CardioSense AI Build Guide

Complete Step-by-Step — Every Command, Every File, Every Line

SPOON-FED EDITION · NOTHING LEFT OUT

Backend: FastAPI + Python | RAG: ChromaDB + Llama 3.2:3b | Frontend: React + Recharts | Hardware: Raspberry Pi 5 · 16GB

READ THIS BEFORE ANYTHING ELSE

> **Everything runs ON the Raspberry Pi 5.** Your laptop just opens a browser pointed at the Pi's IP. Both must be on the same Wi-Fi — use CometNet at UTD if you are a UTD student.

Your Pi should already have Raspbian, Python 3.10+, and Ollama with `llama3.2:3b` installed. Confirm Ollama works by opening a terminal on the Pi and typing `ollama run llama3.2:3b`. It should print a response. Type `/bye` to exit.

BUILD TIMELINE

HOURS	WHAT GETS BUILT	WHO
0 – 2	Folder setup, install packages, FastAPI skeleton running and responding	Backend
1 – 3	ChromaDB loaded with medical knowledge, RAG query tested and working	RAG Engineer
1 – 3	React app created, dark dashboard visible in browser with placeholders	Frontend

HOURS	WHAT GETS BUILT	WHO
2 – 5	Risk engine complete, Llama wired in, RAG connected to LLM service	Backend + RAG
5 – 8	End-to-end data flow working — simulator feeds data, dashboard updates	Everyone
8 – 12	Demo simulator finalized, frontend polished, all 5 tests passing	All
12 – 18	Full demo rehearsed, Devpost submitted, backup video recorded	All

PHASE 1 — PROJECT SETUP (EVERYONE TOGETHER · 15 MIN)

One person types on the Pi, everyone watches. Do this before anything else.

1.1 Create the Folder Structure

Open a terminal on the Pi and run:

```
mkdir cardiosense
cd cardiosense
mkdir backend frontend rag_pipeline tests
```

1.2 Install All Python Packages

From inside the `cardiosense` folder, run this single command:

```
pip install fastapi uvicorn chromadb sentence-transformers ollama pytest httpx requests
```

This takes 5–10 minutes on the Pi. Do not close the terminal. Let it finish completely before moving on.

1.3 Start Ollama (Leave This Running Always)

Open a **second terminal tab** on the Pi and run:

```
ollama serve
```

> **Leave this tab open for the entire hackathon.** Never close it. If Ollama stops, nothing works.

1.4 Find Your Pi's IP Address

```
hostname -I
```

> Write down the number that looks like **192.168.X.X** on a sticky note. Your frontend engineer will need it to connect their laptop to the Pi.

— PHASE 2 — BACKEND (BACKEND ENGINEER)

> **Your job:** Build the FastAPI server that receives vital signs, scores cardiac risk, calls Llama, and returns results. All files go in `cardiosense/backend/`

2.1 Create main.py

Navigate there first: `cd cardiosense/backend` — then create a new file called **main.py**:

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from risk_engine import calculate_risk
from llm_service import get_explanation
```

```
import uvicorn
from datetime import datetime

app = FastAPI()
app.add_middleware(CORSMiddleware, allow_origins=["*"],
    allow_methods=["*"], allow_headers=["*"])

history = []   # stores last 50 readings in memory

class Vitals(BaseModel):
    heart_rate: float
    spo2: float
    systolic_bp: float
    diastolic_bp: float
    hrv: float
    p_wave_present: bool
    t_wave_inverted: bool
    t_wave_peaked: bool
    rr_irregular: bool

@app.post("/vitals")
async def receive_vitals(v: Vitals):
    risk_level, score, flags = calculate_risk(v)
    explanation = get_explanation(v, risk_level, flags)
    result = {
        "timestamp": datetime.now().isoformat(),
        "vitals": v.dict(),
        "risk_level": risk_level,
        "risk_score": score,
        "flags": flags,
        "explanation": explanation
    }
    history.append(result)
    if len(history) > 50:
        history.pop(0)
    return result

@app.get("/history")
async def get_history():
    return history

@app.get("/health")
```

```
async def health():
    return {"status": "ok"}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

