Smart Patient Vitals Monitoring Watch with Wireless Data Logging and Real-Time Dashboard Display

Submitted to: **Engr. Levi A. Corvera, MIT**

Submitted by: **Kurt Ronnel Chua, Morgado Angelo, Dwight Lamoste**

Smart Patient Vitals Monitoring Watch with WiFi-Enabled Wearable (ESP32-C3 + MAX3010x Overview)

This Smart Patient Vitals Monitoring Watch enables real-time monitoring of key vital signs using an ESP32-C3 microcontroller and a MAX3010x pulse oximetry sensor. The wearable collects physiological data such as heart rate (BPM), estimated oxygen saturation (SpO_2), and infrared/red light readings used for pulse detection. An estimated body temperature reading may also be included depending on sensor support and calibration.

The ESP32-C3 provides built-in WiFi connectivity, allowing the watch to transmit readings wirelessly to a Django-based web server through an API endpoint. Each measurement is stored automatically in a database for logging and later review. A real-time dashboard displays the latest values and trends using live charts, making it useful for continuous monitoring during testing and demonstrations.

For usability, the device includes a WiFi configuration portal (captive setup mode) so it can be connected to any available network without reprogramming. It also supports a reset WiFi feature for quick reconfiguration. With wireless data logging and a clean monitoring interface, the system demonstrates a low-cost approach to remote patient vital signs tracking and real-time visualization.

Component	Qty	Description
ESP32-C3 Mini	1	Main microcontroller (WiFi + processing + posts data to Django)
MAX30102 Pulse Oximeter Sensor	1	Measures IR/Red signals for BPM and SpO_2 estimation
MAX30102 Pulse Oximeter Sensor	1	Measures IR/Red signals for BPM and SpO_2 estimation
Li-ion/LiPo Battery (single cell)	1	Charges a single-cell Li-ion/LiPo battery via USB (often includes protection on some boards)

SYSTEM LOGIC (HOW IT WORKS)

1. **Boot + load saved settings**
   - The esp32 code loads saved **device config** from NVS (Preferences) under "dev":
     - **cfgIngestUrl, cfgDeviceId, cfgDeviceName, cfgPatientCode (X-PUBLIC-CODE)**
   - It also loads saved **WiFi SSID/PASS** from NVS under "wifi" (if available).
2. **WiFi connect OR start captive portal**
   - If saved WiFi exists and connects successfully:
     - The esp32 code connects as **STA mode** and starts a small **STA status/config page** at:
       - http://<ESP32_LOCAL_IP>/
   - If WiFi is not saved / fails to connect:
     - The esp32 code starts a **captive portal Access Point**:
       - SSID: **C3-Config**
       - AP IP: **192.168.4.1**
       - DNS redirects most requests to the portal page
     - User enters:
       - WiFi SSID + password
       - ingest URL
       - device info
       - patient code
     - If WiFi connects successfully → esp32 code saves everything to NVS → reboots.
3. **Sensor initialization (MAX3010x)**
   - After WiFi STA is ready, the esp32 code waits ~2 seconds, then initializes the MAX3010x over I2C:
     - SDA = 5, SCL = 6 (Fast I2C)
     - Red + IR mode
     - sampleRate = 100, pulseWidth = 411, adcRange = 16384

- If the sensor init fails:
  - sensorReady = false
  - vitals won't be processed, and the posting task won't send readings.

4. **Continuous sampling (IR + Red)**
   - The esp32 code continuously reads from the MAX3010x FIFO:
     - Updates currentIr and currentRed per sample.
5. **Finger detection (hysteresis thresholds)**
   - Finger presence is detected using IR hysteresis:
     - Finger turns **ON** when IR > 9000
     - Finger turns **OFF** when IR < 7000
   - hasFinger becomes the live finger state.
6. **Temperature reading (once per second)**
   - Every ~1 second, the esp32 code calls sensor.readTemperature().
   - Temperature becomes valid only if:
     - finger is present, **and**
     - temperature is between **20°C and 45°C**
   - Otherwise temperature is marked invalid and sent as **null**.
7. **No-finger behavior (delayed reset)**
   - If no finger is detected, the esp32 code does **not** instantly clear BPM/SpO_2.
   - It waits until finger has been absent continuously for **> 1500 ms**, then resets:
     - beat detector + beat history
     - SpO_2/PI window buffers
     - validity flags (hasValidBpm, hasValidSpO2, etc.)
8. **When finger is present: auto-gain + BPM detection**
   - **Auto-gain** adjusts IR/Red LED amplitudes about once per second to avoid saturation/clipping.
   - Beat detection runs on IR using:
     - DC removal (EMA)
     - polarity learning (handles upward/downward pulse shapes)
     - adaptive threshold + re-arm logic
   - BPM becomes valid once enough beat timing exists to compute a BPM interval.
     - (hasValidBpm becomes true once it can compute one valid beat interval)
9. **SpO_2 + PI calculation (window-based)**
   - The esp32 code fills a **50-sample window** of IR/Red.
   - Once full, it computes:
     - **PI** from IR AC/DC (clamped 0–20)
     - **SpO_2 estimate** using Red/IR AC/DC ratio (clamped 70–100)
   - Until the window fills, SpO_2 is invalid and sent as **null**.
10. **RR + BP estimates (only if BPM is valid)**

- RR and BP are **derived estimates**, not direct measurements.
- If hasValidBpm == true:
  - RR is computed from BPM and clamped **8–30**
  - SBP/DBP computed from BPM and clamped:
    - SBP: 90–200
    - DBP: 50–130
- If BPM is not valid:
  - RR/SBP/DBP are sent as **null**.

11. **Posting to Django (separate task, every ~1 second)**

- A FreeRTOS task posts roughly every **1000 ms** when:
  - not in config mode
  - sensorReady == true
  - WiFi is connected
- Authorization headers used by the esp32 code:
  - X-PUBLIC-CODE: <cfgPatientCode> (**required**, otherwise it won't post)
  - optional: X-DEVICE-NAME
- JSON payload includes:
  - device_id, ir, red, finger
  - bpm and hr (same value)
  - spo2, pi, rr, sbp, dbp, bp, temp
- Invalid values are sent as **null** (not "N/A").

12. **Dashboard behavior (Django side)**

- The esp32 code's local web server is mainly for **WiFi + device config/status**.
- Django stores incoming readings and the dashboard pulls the latest values from Django endpoints to display them.

ESP32-C3 WIFI INTEGRATION

(CAPTIVE PORTAL + HTTPS POST)

The ESP32-C3 reads IR/Red/Temp from the MAX3010x sensor, computes vitals, then uploads the latest readings to the Django backend over WiFi.

Hardware Wiring (Power + I²C)

Component	Connects To	Function
ESP32-C3 **3V3**	ESP32-C3 **GPIO6 (SCL)**	Provides 3.3V power to sensor
ESP32-C3 **GND**	MAX3010x **GND**	Common ground (required)
ESP32-C3 **GPIO5 (SDA)**	MAX3010x **SDA**	I²C data line
ESP32-C3 **GPIO6 (SCL)**	MAX3010x SCL	I²C clock line

WiFi Setup (Saved WiFi + Captive Portal)

The ESP32 switches between **normal WiFi mode (STA)** when saved credentials work and **setup mode (AP/captive portal)** when WiFi is missing or fails, so the user can configure the connection and backend details.

