHAT_SESSION_DB", "chat_sessions.json")

def load_sessions():
    try:
        with open(SESSION_DB, "r", encoding="utf-8") as f:
            return json.load(f)
    except Exception:
        return {}

def save_sessions(sessions):
    with open(SESSION_DB, "w", encoding="utf-8") as f:
        json.dump(sessions, f, indent=2, ensure_ascii=False)

SESSIONS = load_sessions()

SYSTEM_PROMPT = """You are EdgeGuard, a calm and helpful AI cybersecurity assistant for non-technical users.
Rules:
- Explain alerts in simple language.
- Give 3-6 short, actionable steps.
- NEVER ask for OTPs, passwords, PINs, CVV, or private credentials.
- If the user says they already connected to a suspicious network, give immediate steps (disconnect, forget network, ch
- If unsure, say so and provide safe conservative guidance."""

def call_llm(alert, history, user_text=None):
    messages = [{"role":"system","content":SYSTEM_PROMPT}]
    if alert:
        # Include alert context, but don't request secrets
        messages.append({
            "role":"user",
            "content":"Alert context (do not ask user for private credentials):\n```json\n" + json.dumps(alert, indent=
        })
    for m in history:
        messages.append({"role": m["role"], "content": m["content"]})
    if user_text:
        messages.append({"role":"user","content": user_text})

    if not USE_OPENAI:
        # Safe offline fallback
        return (
            "I am running in offline mode. Based on the alert, immediately disconnect from suspicious Wi-Fi, "
            "forget the network on your device, change your router password, run an antivirus scan, and do not share OT
            "If you tell me more about what happened, I will give more specific steps."
        )

    try:
        resp = openai.ChatCompletion.create(
            model=OPENAI_MODEL,
            messages=messages,
```

backend/chat_api.py (cont.)

```
            max_tokens=450,
            temperature=0.15
        )
        return resp.choices[0].message["content"].strip()
    except Exception as e:
        print("LLM error:", e)
        return ("AI service temporarily unavailable. Follow these safe steps: disconnect from suspicious Wi-Fi, forget 

@app.route("/api/chat/session", methods=["POST"])
def create_session():
    payload = request.get_json() or {}
    alert = payload.get("alert") or {}
    session_id = str(uuid.uuid4())
    initial = call_llm(alert, history=[], user_text="Please provide a short summary and 3 actionable steps for a non-te
    SESSIONS[session_id] = {
        "alert": alert,
        "history": [{"role":"assistant","content": initial}],
        "created": time.time()
    }
    save_sessions(SESSIONS)
    return jsonify({"session_id": session_id, "message": {"text": initial}})

@app.route("/api/chat/<session_id>/send", methods=["POST"])
def send(session_id):
    payload = request.get_json() or {}
    text = (payload.get("text") or "").strip()
    if not text:
        return jsonify({"error":"text required"}), 400
    session = SESSIONS.get(session_id)
    if not session:
        return jsonify({"error":"session not found"}), 404
    # add user message
    session["history"].append({"role":"user","content": text})
    reply = call_llm(session.get("alert"), session["history"], user_text=None)
    session["history"].append({"role":"assistant","content": reply})
    save_sessions(SESSIONS)
    return jsonify({"text": reply})

@app.route("/api/chat/<session_id>/history", methods=["GET"])
def history(session_id):
    s = SESSIONS.get(session_id)
    if not s:
        return jsonify({"error":"not found"}), 404
    return jsonify(s["history"])

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=int(os.getenv("CHAT_PORT", 5100)), debug=False)
```

frontend/src/components/ChatWidget.js

```
// frontend/src/components/ChatWidget.js
import React, { useState } from "react";
import "./ChatWidget.css";

export default function ChatWidget({ alert }) {
  const [sessionId, setSessionId] = useState(null);
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState("");
  const [typing, setTyping] = useState(false);

  const API_BASE = process.env.REACT_APP_API_URL || "";

  async function startSession() {
    try {
      const resp = await fetch(`${API_BASE}/api/chat/session`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ alert })
```

frontend/src/components/ChatWidget.js (cont.)

```
      });
      const j = await resp.json();
      setSessionId(j.session_id);
      setMessages([{ from: "bot", text: j.message.text }]);
    } catch (err) {
      setMessages([{ from: "bot", text: "Assistant failed to start. Please try again." }]);
    }
  }

  async function sendMessage() {
    if (!input.trim()) return;
    const text = input.trim();
    setInput("");
    setMessages(m => [...m, { from: "user", text }]);
    if (!sessionId) {
      await startSession();
    }
    setTyping(true);
    try {
      const r = await fetch(`${API_BASE}/api/chat/${sessionId}/send`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ text })
      });
      const j = await r.json();
      setMessages(m => [...m, { from: "bot", text: j.text }]);
    } catch (err) {
      setMessages(m => [...m, { from: "bot", text: "Assistant failed. Try again later." }]);
    } finally {
      setTyping(false);
    }
  }

  return (
    <div className="chat-widget">
      {!sessionId && (
        <div className="start-panel">
          <button className="start-btn" onClick={startSession}>Ask Assistant</button>
        </div>
      )}
      <div className="chat-window">
        <div className="messages">
          {messages.map((m, i) => (
            <div key={i} className={`msg ${m.from}`}>
              {m.text}
            </div>
          ))}
          {typing && <div className="msg bot">EdgeGuard is typing...</div>}
        </div>

        <div className="chat-input">
          <input
            value={input}
            onChange={(e) => setInput(e.target.value)}
            onKeyDown={(e) => e.key === "Enter" && sendMessage()}
            placeholder="Type your question..."
          />
          <button onClick={sendMessage}>Send</button>
        </div>
      </div>
    </div>
  );
}
```

frontend/src/components/ChatWidget.css

```
/* frontend/src/components/ChatWidget.css */
.chat-widget { width: 100%; max-width: 420px; border: 1px solid #263238; border-radius: 8px; background: #0f1720; color
```

frontend/src/components/ChatWidget.css (cont.)

```
.chat-window { display:flex; flex-direction: column; gap:8px; }
.messages { max-height: 320px; overflow:auto; padding:8px; background: #071026; border-radius:6px; }
.msg { padding:8px 10px; border-radius:8px; margin-bottom:6px; }
.msg.user { background: #0b5d8f; align-self: flex-end; color: #fff; }
.msg.bot { background: #10233a; align-self: flex-start; color: #dfeffd; }
.chat-input { display:flex; gap:8px; margin-top:6px; }
.chat-input input { flex:1; padding:8px; border-radius:6px; border:1px solid #213544; background:#071826; color:#e6eef8
.chat-input button { padding:8px 12px; border-radius:6px; background:#0b84ff; border:none; color:#fff; cursor:pointer;
.start-btn { padding:8px 12px; border-radius:6px; background:#0b84ff; border:none; color:#fff; cursor:pointer; }
```

backend/requirements_extra.txt

```
openai
flask-cors
python-dotenv
```