2.2 Create risk_engine.py

Create a second file called **risk_engine.py** in the same backend folder:

```
def calculate_risk(v):
    score = 0
    flags = []

    # --- ECG Waveform Checks (most clinically important) ---
    if not v.p_wave_present:
        score += 40
        flags.append("P-wave absent - possible arrhythmia or heart block")
    if v.t_wave_inverted:
        score += 30
        flags.append("T-wave inversion - possible myocardial ischemia")
    if v.t_wave_peaked:
        score += 25
        flags.append("T-wave peaked - possible hyperkalemia")
    if v.rr_irregular:
        score += 20
        flags.append("Irregular RR interval - arrhythmia marker")

    # --- Oxygen Saturation ---
    if v.spo2 < 90:
        score += 35
        flags.append("Critical hypoxia - SpO2 below 90%")
    elif v.spo2 < 94:
        score += 20
        flags.append("Low oxygen saturation")

    # --- Heart Rate ---
    if v.heart_rate > 130 or v.heart_rate < 45:
        score += 25
```

```
        flags.append("Extreme heart rate")
    elif v.heart_rate > 110 or v.heart_rate < 55:
        score += 12
        flags.append("Abnormal heart rate")

    # --- Blood Pressure ---
    if v.systolic_bp > 180 or v.systolic_bp < 85:
        score += 20
        flags.append("Critical blood pressure")
    elif v.systolic_bp > 150 or v.systolic_bp < 100:
        score += 10
        flags.append("Elevated blood pressure")

    # --- Heart Rate Variability ---
    if v.hrv < 10:
        score += 25
        flags.append("HRV near zero - autonomic failure risk")
    elif v.hrv < 25:
        score += 12
        flags.append("Low heart rate variability")

    # --- Combined Pre-Arrest Pattern (bonus score) ---
    if not v.p_wave_present and v.spo2 < 94 and v.hrv < 20:
        score += 20
        flags.append("COMBINED: Pre-cardiac-arrest pattern detected")

    # --- Map score to triage level ---
    if score ⩾ 76:
        level = "CRITICAL"
    elif score ⩾ 46:
        level = "MODERATE"
    elif score ⩾ 21:
        level = "MILD"
    else:
        level = "NORMAL"

    return level, score, flags
```

2.3 Create llm_service.py (basic version first)

Create a third file called **llm_service.py**. You will upgrade this in Phase 3 after RAG is ready:

```
import subprocess

def get_explanation(vitals, risk_level, flags):
    flags_text = ". ".join(flags) if flags else "No abnormalities detected"

    prompt = f"""You are a cardiac triage assistant.
Risk level detected: {risk_level}
Issues found: {flags_text}
Patient vitals: HR={vitals.heart_rate}bpm, SpO2={vitals.spo2}%,
BP={vitals.systolic_bp}/{vitals.diastolic_bp}mmHg, HRV={vitals.hrv}ms

Write exactly 2 simple sentences:
1) What was detected in plain language.
2) What the patient must do right now.
No medical jargon. Be calm but direct."""

    try:
        result = subprocess.run(
            ["ollama", "run", "llama3.2:3b", prompt],
            capture_output=True, text=True, timeout=30
        )
        return result.stdout.strip()
    except Exception:
        fallbacks = {
            "NORMAL":   "All your vital signs are within healthy ranges. Continue your normal activities."
            "MILD":     "Minor irregularities detected in your heart signals. Rest, drink water, and reche
            "MODERATE": "Your heart signals show patterns that need medical attention. Contact your doctor
            "CRITICAL": "A dangerous cardiac pattern has been detected. Call 911 immediately."
        }
        return fallbacks.get(risk_level, "Please seek medical attention.")
```

2.4 Start the Backend Server

```
cd cardiosense/backend
```

```
python main.py
```

Success looks like: INFO: Uvicorn running on http://0.0.0.0:8000
Test it: Open a browser on your laptop, go to `http://YOUR_PI_IP:8000/health` — you should see `{"status": "ok"}`