Mode	Trigger	Result
STA (normal mode)	Saved SSID/password works	Connects to WiFi and starts a local status/config page at http://<esp32-ip>/
AP (config portal)	No saved WiFi / connection fails	Starts AP: **C3-Config** at **192.168.4.1** and shows a setup page to enter WiFi + backend info

Saved Device Settings (Preferences / NVS)

Lists the settings the ESP32 stores in memory so it can remember WiFi and backend configuration after reboot.

Setting	Stored As	Purpose
Ingest URL	cfgIngestUrl	Django endpoint to receive readings (/api/ingest/)
Device ID	`cfgDeviceId`	Identifier sent in JSON (device_id)
Device Name	cfgDeviceName	Optional header label (X-DEVICE-NAME)
Public Code	cfgPatientCode	Optional header label (X-DEVICE-NAME)
WiFi SSID/PASS	"wifi" namespace	Required auth header (X-PUBLIC-CODE)

Django Upload (HTTPS POST)

How the ESP32 sends readings to the Django backend using an HTTP POST request with required headers.

Item	Value Used	Purpose
Method	HTTP POST	Sends readings to backend
URL	`cfgIngestUrl`	https://medsite.onrender.com/api/ingest/
Header	Content-Type: application/json	JSON payload
Header	X-PUBLIC-CODE: <cfgPatientCode>	Authorization / patient pairing (required)
Header	X-DEVICE-NAME: <cfgDeviceName>	Optional device label

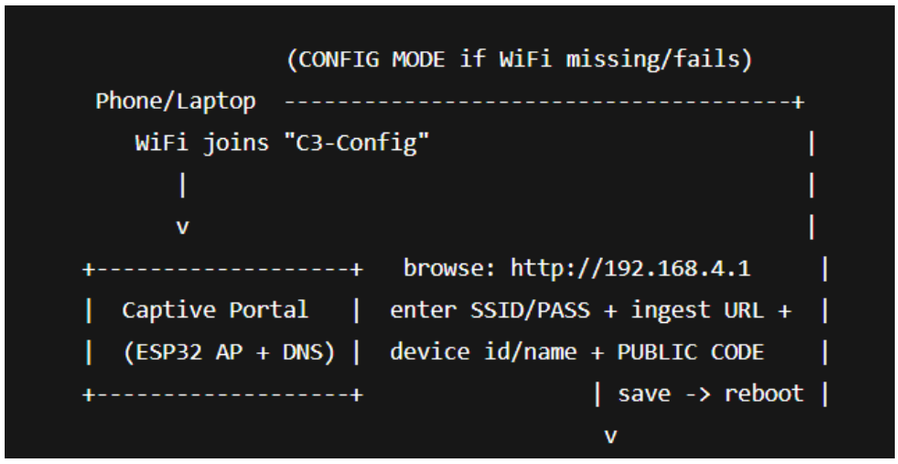
Data Sent (JSON Payload)

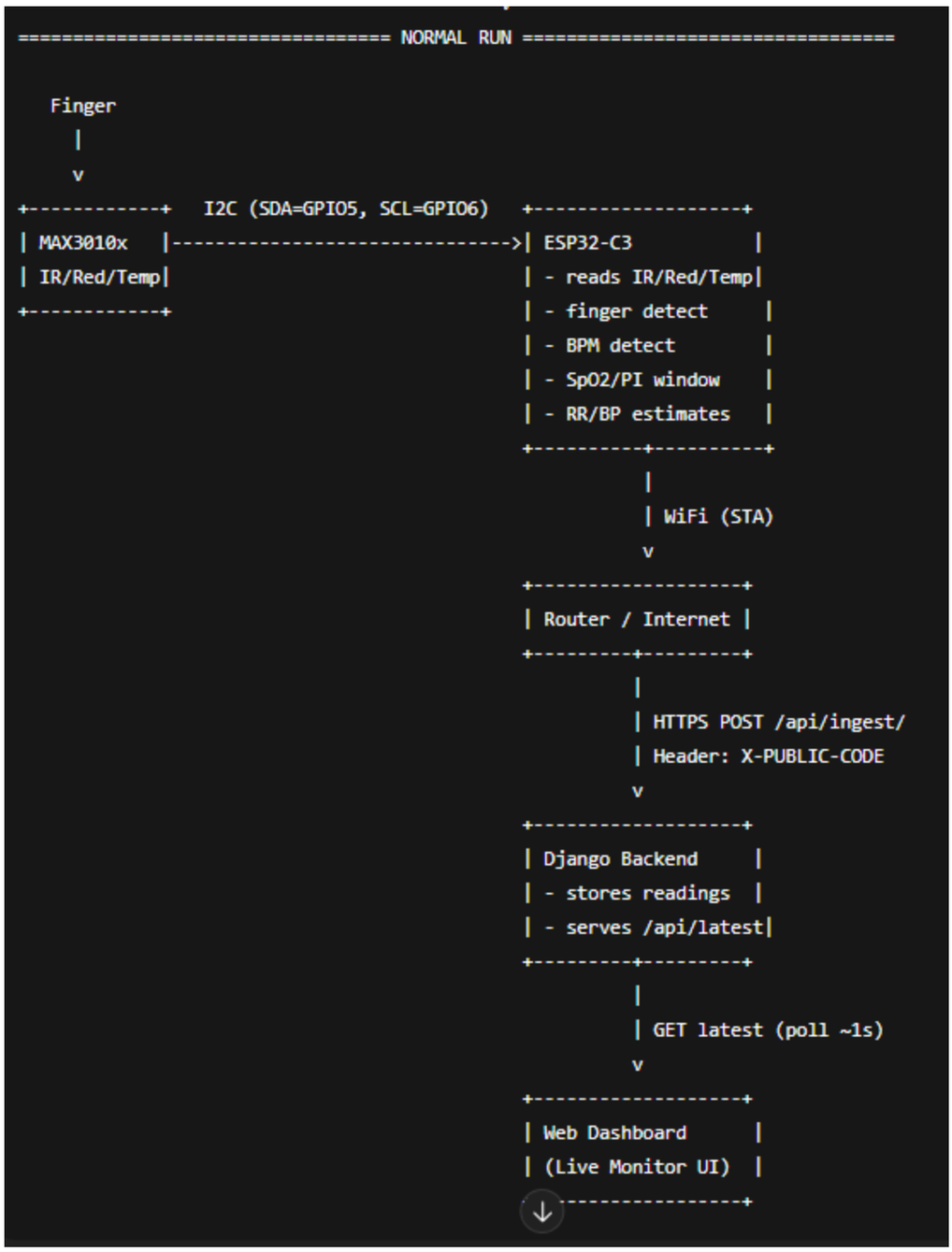
The Esp32 code sends the latest snapshot as JSON fields such as:

- **device_id, ir, red, finger**
- **bpm and hr (same value)**
- **spo2, pi, temp**
- **rr, sbp, dbp, bp**

Invalid values are sent as null (example: no valid BPM yet → bpm: null, rr: null, etc.).

ASCII Diagram




NORMAL RUN
Finger
MAX3010x
IR/Red/Temp
I2C (SDA=GPIO5, SCL=GPIO6)
ESP32-C3
- reads IR/Red/Temp
- finger detect
- BPM detect
- SpO2/PI window
- RR/BP estimates
WiFi (STA)
Router / Internet
HTTPS POST /api/ingest/
Header: X-PUBLIC-CODE
Django Backend
- stores readings
- serves /api/latest
GET latest (poll ~1s)
Web Dashboard
(Live Monitor UI)

STEP-BY-STEP WIRING GUIDE/HOW TO USE

STEP 1: Wire the MAX3010x to ESP32-C3 (Power + I²C)

- VCC/VIN → 3V3
- GND → GND
- SDA → GPIO5
- SCL → GPIO6

STEP 2: Power the ESP32-C3

- Plug ESP32-C3 via USB (laptop) or USB power bank/adapter.

STEP 3: Configure WiFi (First-time setup)

- If it can't connect, ESP32 starts **C3-Config** → connect to it
- Open **http://192.168.4.1** → enter **WiFi + ingest URL + patient code** → save → reboot

STEP 4: Start Monitoring + Upload to Django

- Place finger on sensor → ESP32 reads IR/Red/Temp, computes BPM/SpO_2
- Posts data to Django about every 1 second (invalid values sent as null)

ESP32 CODE OVERVIEW

The provided code:

Reads:

- MAX3010x IR signal
- MAX3010x Red signal
- MAX3010x temperature (estimated)

Detects:

- Finger presence using IR hysteresis
- ON if IR > 9000, OFF if IR < 7000

Calculates:

- **Heart Rate (BPM)** from beat-to-beat timing (adaptive beat detector)
- **SpO_2 estimate** using Red/IR AC/DC ratio (window-based)
- **Perfusion Index (PI)** from IR AC/DC
- **RR estimate** (derived from BPM)
- **BP estimate (SBP/DBP)** (derived from BPM)

Sends data to Django:

- **HTTPS POST** to cfgIngestUrl every ~1 second (separate task)
- Uses header: **X-PUBLIC-CODE** (required)
- Optional header: **X-DEVICE-NAME**

JSON format (example fields):

- **device_id, ir, red, finger**
- **bpm/hr, spo2, pi, temp**
- **rr, sbp, dbp, bp**

Note: If a value is not valid yet, the esp32 code sends it as null (not “N/A”).

HOW THE SYSTEM WORKS (Simplified)

1. Read IR + Red + temperature from the MAX3010x sensor
2. Detect finger presence using the IR threshold (ON/OFF hysteresis)
3. If finger is present, compute Heart Rate (BPM) and build the SpO_2/PI window
4. If finger is absent for >1500 ms, reset BPM/SpO_2/PI to invalid (null)
5. Estimate RR and BP only when BPM is valid
6. Package the latest values into a JSON payload (invalid values = null)
7. Send readings to the Django backend via HTTPS POST every ~1 second using **X-PUBLIC-CODE**