PHASE 3 — RAG PIPELINE (RAG ENGINEER)

Your job: Build the medical knowledge retrieval system that gives Llama real clinical facts. All files go in `cardiosense/rag_pipeline/`

3.1 Create medical_knowledge.txt

Create a file called **medical_knowledge.txt** and paste ALL of this text into it:

```
P-WAVE FACTS:
The P-wave represents electrical activation of the atria (upper heart chambers).
Absent P-waves indicate atrial fibrillation, junctional rhythm, or AV block.
Missing P-waves with irregular RR intervals strongly indicate atrial fibrillation.
Complete heart block presents with absent or dissociated P-waves from QRS complexes.
P-wave absence is a critical warning that the heart electrical conduction is compromised.

T-WAVE FACTS:
T-waves represent ventricular repolarization - the heart resetting after each beat.
Inverted T-waves indicate myocardial ischemia - the heart muscle lacks sufficient oxygen.
T-wave inversion in multiple leads suggests a possible heart attack in progress.
Peaked or tall T-waves are associated with hyperkalemia disrupting heart rhythm.
T-wave changes combined with chest symptoms require immediate emergency evaluation.

OXYGEN SATURATION (SpO2):
Normal SpO2 is 95-100%. Below 90% is a medical emergency called hypoxia.
SpO2 between 90-94% indicates mild hypoxia requiring immediate attention.
Hypoxia combined with tachycardia means the heart is overcompensating for low oxygen.
Low SpO2 deprives the heart of oxygen needed to maintain normal rhythm.
Hypoxia is a direct cause of cardiac arrhythmias and can trigger cardiac arrest.
```

```
HEART RATE VARIABILITY (HRV):
HRV measures variation in time between heartbeats in milliseconds.
Healthy HRV ranges from 20-70ms depending on age and fitness level.
HRV below 10ms indicates severely compromised autonomic nervous system function.
Collapsing HRV over minutes is a pre-cardiac-arrest warning sign.
Low HRV combined with arrhythmia markers indicates very high cardiac event risk.

CARDIAC ARREST WARNING SIGNS:
Absent P-waves plus T-wave inversion plus collapsing HRV is a pre-arrest pattern.
Early detection of pre-arrest patterns allows 10-15 minutes for intervention.
If SpO2 drops below 88% with simultaneous ECG abnormalities, call 911 immediately.
Cardiac arrest survival rates drop 10 percent for every minute without intervention.

BLOOD PRESSURE:
Normal blood pressure is 120/80 mmHg.
Hypertensive crisis is systolic above 180 mmHg and requires emergency care.
Hypotension with systolic below 90 combined with tachycardia indicates shock.
Blood pressure changes combined with ECG abnormalities indicate high cardiac risk.
```

3.2 Create and Run setup_rag.py (Run This Once)

Create **setup_rag.py** in the rag_pipeline folder:

```
import chromadb
from chromadb.utils import embedding_functions

client = chromadb.PersistentClient(path="./chroma_db")
ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)
collection = client.get_or_create_collection(
    name="cardiac_knowledge", embedding_function=ef
)

with open("medical_knowledge.txt", "r") as f:
    content = f.read()

chunks = [c.strip() for c in content.split("\n\n") if len(c.strip()) > 30]
```

```
collection.add(
    documents=chunks,
    ids=[f"chunk_{i}" for i in range(len(chunks))]
)
print(f"Loaded {len(chunks)} knowledge chunks. RAG database ready!")
```

Run it once:

```
cd cardiosense/rag_pipeline
python setup_rag.py
```

> You should see: **"Loaded X knowledge chunks. RAG database ready!"** — if yes, your vector database is built.