ESP32 CODE

```
#include <WiFi.h>
#include <WebServer.h>
#include <DNSServer.h>
#include "esp_wifi.h"
#include <Preferences.h>

#include <HTTPClient.h>
#include <WiFiClientSecure.h>

#include <Wire.h>
#include <MAX30105.h>
#include <math.h>

// ---------------------------------------------------------------
// OPTIONAL: uncomment this to always clear
// saved WiFi on boot while debugging.
// ---------------------------------------------------------------
// #define RESET_SAVED_WIFI_ON_BOOT
// ---------------------------------------------------------------

// ====================== DEFAULT DEVICE CONFIG (overridable via portal/NVS)
======================
String cfgIngestUrl    = "https://medsite.onrender.com/api/ingest/";
String cfgDeviceId     = "c3-001";
String cfgPatientCode  = "";               // X-PUBLIC-CODE
String cfgDeviceName   = "MedSite C3";
//
================================================================================================
===

// ---------------- Captive Portal ----------------
WebServer   server(80);
DNSServer   dns;
Preferences prefs;

const byte DNS_PORT = 53;

// AP used for config mode
const char* AP_SSID     = "C3-Config";
// const char* AP_PASSWORD = nullptr;
IPAddress apIP(192, 168, 4, 1);

String networksHTML;
bool   inConfigMode = false;

// ---------------- MAX30105 / I2C ----------------
#define SDA_PIN 5
#define SCL_PIN 6

// Finger hysteresis
#define FINGER_ON_THRESHOLD  9000
#define FINGER_OFF_THRESHOLD 7000

#define MIN_BPM 40
#define MAX_BPM 200

MAX30105 sensor;

// Raw signals
long currentIr    = 0;
long  currentRed  = 0;

// Derived values
int   currentBpm      = 0;
float currentSpO2     = 0.0f;
float currentPI       = 0.0f;
float currentRR       = 0.0f;
int   currentSBP      = 0;
int   currentDBP      = 0;
```

```
float currentTemp     = 0.0f;

bool  hasFinger      =  false;
bool  hasValidBpm    =  false;
bool  hasValidSpO2   =  false;
bool  hasValidTemp    = false;

bool sensorReady = false;

// ---- Beat averaging ----
const int BEAT_HISTORY = 5;
float beatBpmHistory[BEAT_HISTORY];
int   beatIndex = 0;
int   beatCount = 0;
static float posAbsEma = 0.0f;
static float negAbsEma = 0.0f;
static bool  useNegPolarity = false;    // true = detect downward pulses

// ---- SpO2 / PI window ----
const int WINDOW_SIZE = 50;
float irWindow[WINDOW_SIZE];
float redWindow[WINDOW_SIZE];
int   windowIndex = 0;
int   windowCount = 0;

// ======= TLS client =======
WiFiClientSecure tlsClient;

// ====================== POSTING ====================
static const uint32_t POST_INTERVAL_MS = 1000;
static const uint32_t HTTP_TIMEOUT_MS  = 1500;

// ====================== SNAPSHOT FOR POST TASK ====================
typedef struct {
  long ir;
  long red;
  bool finger;

  bool validBpm;
  int   bpm;

  bool validSpO2;
  float spo2;

  float pi;

  bool validTemp;
  float temp;

  float rr;
  int   sbp;
  int   dbp;
} VitalsSnapshot;

static VitalsSnapshot gSnap;
static portMUX_TYPE   gSnapMux = portMUX_INITIALIZER_UNLOCKED;

// ====================== SENSOR AUTO-GAIN ====================
// Start moderate (prevents 262143 clipping)
static uint8_t irAmp  = 0x30;
static uint8_t redAmp = 0x24;

static uint32_t lastGainAdjust = 0;

static void applyLedAmps() {
  sensor.setPulseAmplitudeIR(irAmp);
  sensor.setPulseAmplitudeRed(redAmp);
}

static void autoGain(long ir) {
```

```
  // only adjust once/sec
  if (millis() - lastGainAdjust < 1000) return;
  lastGainAdjust = millis();

  // If clipping/saturated -> reduce LED currents
  if (ir >= 240000) {
    if (irAmp > 0x08) irAmp -= 0x08;
    if (redAmp > 0x06) redAmp -= 0x06;
    applyLedAmps();
    Serial.print("[GAIN] too high -> IRamp=0x"); Serial.print(irAmp, HEX);
    Serial.print(" REDamp=0x"); Serial.println(redAmp, HEX);
    return;
  }

  // If too low -> increase LED currents
  if (ir <= 15000) {
    if (irAmp < 0x7F) irAmp += 0x08;
    if (redAmp < 0x5F) redAmp += 0x06;
    applyLedAmps();
    Serial.print("[GAIN] too low -> IRamp=0x"); Serial.print(irAmp, HEX);
    Serial.print(" REDamp=0x"); Serial.println(redAmp, HEX);
    return;
  }

  // In the “good zone” (roughly 20k–200k) do nothing
}

// ===================== NVS HELPERS =====================
void clearSavedWifi() {
  Serial.println("Clearing saved WiFi credentials from Preferences...");
  prefs.begin("wifi", false);
  prefs.clear();
  prefs.end();
}

bool loadSavedWifi(String &ssid, String &pass) {
  prefs.begin("wifi", true);
  ssid = prefs.getString("ssid", "");
  pass = prefs.getString("pass", "");
  prefs.end();

  ssid.trim();
  pass.trim();

  if (ssid.length() == 0) {
    Serial.println("No saved WiFi credentials in NVS.");
    return false;
  }

  Serial.println("Loaded WiFi credentials from NVS:");
  Serial.print("  SSID: '"); Serial.print(ssid); Serial.println("'");
  Serial.print("  PASS length: "); Serial.println(pass.length());
  return true;
}

void saveWifi(const String &ssid, const String &pass) {
  prefs.begin("wifi", false);
  prefs.putString("ssid", ssid);
  prefs.putString("pass", pass);
  prefs.end();
  Serial.println("WiFi credentials saved to NVS.");
}

void clearDeviceConfig() {
  Serial.println("Clearing device config from Preferences...");
  prefs.begin("dev", false);
  prefs.clear();
  prefs.end();
}
```

```
void loadDeviceConfig() {
  prefs.begin("dev", true);

  String url  = prefs.getString("url", "");
  String id   = prefs.getString("id", "");
  String name = prefs.getString("name", "");
  String pub  = prefs.getString("pub", "");
  String pair = prefs.getString("pair", "");

  prefs.end();

  url.trim(); id.trim(); name.trim(); pub.trim(); pair.trim();

  if (url.length())  cfgIngestUrl  = url;
  if (id.length())   cfgDeviceId   = id;
  if (name.length()) cfgDeviceName = name;

  if (pub.length()) cfgPatientCode = pub;
  else if (pair.length()) cfgPatientCode = pair;

  Serial.println("Loaded device config:");
  Serial.println("  url=" + cfgIngestUrl);
  Serial.println("  id=" + cfgDeviceId);
  Serial.println("  patient_code=" + cfgPatientCode);
  Serial.println("  name=" + cfgDeviceName);
}

void saveDeviceConfig() {
  prefs.begin("dev", false);
  prefs.putString("url", cfgIngestUrl);
  prefs.putString("id",   cfgDeviceId);
  prefs.putString("pub", cfgPatientCode);
  prefs.putString("name", cfgDeviceName);
  prefs.remove("pair");
  prefs.end();
  Serial.println("Device config saved to NVS.");
}

// ===================== NETWORK SCAN =====================
void buildNetworksList() {
  Serial.println("Scanning for WiFi networks...");
  networksHTML = "";

  int n = WiFi.scanNetworks();
  if (n <= 0) {
    networksHTML = "<option value=''>No networks found</option>";
    Serial.println("No networks found.");
  } else {
    for (int i = 0; i < n; i++) {
      String ssid = WiFi.SSID(i);
      if (ssid.length() == 0) continue;

      networksHTML += "<option value='";
      networksHTML += ssid;
      networksHTML += "'>";
      networksHTML += ssid;
      networksHTML += "</option>";
    }
  }
  WiFi.scanDelete();
}

// ===================== PORTAL HTML =====================
String buildPortalHtml() {
  if (networksHTML.length() == 0) buildNetworksList();

  String html =
    "<!DOCTYPE html><html lang='en'>"
    "<head><meta charset='UTF-8'/>"
    "<meta name='viewport' content='width=device-width, initial-scale=1.0'/>"
```

```
    "<title>ESP32-C3 WiFi Config</title>"
    "<style>"
    "body{font-family:system-ui;background:#f5f5f5;margin:0;}"
    ".c{max-width:560px;margin:40px auto;background:#fff;border:1px solid #ddd;border-
radius:18px;padding:20px;}"
    "label{display:block;margin-top:10px;font-size:13px;color:#333;}"