3.3 Create rag_query.py

```
import chromadb
from chromadb.utils import embedding_functions

_collection = None

def get_collection():
    global _collection
    if _collection is None:
        client = chromadb.PersistentClient(path="./chroma_db")
        ef = embedding_functions.SentenceTransformerEmbeddingFunction(
            model_name="all-MiniLM-L6-v2"
        )
        _collection = client.get_collection(
            name="cardiac_knowledge", embedding_function=ef
        )
    return _collection

def retrieve_context(query: str, n_results: int = 3) -> str:
    results = get_collection().query(
        query_texts=[query], n_results=n_results
    )
```

```
    return "\n".join(results['documents'][0])

if __name__ == "__main__":
    ctx = retrieve_context("P-wave absent irregular heartbeat")
    print("RAG Test OK - Retrieved:")
    print(ctx[:400])
```

Test it:

```
python rag_query.py
```

> You should see medical text about P-waves printed. If yes, RAG is working perfectly.

3.4 Upgrade llm_service.py to Use RAG

Go back to `backend/llm_service.py` and **replace the entire file** with this:

```
import subprocess, sys
sys.path.append('../rag_pipeline')
from rag_query import retrieve_context

def get_explanation(vitals, risk_level, flags):
    query = " ".join(flags) if flags else "normal cardiac reading"
    medical_context = retrieve_context(query)
    flags_text = ". ".join(flags) if flags else "No abnormalities"

    prompt = f"""You are a cardiac triage assistant.
Use the medical knowledge below to explain in exactly 2 simple sentences.

MEDICAL KNOWLEDGE:
{medical_context}

PATIENT DATA:
Risk Level: {risk_level}
Detected Issues: {flags_text}
Vitals: HR={vitals.heart_rate}bpm, SpO2={vitals.spo2}%,
BP={vitals.systolic_bp}/{vitals.diastolic_bp}mmHg, HRV={vitals.hrv}ms
```

```
Write 2 sentences only.
Sentence 1: what was detected.
Sentence 2: what the patient must do right now.
Use plain simple words. No abbreviations."""

    try:
        result = subprocess.run(
            ["ollama", "run", "llama3.2:3b", prompt],
            capture_output=True, text=True, timeout=30
        )
        return result.stdout.strip()
    except Exception:
        fallbacks = {
            "NORMAL":   "All your vital signs are within healthy ranges. Continue normal activities.",
            "MILD":     "Minor irregularities detected. Rest, hydrate, recheck in 10 minutes.",
            "MODERATE": "Heart signals need medical attention. Contact your doctor now.",
            "CRITICAL": "Dangerous cardiac pattern detected. Call 911 immediately."
        }
        return fallbacks.get(risk_level, "Please seek medical attention.")
```

PHASE 4 — FRONTEND (FRONTEND ENGINEER · ON YOUR LAPTOP)

Your job: Build the dashboard web app on YOUR LAPTOP — not the Pi. It talks to the Pi over Wi-Fi. Make sure both are on the same network.

4.1 Create React App

```
npx create-react-app cardiosense-frontend
cd cardiosense-frontend
npm install recharts axios
```

4.2 Create .env File

In the `cardiosense-frontend` folder, create a file called **.env**:

```
REACT_APP_API_URL=http://192.168.X.X:8000
```

> Replace `192.168.X.X` with your actual Pi IP from Step 1.4. Without this the app cannot talk to the backend.

4.3 Replace src/App.js

Delete **all content** in `src/App.js` and paste this complete dashboard:

```
import React, { useState, useEffect, useRef } from 'react';
import { LineChart, Line, XAxis, YAxis, Tooltip, ResponsiveContainer } from 'recharts';
import axios from 'axios';

const API = process.env.REACT_APP_API_URL;
const COLORS = { NORMAL:'#27AE60', MILD:'#F39C12', MODERATE:'#E67E22', CRITICAL:'#C0392B' };
const ICONS  = { NORMAL:'🟢', MILD:'🟡', MODERATE:'🟠', CRITICAL:'🔴' };

export default function App() {
  const [current, setCurrent]   = useState(null);
  const [history, setHistory]   = useState([]);
  const [wave,    setWave]      = useState([]);
  const wRef = useRef(0);

  // Animated ECG waveform
  useEffect(() => {
    const id = setInterval(() => {
      wRef.current += 0.3;
      const t = wRef.current;
      let v = Math.sin(t) * 0.1;
      if (Math.sin(t * 2) > 0.95)  v = 1.2;   // R spike
      if (Math.sin(t * 2 - 0.3) > 0.9) v = -0.3; // S wave
      setWave(prev => [...prev, { t: t.toFixed(1), v: v.toFixed(2) }].slice(-60));
    }, 50);
    return () => clearInterval(id);
  }, []);
```

```
  // Poll backend every 3 seconds
  useEffect(() => {
    const poll = async () => {
      try {
        const res = await axios.get(`${API}/history`);
        if (res.data.length > 0) {
          setCurrent(res.data[res.data.length - 1]);
          setHistory(res.data.slice(-20).map((d, i) => ({
            i, score: d.risk_score, hr: d.vitals.heart_rate, spo2: d.vitals.spo2
          })));
        }
      } catch(e) { console.log('Waiting for backend...'); }
    };
    poll();
    const id = setInterval(poll, 3000);
    return () => clearInterval(id);
  }, []);

  const risk  = current?.risk_level || 'NORMAL';
  const color = COLORS[risk];

  return (
    <div style={{ background:'#0A0A0A', minHeight:'100vh', padding:'20px',
      fontFamily:'monospace', color:'white' }}>
      <div style={{ maxWidth:'1100px', margin:'0 auto' }}>

        <div style={{ textAlign:'center', marginBottom:'20px' }}>
          <h1 style={{ color:'#C0392B', fontSize:'2rem', margin:0 }}>CardioSense AI</h1>
          <p style={{ color:'#888', margin:'4px 0' }}>Real-Time Cardiac Risk Monitoring</p>
        </div>

        <div style={{ background:color, borderRadius:'12px', padding:'20px',
          textAlign:'center', marginBottom:'20px', boxShadow:`0 0 30px ${color}55` }}>
          <div style={{ fontSize:'3rem' }}>{ICONS[risk]}</div>
          <h2 style={{ margin:'6px 0', fontSize:'2rem' }}>{risk} RISK</h2>
          <p style={{ margin:0, opacity:0.9 }}>Score: {current?.risk_score || 0} / 100</p>
        </div>

        {current && (
          <div style={{ display:'grid', gridTemplateColumns:'repeat(4,1fr)',
            gap:'12px', marginBottom:'20px' }}>
            {[
```

```
      { label:'Heart Rate',     val:`${current.vitals.heart_rate} bpm` },
      { label:'SpO2',           val:`${current.vitals.spo2}%` },
      { label:'Blood Pressure', val:`${current.vitals.systolic_bp}/${current.vitals.diastolic_bp}`
      { label:'HRV',            val:`${current.vitals.hrv} ms` },
    ].map(({ label, val }) => (
      <div key={label} style={{ background:'#1A1A1A', borderRadius:'8px',
        padding:'16px', textAlign:'center', border:`1px solid ${color}44` }}>
        <div style={{ color:'#888', fontSize:'0.75rem', marginBottom:'4px' }}>{label}</div>
        <div style={{ fontSize:'1.4rem', fontWeight:'bold', color }}>{val}</div>