    "input,select{width:100%;padding:10px 12px;border:1px solid #ccc;border-
radius:10px;font-size:14px;}"
    "button,a{display:block;width:100%;margin-top:12px;padding:12px;border-
radius:12px;border:none;background:#111;color:#fff;"
    "text-decoration:none;text-align:center;font-weight:800;}"
    "a{background:#6b7280;}"
    "</style></head><body>"
    "<div class='c'>"
    "<h2>WiFi Setup</h2>"
    "<form method='POST' action='/save'>"
      "<label>WiFi SSID</label>"
      "<select name='ssid'>" + networksHTML + "</select>"
      "<label>Password</label>"
      "<input type='password' name='pass' placeholder='Enter WiFi password'/>"
      "<label>Django Ingest URL</label>"
      "<input type='text' name='url' value='" + cfgIngestUrl + "'/>"
      "<label>Device ID (optional)</label>"
      "<input type='text' name='device_id' value='" + cfgDeviceId + "'/>"
      "<label>Device Name</label>"
      "<input type='text' name='device_name' value='" + cfgDeviceName + "'/>"
      "<label>Patient Code (X-PUBLIC-CODE)</label>"
      "<input type='text' name='patient_code' value='" + cfgPatientCode + "'
placeholder='PUB-XXXXXXX'/>"
      "<button type='submit'>Connect & Save</button>"
    "</form>"
    "<a href='/rescan'>Rescan Networks</a>"
    "<a href='/factory' onclick=\"return confirm('Factory reset?');\">Factory Reset</a>"
    "</div></body></html>";

  return html;
}

void handlePortal() { server.send(200, "text/html", buildPortalHtml()); }

void handleRescan() {
  buildNetworksList();
  server.sendHeader("Location", "/", true);
  server.send(302, "text/plain", "Rescanning...");
}

void handleFactory() {
  clearSavedWifi();
  clearDeviceConfig();
  server.send(200, "text/plain", "Factory reset done. Rebooting...");
  delay(800);
  ESP.restart();
}

// ==================== CONNECT HELPER ====================
bool connectWiFiWith(const String &ssid, const String &pass, bool keepAP) {
  WiFi.disconnect(true, true);
  delay(300);

  WiFi.mode(keepAP ? WIFI_AP_STA : WIFI_STA);
  WiFi.setSleep(false);
  WiFi.setAutoReconnect(true);
  WiFi.persistent(false);
  esp_wifi_set_ps(WIFI_PS_NONE);
  esp_wifi_set_max_tx_power(78);

  const int MAX_ATTEMPTS = 3;
  for (int attempt = 1; attempt <= MAX_ATTEMPTS; attempt++) {
    Serial.print("Connecting to '"); Serial.print(ssid);
    Serial.print("' (attempt "); Serial.print(attempt); Serial.println(")...");
```

```
    WiFi.begin(ssid.c_str(), pass.c_str());

    unsigned long start = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - start < 20000) {
      delay(250);
      Serial.print(".");
    }
    Serial.println();

    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("☑ WiFi connected");
      Serial.print("IP: "); Serial.println(WiFi.localIP());
      return true;
    }
    Serial.println("❌ WiFi failed");
    delay(1500);
  }

  return false;
}

bool connectSavedWifi() {
  String ssid, pass;
  if (!loadSavedWifi(ssid, pass)) return false;
  return connectWiFiWith(ssid, pass, false);
}

// ==================== SENSOR INIT ====================
bool initSensor() {
  Wire.begin(SDA_PIN, SCL_PIN);
  Wire.setClock(400000);
  delay(200);

  const int MAX_TRIES = 5;
  for (int i = 0; i < MAX_TRIES; i++) {
    Serial.print("Trying MAX3010x init, attempt ");
    Serial.println(i + 1);

    if (sensor.begin(Wire, I2C_SPEED_FAST)) {
      Serial.println("☑ MAX3010x found!");

      // ☑ IMPORTANT: use a higher ADC range to avoid 262143 clipping
      byte ledBrightness = 60;
      byte sampleAverage = 4;
      byte ledMode = 2;        // Red + IR
      int  sampleRate = 100;
      int  pulseWidth = 411;
      int  adcRange = 16384;

      sensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth,
adcRange);

      // Start moderate (autoGain will tune)
      applyLedAmps();
      sensor.setPulseAmplitudeGreen(0x00);

      return true;
    }

    Serial.println("❌ MAX3010x not found, retrying...");
    delay(300);
  }
  return false;
}

// ==================== SPO2 / PI ====================
void computeSpO2andPIFromWindow() {
  if (windowCount < WINDOW_SIZE) return;
```

```
  float irMean = 0.0f, redMean = 0.0f;
  for (int i = 0; i < WINDOW_SIZE; i++) {
    irMean  += irWindow[i];
    redMean += redWindow[i];
  }
  irMean  /= WINDOW_SIZE;
  redMean /= WINDOW_SIZE;

  if (irMean < 2000 || redMean < 2000) {
    hasValidSpO2 = false;
    currentPI = 0.0f;
    return;
  }

  float irAcMean = 0.0f, redAcMean = 0.0f;
  for (int i = 0; i < WINDOW_SIZE; i++) {
    irAcMean  += fabs(irWindow[i]  - irMean);
    redAcMean += fabs(redWindow[i] - redMean);
  }
  irAcMean  /= WINDOW_SIZE;
  redAcMean /= WINDOW_SIZE;

  float pi = (irAcMean / irMean) * 100.0f;
  if (pi < 0.0f)  pi = 0.0f;
  if (pi > 20.0f) pi = 20.0f;
  currentPI = pi;

  float irAcDc  = irAcMean  / irMean;
  float redAcDc = redAcMean / redMean;
  if (irAcDc <= 0.0f) {
    hasValidSpO2 = false;
    return;
  }

  float R    = redAcDc / irAcDc;
  float spo2 = 110.0f - 25.0f * R;

  if (spo2 > 100.0f) spo2 = 100.0f;
  if (spo2 < 70.0f)  spo2 = 70.0f;

  currentSpO2  = spo2;
  hasValidSpO2 = true;
}

// ====================== BEAT DETECTOR (fixed threshold logic) ====================
static bool     emaInit = false;
static float    dcEma = 0.0f;
static float    acAbsEma = 0.0f;
static bool     aboveThr = false;
static float    peakAc = 0.0f;
static uint32_t peakTime = 0;
static uint32_t lastBeatTime = 0;

static inline float fmaxf2(float a, float b) { return (a > b) ? a : b; }

void resetBeatDetector() {
  emaInit = false;
  dcEma = 0.0f;
  acAbsEma = 0.0f;
  aboveThr = false;
  peakAc = 0.0f;
  peakTime = 0;
  lastBeatTime = 0;

  posAbsEma = 0.0f;
  negAbsEma = 0.0f;
  useNegPolarity = false;
}

void onBeat(uint32_t beatTime) {
```

```
  if (lastBeatTime == 0) {
    lastBeatTime = beatTime;
    return; // need 2 beats to get dt
  }

  uint32_t dt = beatTime - lastBeatTime;
  if (dt < 250 || dt > 2000) return; // more tolerant window

  float instBpm = 60000.0f / (float)dt;
  if (instBpm < MIN_BPM || instBpm > MAX_BPM) return;

  lastBeatTime = beatTime;

  beatBpmHistory[beatIndex] = instBpm;
  beatIndex = (beatIndex + 1) % BEAT_HISTORY;
  if (beatCount < BEAT_HISTORY) beatCount++;

  float sum = 0.0f;
  for (int i = 0; i < beatCount; i++) sum += beatBpmHistory[i];
  currentBpm = (int)(sum / beatCount + 0.5f);

  // ☑ show BPM as soon as we have 1 interval (2 beats)
  hasValidBpm = (beatCount >= 1);
}

void processIrForBeats(long ir) {
  static uint32_t aboveStartMs = 0;

  if (!emaInit) {
    dcEma = (float)ir;
    acAbsEma = 0.0f;
    posAbsEma = 0.0f;
    negAbsEma = 0.0f;
    useNegPolarity = false;
    aboveThr = false;
    aboveStartMs = 0;
    emaInit = true;
    return;
  }

  // DC removal
  dcEma += 0.02f * ((float)ir - dcEma);
  float ac = (float)ir - dcEma;

  // polarity learning
  float absac = fabs(ac);
  if (ac >= 0) posAbsEma += 0.05f * (absac - posAbsEma);
  else         negAbsEma += 0.05f * (absac - negAbsEma);

  if (!useNegPolarity && negAbsEma > posAbsEma * 1.6f) useNegPolarity = true;
  if ( useNegPolarity && posAbsEma > negAbsEma * 1.6f) useNegPolarity = false;

  // chosen polarity signal
  float x = useNegPolarity ? (-ac) : (ac);

  // adaptive threshold (a bit easier to trigger)
  acAbsEma += 0.05f * (fabs(x) - acAbsEma);
  float thr = fmaxf2(acAbsEma * 0.40f, 12.0f);

  if (!aboveThr) {
    if (x > thr) {
      aboveThr = true;
      aboveStartMs = millis();
      peakAc = x;
      peakTime = millis();
    }
  } else {
    if (x > peakAc) {
      peakAc = x;
      peakTime = millis();
```

```
    }

    // ☑ easier release so it re-arms every beat