      </div>
    ))}
  </div>
)}

<div style={{ display:'grid', gridTemplateColumns:'1fr 1fr', gap:'16px', marginBottom:'20px' }}>
  <div style={{ background:'#1A1A1A', borderRadius:'8px', padding:'16px', border:'1px solid #333'
    <h3 style={{ color:'#27AE60', margin:'0 0 10px', fontSize:'0.85rem' }}>LIVE ECG WAVEFORM</h3>
    <ResponsiveContainer width="100%" height={110}>
      <LineChart data={wave}>
        <Line type="monotone" dataKey="v" stroke="#27AE60"
          dot={false} strokeWidth={1.5} isAnimationActive={false}/>
        <YAxis domain={[-0.5, 1.5]} hide />
        <XAxis dataKey="t" hide />
      </LineChart>
    </ResponsiveContainer>
  </div>
  <div style={{ background:'#1A1A1A', borderRadius:'8px', padding:'16px', border:'1px solid #333'
    <h3 style={{ color:'#C0392B', margin:'0 0 10px', fontSize:'0.85rem' }}>ECG STATUS FLAGS</h3>
    {current && [
      { label:'P-Wave',      ok: current.vitals.p_wave_present },
      { label:'T-Wave',      ok: !current.vitals.t_wave_inverted && !current.vitals.t_wave_peaked
      { label:'RR Interval', ok: !current.vitals.rr_irregular },
    ].map(({ label, ok }) => (
      <div key={label} style={{ display:'flex', justifyContent:'space-between',
        marginBottom:'8px', padding:'7px 10px', borderRadius:'4px',
        background: ok ? '#1B2E1B' : '#2E1B1B' }}>
        <span style={{ fontSize:'0.85rem' }}>{label}</span>
        <span style={{ color: ok ? '#27AE60' : '#C0392B', fontWeight:'bold', fontSize:'0.8rem' }}>
          {ok ? 'NORMAL' : 'ABNORMAL'}
        </span>
      </div>
    ))}
```

```
  </div>
</div>

{current?.explanation && (
  <div style={{ background:'#1A1A2E', borderRadius:'8px', padding:'20px',
    marginBottom:'20px', border:`2px solid ${color}55` }}>
    <h3 style={{ color:'#7986CB', margin:'0 0 10px', fontSize:'0.85rem' }}>
      AI CARDIAC ANALYSIS — Powered by Llama 3.2 + Medical RAG
    </h3>
    <p style={{ margin:0, lineHeight:'1.7', color:'#DDD', fontSize:'0.95rem' }}>
      {current.explanation}
    </p>
  </div>
)}

{current?.flags?.length > 0 && (
  <div style={{ background:'#1A1A1A', borderRadius:'8px', padding:'16px',
    marginBottom:'20px', border:'1px solid #C0392B33' }}>
    <h3 style={{ color:'#C0392B', margin:'0 0 10px', fontSize:'0.85rem' }}>DETECTED ANOMALIES</h3>
    {current.flags.map((f, i) => (
      <div key={i} style={{ padding:'6px 10px', marginBottom:'4px',
        background:'#2E1B1B', borderRadius:'4px', fontSize:'0.85rem',
        borderLeft:'3px solid #C0392B' }}>⚠ {f}</div>
    ))}
  </div>
)}

{history.length > 1 && (
  <div style={{ background:'#1A1A1A', borderRadius:'8px', padding:'16px', border:'1px solid #333'
    <h3 style={{ color:'#888', margin:'0 0 10px', fontSize:'0.85rem' }}>RISK SCORE HISTORY</h3>
    <ResponsiveContainer width="100%" height={90}>
      <LineChart data={history}>
        <Line type="monotone" dataKey="score" stroke="#C0392B" dot={false} strokeWidth={2}/>
        <YAxis domain={[0,100]} hide />
        <Tooltip contentStyle={{ background:'#222', border:'none', fontSize:'12px' }}/>
      </LineChart>
    </ResponsiveContainer>
  </div>
)}

{!current && (
  <div style={{ textAlign:'center', padding:'40px', color:'#555' }}>
```

```
            Waiting for vital signs from backend...<br/>
            Make sure the backend is running and the demo simulator is sending data.
          </div>
        )}
      </div>
    </div>
  );
}
```