    bool released = (x < (thr * 0.65f));

    // ☑ safety timeout: if we stayed "above" too long, force release
    bool timedOut = (millis() - aboveStartMs > 420);

    if (released || timedOut) {
      aboveThr = false;

      static uint32_t lastAccepted = 0;
      if (millis() - lastAccepted >= 280) {
        lastAccepted = millis();
        onBeat(peakTime);
      }
    }
  }
}

// ==================== HTTPS POST ====================
int postToDjangoWithSnapshot(const VitalsSnapshot &s) {
  if (WiFi.status() != WL_CONNECTED) return -1;
  if (!cfgPatientCode.length()) return -2;

  String bpmStr = s.validBpm ? String(s.bpm) : "null";
  String sbpStr = s.validBpm ? String(s.sbp) : "null";
  String dbpStr = s.validBpm ? String(s.dbp) : "null";
  String spo2Str = s.validSpO2 ? String(s.spo2, 1) : "null";
  String piStr = s.finger ? String(s.pi, 1) : "null";
  String rrStr = s.validBpm ? String(s.rr, 1) : "null";
  String tmpStr = s.validTemp ? String(s.temp, 1) : "null";
  String bpStr = s.validBpm ? ("\"" + String(s.sbp) + "/" + String(s.dbp) + "\"") :
"null";

  String payload = "{";
  payload += "\"device_id\":\"" + cfgDeviceId + "\",";
  payload += "\"ir\":" + String(s.ir) + ",";
  payload += "\"red\":" + String(s.red) + ",";
  payload += "\"finger\":" + String(s.finger ? "true" : "false") + ",";
  payload += "\"bpm\":" + bpmStr + ",";
  payload += "\"hr\":" + bpmStr + ",";
  payload += "\"sbp\":" + sbpStr + ",";
  payload += "\"dbp\":" + dbpStr + ",";
  payload += "\"bp\":" + bpStr + ",";
  payload += "\"spo2\":" + spo2Str + ",";
  payload += "\"pi\":" + piStr + ",";
  payload += "\"rr\":" + rrStr + ",";
  payload += "\"temp\":" + tmpStr;
  payload += "}";

  HTTPClient http;
  http.setTimeout(HTTP_TIMEOUT_MS);
  http.setReuse(true);

  int code = -10;
  String resp;

  if (cfgIngestUrl.startsWith("https://")) {
    tlsClient.setInsecure();
    tlsClient.setTimeout(HTTP_TIMEOUT_MS);
    if (!http.begin(tlsClient, cfgIngestUrl.c_str())) return -10;
  } else {
    if (!http.begin(cfgIngestUrl.c_str())) return -10;
  }

  http.addHeader("Content-Type", "application/json");
  http.addHeader("X-PUBLIC-CODE", cfgPatientCode);
  if (cfgDeviceName.length()) http.addHeader("X-DEVICE-NAME", cfgDeviceName);
```

```
  String esp32Url = "http://" + WiFi.localIP().toString() + "/";
  http.addHeader("X-ESP32-URL", esp32Url);

  code = http.POST(payload);
  if (code > 0) resp = http.getString();
  else resp = http.errorToString(code);

  http.end();

  Serial.print("[Django POST] code=");
  Serial.print(code);
  Serial.print(" resp=");
  Serial.println(resp);

  return code;
}

void postTask(void *param) {
  (void)param;
  uint32_t lastPost = 0;

  for (;;) {
    if (inConfigMode || !sensorReady) {
      vTaskDelay(pdMS_TO_TICKS(250));
      continue;
    }

    if (WiFi.status() == WL_CONNECTED && (millis() - lastPost >= POST_INTERVAL_MS)) {
      VitalsSnapshot local;
      portENTER_CRITICAL(&gSnapMux);
      local = gSnap;
      portEXIT_CRITICAL(&gSnapMux);

      postToDjangoWithSnapshot(local);
      lastPost = millis();
    }

    vTaskDelay(pdMS_TO_TICKS(50));
  }
}

// ===================== RESET / REBOOT =====================
void handleReset() {
  clearSavedWifi();
  server.send(200, "text/plain", "WiFi cleared. Rebooting...");
  delay(600);
  ESP.restart();
}

void handleReboot() {
  server.send(200, "text/plain", "Rebooting...");
  delay(400);
  ESP.restart();
}

// ===================== STA PAGE =====================
void handleStaRoot() {
  String ip = WiFi.localIP().toString();

  String html =
    "<!DOCTYPE html><html><head><meta charset='utf-8'/>"
    "<meta name='viewport' content='width=device-width, initial-scale=1'/>"
    "<title>ESP32 Status</title>"
    "<style>"
      "body{font-family:system-ui;background:#f5f5f5;margin:0;}"
      ".card{max-width:720px;margin:40px auto;background:#fff;border:1px solid
#ddd;border-radius:18px;padding:18px;}"
      "code{background:#f1f1f1;padding:2px 6px;border-radius:8px;}"
      "label{display:block;margin-top:10px;font-size:13px;color:#333;}"
      "input{width:100%;padding:10px 12px;border:1px solid #ccc;border-radius:10px;font-
```

```
size:14px;}"
      ".row{display:flex;gap:10px;margin-top:12px;}"
      ".btn{flex:1;display:block;text-align:center;padding:12px;border-
```

```
radius:12px;background:#111;color:#fff;"
      "text-decoration:none;font-weight:800;border:none;cursor:pointer;}"
      ".btn.red{background:#dc2626;}"
      ".btn.gray{background:#6b7280;}"
      "#toast{position:fixed;left:50%;bottom:22px;transform:translateX(-50%);"
      "background:#111;color:#fff;padding:12px 14px;border-radius:12px;"
      "box-shadow:0 10px 30px rgba(0,0,0,.18);display:none;min-width:240px;"
      "text-align:center;font-weight:800;z-index:9999;}"
      "#toast.ok{background:#16a34a;}"
      "#toast.bad{background:#dc2626;}"
    "</style></head><body>"
    "<div id='toast'></div>"
    "<div class='card'>"
      "<h2>ESP32 Connected ☑</h2>"
      "<p><b>IP:</b> " + ip + "</p>"
      "<p><b>Posting to:</b> <code>" + cfgIngestUrl + "</code></p>"
      "<p><b>Patient Code:</b> <code>" + (cfgPatientCode.length() ? cfgPatientCode :
"(none)") + "</code></p>"
      "<hr/>"
      "<h3>Edit Device Settings</h3>"

      "<form id='deviceForm'>"
        "<label>Django Ingest URL</label>"
        "<input name='url' value='" + cfgIngestUrl + "'/>"
        "<label>Device ID</label>"
        "<input name='device_id' value='" + cfgDeviceId + "'/>"
        "<label>Device Name</label>"
        "<input name='device_name' value='" + cfgDeviceName + "'/>"
        "<label>Patient Code (X-PUBLIC-CODE)</label>"
        "<input name='patient_code' value='" + cfgPatientCode + "'/>"

        "<div class='row'>"
          "<button class='btn' type='submit'>Save</button>"
          "<button class='btn red' type='button' onclick='doReset()'>Reset WiFi</button>"
        "</div>"
        "<div class='row'>"
          "<button class='btn gray' type='button' onclick='doReboot()'>Reboot</button>"
          "<button class='btn gray' type='button' onclick='doFactory()'>Factory
Reset</button>"
        "</div>"
      "</form>"
    "</div>"

    "<script>"
      "const toastEl=document.getElementById('toast');"
      "function toast(msg, ok=true){"
        "toastEl.className='';"
        "toastEl.classList.add(ok?'ok':'bad');"
        "toastEl.textContent=msg;"
        "toastEl.style.display='block';"
        "clearTimeout(window.__tHide);"
        "window.__tHide=setTimeout(()=>toastEl.style.display='none',1800);"
      "}"

      "async function pingBack(){"
        "let tries=0;"
        "const timer=setInterval(async()=>{"
          "tries++;"
          "try{"
            "const r=await fetch('/ping',{cache:'no-store'});"
            "if(r.ok){ clearInterval(timer); location.href='/'; }"
          "}catch(e){}"
          "if(tries>40){ clearInterval(timer); toast('Still rebooting… refresh', false);
}"
        "},500);"
      "}"

      "document.getElementById('deviceForm').addEventListener('submit', async (e)=>{"
        "e.preventDefault();"
        "toast('Saving…');"
```

```
        "try{"
          "const fd=new FormData(e.target);"
          "const r=await fetch('/save_device',{method:'POST',body:fd,cache:'no-store'});"
          "toast(r.ok ? '☑ Saved!' : '✕ Save failed', r.ok);"
        "}catch(err){ toast('✕ Save failed', false); }"
      "});"

      "async function doReboot(){"
        "toast('Rebooting…');"
        "try{ fetch('/reboot',{cache:'no-store'}); }catch(e){}"
        "setTimeout(pingBack,700);"
      "}"

      "async function doReset(){"