4.4 Start the Frontend

```
npm start
```

> A browser opens at **http://localhost:3000**. You will see the dark dashboard. It shows "Waiting for data" until the backend receives vitals — that is expected.

PHASE 5 — DEMO SIMULATOR (BACKEND + TESTING ENGINEER)

> **This is your WOW moment.** This script sends vitals that start normal and escalate to CRITICAL over 3 minutes. This is exactly what you run during the judge demo.

5.1 Create demo_simulator.py

Create this file in the main `cardiosense/` folder:

```
import requests, time

API = "http://localhost:8000/vitals"
# If running from your laptop: API = "http://YOUR_PI_IP:8000/vitals"
```

```
demo_sequence = [
    # NORMAL (readings 1-2)
    {"heart_rate":72,"spo2":98,"systolic_bp":120,"diastolic_bp":80,
     "hrv":55,"p_wave_present":True,"t_wave_inverted":False,"t_wave_peaked":False,"rr_irregular":False},
    {"heart_rate":74,"spo2":97,"systolic_bp":122,"diastolic_bp":80,
     "hrv":52,"p_wave_present":True,"t_wave_inverted":False,"t_wave_peaked":False,"rr_irregular":False},
    # MILD (readings 3-4)
    {"heart_rate":88,"spo2":95,"systolic_bp":132,"diastolic_bp":85,
     "hrv":35,"p_wave_present":True,"t_wave_inverted":False,"t_wave_peaked":False,"rr_irregular":True},
    {"heart_rate":95,"spo2":94,"systolic_bp":138,"diastolic_bp":88,
     "hrv":26,"p_wave_present":True,"t_wave_inverted":False,"t_wave_peaked":False,"rr_irregular":True},
    # MODERATE (readings 5-6)
    {"heart_rate":108,"spo2":92,"systolic_bp":148,"diastolic_bp":93,
     "hrv":16,"p_wave_present":False,"t_wave_inverted":False,"t_wave_peaked":False,"rr_irregular":True},
    {"heart_rate":115,"spo2":91,"systolic_bp":155,"diastolic_bp":96,
     "hrv":12,"p_wave_present":False,"t_wave_inverted":True,"t_wave_peaked":False,"rr_irregular":True},
    # CRITICAL (readings 7-8)
    {"heart_rate":128,"spo2":88,"systolic_bp":168,"diastolic_bp":102,
     "hrv":6,"p_wave_present":False,"t_wave_inverted":True,"t_wave_peaked":True,"rr_irregular":True},
    {"heart_rate":138,"spo2":85,"systolic_bp":178,"diastolic_bp":106,
     "hrv":3,"p_wave_present":False,"t_wave_inverted":True,"t_wave_peaked":True,"rr_irregular":True},
]

print("CardioSense AI Demo Starting...")
print("Watch browser at http://localhost:3000")
print("-" * 50)

for i, vitals in enumerate(demo_sequence):
    try:
        r = requests.post(API, json=vitals, timeout=10)
        d = r.json()
        print(f"Reading {i+1}: {d['risk_level']} | Score: {d['risk_score']}")
        if d['flags']:
            print(f"  Flag: {d['flags'][0]}")
        print(f"  AI: {d['explanation'][:80]}...")
        print()
    except Exception as e:
        print(f"Error: {e} — is the backend running?")
    time.sleep(15)  # 15 seconds between readings
```

```
print("Demo complete!")
```

5.2 Run the Demo

```
python demo_simulator.py
```

> The dashboard will update every 15 seconds. Colors escalate: **GREEN** → **YELLOW** → **ORANGE** → **RED**. Practice narrating this 2–3 times before judging.

PHASE 6 — TESTING (TESTING ENGINEER)

6.1 Create test_risk_engine.py

Create in `cardiosense/tests/` folder:

```
import sys, pytest
sys.path.append('../backend')
from risk_engine import calculate_risk

class V:
    def __init__(self, **kw): self.__dict__.update(kw)

def normal():
    return V(heart_rate=72, spo2=98, systolic_bp=120, diastolic_bp=80,
             hrv=55, p_wave_present=True, t_wave_inverted=False,
             t_wave_peaked=False, rr_irregular=False)

def test_normal_vitals():
    level, score, _ = calculate_risk(normal())
    assert level == "NORMAL" and score < 21
```

```
def test_missing_p_wave():
    v = normal(); v.p_wave_present = False
    level, score, flags = calculate_risk(v)
    assert score ⩾ 40
    assert any("P-wave" in f for f in flags)

def test_critical_hypoxia():
    v = normal(); v.spo2 = 85
    level, _, _ = calculate_risk(v)
    assert level in ["MODERATE", "CRITICAL"]

def test_full_pre_arrest():
    v = V(heart_rate=130, spo2=86, systolic_bp=170, diastolic_bp=105,
          hrv=5, p_wave_present=False, t_wave_inverted=True,
          t_wave_peaked=True, rr_irregular=True)
    level, score, _ = calculate_risk(v)
    assert level == "CRITICAL" and score ⩾ 76

def test_t_wave_flagged():
    v = normal(); v.t_wave_inverted = True
    _, score, flags = calculate_risk(v)
    assert any("T-wave" in f for f in flags)
    assert score ⩾ 30
```

6.2 Run Tests

```
cd cardiosense/tests
python -m pytest test_risk_engine.py -v
```

> All 5 tests must show **PASSED**. If any fail, fix the corresponding logic in `risk_engine.py` .

6.3 Pre-Judging System Checklist

Complete every item at least **1 hour before judging**:

- [ ] Backend running: `http://PI_IP:8000/health` shows `{"status":"ok"}`
- [ ] Ollama running: separate terminal has `ollama serve` running — never closed
- [ ] RAG database: `rag_pipeline/chroma_db/` folder exists and is not empty
- [ ] Frontend visible: dark dashboard loads at `http://localhost:3000`
- [ ] Demo simulator works: run it, dashboard updates every 15 seconds
- [ ] Colors escalate correctly: GREEN → YELLOW → ORANGE → RED during demo run
- [ ] AI explanation appears: blue text box shows explanation text after each reading
- [ ] ECG flags show ABNORMAL in red during moderate and critical phases
- [ ] History chart fills in at bottom as readings come in
- [ ] All 5 unit tests pass: pytest shows 5 passed, 0 failed
- [ ] Demo video recorded as backup (2-min screen capture of full green-to-red escalation)
- [ ] Devpost team registered by 11PM Saturday

PHASE 7 — PITCH & DEVPOST SUBMISSION

Your 30-second pitch — memorize this word for word:

> *Every year thousands die from cardiac events that could have been prevented. Wearables show you numbers — but numbers without interpretation are useless. CardioSense AI is a digital cardiac triage nurse. It watches your heart rate, blood oxygen, blood pressure, and ECG waveform — specifically detecting missing P-waves and T-wave inversions, the exact same markers emergency doctors check first. Instead of showing HR equals 118, it tells you: call 911 now. Everything runs on a Raspberry Pi — no cloud, total patient privacy. Let me show you the live demo.*

How to narrate the demo to judges:

- Point at green badge: *"Patient vitals look completely normal. HR 72, SpO_2 98%. Nothing alarming."*
- Yellow appears: *"Watch — HRV is dropping and RR interval just became irregular. Mild risk detected."*
- Orange appears: *"P-wave just disappeared from the ECG. That means the atria may have stopped coordinating normally. The AI just told the patient to call a doctor immediately."*
- Red appears: *"T-wave inversion, SpO_2 at 88%, HRV near zero. Critical. The AI generated this explanation from real medical guidelines — not guesswork."*
- Closing: *"The patient still feels completely fine. CardioSense AI detected this pre-arrest pattern 15 minutes before any symptoms. That is the difference between life and death."*

Devpost Submission Checklist:

- ☐ Register team on Devpost by **11:00 PM Saturday** — go to Axxess Devpost page
- ☐ Project name: **CardioSense AI**
- ☐ Elevator pitch: "Real-time cardiac risk monitoring that detects pre-arrest ECG patterns before symptoms appear"
- ☐ Track: **AI-Driven Preventive Health Partner**
- ☐ Invite ALL teammates to the project (required for prize eligibility)
- ☐ Add GitHub repository link — must be **PUBLIC** (private repos get disqualified)
- ☐ Upload 2-minute demo video to YouTube, add link to Devpost
- ☐ Add screenshots: one at each risk level (green, yellow, orange, red)
- ☐ Write project story: problem → solution → P-wave/T-wave innovation → RAG pipeline → tech stack
- ☐ **Final submission by 10:30 AM Sunday** — no edits after that

> **If the live demo crashes during judging:** Stay calm. Play your pre-recorded backup video. Explain what happened and what you would fix with more time. Judges respect composure and technical honesty — a calm explanation of a failure beats a panicked live debug every time.

You have a strong concept, complete code, and a compelling demo.
Execute cleanly and you win. GO GET IT.