        "toast('Resetting WiFi…');"
        "try{ fetch('/reset',{cache:'no-store'}); }catch(e){}"
      "}"

      "async function doFactory(){"
        "if(!confirm('Factory reset?')) return;"
        "toast('Factory reset…');"
        "try{ fetch('/factory',{cache:'no-store'}); }catch(e){}"
      "}"
    "</script>"

    "</body></html>";

  server.send(200, "text/html", html);
}

void handleSaveDevice() {
  String url  = server.arg("url"); url.trim();
  String did  = server.arg("device_id"); did.trim();
  String name = server.arg("device_name"); name.trim();
  String pub  = server.arg("patient_code"); pub.trim();

  if (url.length())  cfgIngestUrl  = url;
  if (did.length())  cfgDeviceId   = did;
  if (name.length()) cfgDeviceName = name;
  cfgPatientCode = pub;

  saveDeviceConfig();
  server.send(200, "application/json", "{\"ok\":true}");
}

void handleNotFound() {
  if (inConfigMode) server.send(200, "text/html", buildPortalHtml());
  else server.send(404, "text/plain", "Not found");
}

// ===================== PORTAL SAVE HANDLER =====================
void handleSaveConfig() {
  String ssid = server.arg("ssid"); ssid.trim();
  String pass = server.arg("pass"); pass.trim();

  String url  = server.arg("url"); url.trim();
  String did  = server.arg("device_id"); did.trim();
  String name = server.arg("device_name"); name.trim();
  String pub  = server.arg("patient_code"); pub.trim();

  bool ok = connectWiFiWith(ssid, pass, true);

  if (ok) {
    saveWifi(ssid, pass);

    if (url.length())  cfgIngestUrl  = url;
    if (did.length())  cfgDeviceId   = did;
    if (name.length()) cfgDeviceName = name;
    cfgPatientCode = pub;
```

```
    saveDeviceConfig();

    server.send(200, "text/plain", "Connected & saved. Rebooting...");
    delay(800);
    ESP.restart();
  } else {
    server.send(200, "text/plain", "Failed to connect. Try again.");
  }
}

// ===================== START CONFIG AP ====================
void startConfigAP() {
  Serial.println("Starting CONFIG AP (captive)…");
  inConfigMode = true;

  WiFi.mode(WIFI_AP_STA);
  esp_wifi_set_ps(WIFI_PS_NONE);
  esp_wifi_set_max_tx_power(78);

  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(AP_SSID, nullptr, 1, 0, 4);

  dns.start(DNS_PORT, "*", apIP);

  buildNetworksList();

  server.on("/", handlePortal);
  server.on("/save", HTTP_POST, handleSaveConfig);
  server.on("/rescan", handleRescan);
  server.on("/factory", handleFactory);

  server.on("/generate_204", handlePortal);
  server.on("/gen_204", handlePortal);
  server.on("/hotspot-detect.html", handlePortal);
  server.on("/ncsi.txt", handlePortal);
  server.on("/connecttest.txt", handlePortal);

  server.onNotFound(handleNotFound);
  server.begin();
}

// ===================== START STA SERVER ====================
void startStaServer() {
  inConfigMode = false;
  dns.stop();

  server.on("/", handleStaRoot);
  server.on("/reset", handleReset);
  server.on("/reboot", handleReboot);
  server.on("/factory", handleFactory);
  server.on("/save_device", HTTP_POST, handleSaveDevice);
  server.on("/ping", [](){ server.send(200, "text/plain", "ok"); });
  server.onNotFound(handleNotFound);

  server.begin();

  Serial.println("☑ STA server started.");
  Serial.print("Open: http://");
  Serial.print(WiFi.localIP());
  Serial.println("/");
}

// ===================== SETUP / LOOP ====================
void resetVitalsState() {
  windowIndex  = 0;
  windowCount  = 0;
  beatIndex    = 0;
  beatCount    = 0;
  hasValidBpm  = false;
  hasValidSpO2 = false;
```

```
  currentPI    = 0.0f;
  currentRR    = 0.0f;
  currentSBP   = 0;
  currentDBP   = 0;
  resetBeatDetector();
}

void setup() {
  Serial.begin(115200);
  delay(600);
  Serial.println();
  Serial.println("=== ESP32-C3 + BPM FIX (no clipping) ===");

  loadDeviceConfig();

#ifdef RESET_SAVED_WIFI_ON_BOOT
  clearSavedWifi();
#endif

  if (!connectSavedWifi()) {
    startConfigAP();
    return;
  }

  startStaServer();

  Serial.println("[BOOT] Warm-up 2s...");
  delay(2000);

  sensorReady = initSensor();
  if (!sensorReady) Serial.println("⚠ Sensor not ready.");

  Serial.print("Posting to Django: ");
  Serial.println(cfgIngestUrl);

  xTaskCreatePinnedToCore(
    postTask,
    "postTask",
    8192,
    NULL,
    1,
    NULL,
    0
  );
}

void loop() {
  if (inConfigMode) {
    dns.processNextRequest();
    server.handleClient();
    delay(10);
    return;
  }

  server.handleClient();
  if (!sensorReady) { delay(10); return; }

  sensor.check();

  static uint32_t noFingerSince = 0;
  static uint32_t lastTempRead = 0;
  static bool fingerState = false;

  // ☑ signal debug window
  static long irMin = 2147483647;
  static long irMax = 0;
  static uint32_t sigStart = 0;

  while (sensor.available()) {
    long ir  = sensor.getIR();
```

```
long red = sensor.getRed();
sensor.nextSample();

currentIr = ir;
currentRed = red;

// finger hysteresis
if (!fingerState && ir > FINGER_ON_THRESHOLD) fingerState = true;
if (fingerState && ir < FINGER_OFF_THRESHOLD) fingerState = false;
hasFinger = fingerState;

// temp (once/sec)
if (millis() - lastTempRead > 1000) {
  float t = sensor.readTemperature();
  if (hasFinger && t > 20.0f && t < 45.0f) { currentTemp = t; hasValidTemp = true; }
  else { hasValidTemp = false; }
  lastTempRead = millis();
}

// require 1500ms no-finger before wipe
if (!hasFinger) {
  if (noFingerSince == 0) noFingerSince = millis();
  if (millis() - noFingerSince > 1500) {
    resetVitalsState();
  }
  continue;
} else {
  noFingerSince = 0;
}

// ☑ auto-gain prevents 262143 saturation
autoGain(ir);

// ☑   signal debug min/max
if (ir < irMin) irMin = ir;
if (ir > irMax) irMax = ir;
if (sigStart == 0) sigStart = millis();
if (millis() - sigStart >= 1000) {
  long delta = (irMax > irMin) ? (irMax - irMin) : 0;
  Serial.print("[SIG] IRmin="); Serial.print(irMin);
  Serial.print(" IRmax="); Serial.print(irMax);
  Serial.print(" delta="); Serial.print(delta);
  Serial.print(" | BPM=");
  if (hasValidBpm) Serial.print(currentBpm); else Serial.print("null");
  Serial.print(" beats="); Serial.print(beatCount);
  Serial.print(" IRamp=0x"); Serial.print(irAmp, HEX);
  Serial.print(" REDamp=0x"); Serial.println(redAmp, HEX);

  irMin = 2147483647;
  irMax = 0;
  sigStart = millis();
}

// SpO2/PI window update (optional)
irWindow[windowIndex]  = (float)ir;
redWindow[windowIndex] = (float)red;
windowIndex = (windowIndex + 1) % WINDOW_SIZE;
if (windowCount < WINDOW_SIZE) windowCount++;
if (windowCount == WINDOW_SIZE) computeSpO2andPIFromWindow();

// ☑ Beat detection
processIrForBeats(ir);

// RR + BP (depends on BPM)
if (hasValidBpm) {
  float rr = 16.0f + (currentBpm - 70) * 0.05f;
  if (rr < 8.0f)  rr = 8.0f;
  if (rr > 30.0f) rr = 30.0f;
  currentRR = rr;
```

```
      float sys = 110.0f + (currentBpm - 70) * 0.4f;
      float dia = 70.0f  + (currentBpm - 70) * 0.3f;

      if (sys < 90.0f)  sys = 90.0f;
      if (sys > 200.0f) sys = 200.0f;
      if (dia < 50.0f)  dia = 50.0f;
      if (dia > 130.0f) dia = 130.0f;

      currentSBP = (int)(sys + 0.5f);
      currentDBP = (int)(dia + 0.5f);
    }
  }

  // ☑ update snapshot for POST task
  portENTER_CRITICAL(&gSnapMux);
  gSnap.ir        = currentIr;
  gSnap.red       = currentRed;
  gSnap.finger    = hasFinger;
  gSnap.validBpm  = hasValidBpm;
  gSnap.bpm       = currentBpm;
  gSnap.validSpO2 = hasValidSpO2;
  gSnap.spo2      = currentSpO2;
  gSnap.pi        = currentPI;
  gSnap.validTemp = hasValidTemp;
  gSnap.temp      = currentTemp;
  gSnap.rr        = currentRR;
  gSnap.sbp       = currentSBP;
  gSnap.dbp       = currentDBP;
  portEXIT_CRITICAL(&gSnapMux);

  delay(1);
}
```

DJANGO BACKEND OVERVIEW (ESP32 → Django Integration)

Core Functions

🟩 Receives Data (Ingest API)

- The ESP32 sends readings to Django through an HTTP/HTTPS POST request to: /api/ingest/
- Each request includes a required header for pairing/auth: X-PUBLIC-CODE (patient public code)

🟩 Validates and Saves Readings

- Django checks the incoming payload (device ID, finger flag, vitals, timestamps).
- Readings are stored in the database (ex: Reading model) linked to the correct patient/device.

🟩 Provides Latest Reading (Live API)

- The dashboard requests the most recent reading using: GET /api/latest/<public_code>/ *(or your equivalent route)*
- Django returns a JSON object containing the newest values (BPM, SpO_2, IR, Red, Temp, BP, etc.).

🟩 Supports Live Monitoring UI

- The Live Monitor page uses JavaScript polling (ex: every 1 second) to refresh displayed values and update graphs/history.
- If Django reports no recent/available reading, the UI shows “Disconnected” or “N/A” values.

DJANGO STRUCTURE OVERVIEW (URLs, Views, Models)

1) URLs (Routing Layer)

The file medsite/urls.py defines the web routes for both the API and the monitoring pages.
API routes (JSON endpoints)

- POST /api/ingest/ → receives ESP32 readings
- GET /api/latest/<public_code>/ → returns latest reading by public code
- GET /api/latest/p/<patient_id>/ → returns latest reading by patient ID

Web pages (UI)

- /stats/<patient_id>/ → live monitor page (doctor/internal view)
- /m/<public_code>/ → public/share monitor page

Patient management

- /patients/create/ → create patient
- /patients/<patient_id>/ → patient details
- /patients/<patient_id>/archive/ → archive patient
- /patients/<patient_id>/restore/ → restore patient

URLS CODE:

```
# medsite/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name="home"),
    path("api/ingest/", views.api_ingest),

    path("api/latest/p/<int:patient_id>/", views.api_latest_patient, name="api_latest_patient"),
    path("api/latest/<str:public_code>/", views.api_latest, name="api_latest"),
    path("stats/<int:patient_id>/", views.stats_page, name="stats"),
    path("patients/create/", views.create_patient, name="create_patient"),
    path("patients/<int:patient_id>/", views.patient_detail, name="patient_detail"),
    path("patients/<int:patient_id>/archive/", views.archive_patient, name="archive_patient"),
    path("patients/<int:patient_id>/restore/", views.unarchive_patient, name="unarchive_patient"),
    path("m/<str:public_code>/", views.public_monitor, name="public_monitor"),


    path("login/", views.login_view, name="login"),
    path("register/", views.register_view, name="register"),
    path("logout/", views.logout_view, name="logout"),
]
```

2) Views (Backend Logic)

The file medsite/views.py contains the functions that run when a URL is requested.
Main responsibilities

- **api_ingest**
  Accepts ESP32 POST data, identifies the patient using X-PUBLIC-CODE, and saves a new Reading.
- **api_latest / api_latest_patient**
  Fetches the newest Reading from the database and returns it as JSON for the dashboard.
- **stats_page / public_monitor**
  Renders the HTML pages that show live vitals and charts (the frontend polls the API).
- **Patient actions**
  Create patients, show patient info, and toggle archive/restore.

VIEWS CODE:

```
# medsite/views.py
import json
from datetime import timedelta

from django.contrib.auth import login, logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.forms import AuthenticationForm
from django.http import JsonResponse
from django.shortcuts import get_object_or_404, redirect, render
from django.utils import timezone
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_GET, require_POST
from django.conf import settings
from .forms import PatientForm, RegisterForm
from .models import Patient, Reading
from django.urls import reverse


# ------------------ helpers ------------------
def to_bool(v):
    if isinstance(v, bool):
        return v
    if v is None:
        return False
    if isinstance(v, (int, float)):
        return bool(v)
    if isinstance(v, str):
        return v.strip().lower() in ("1", "true", "t", "yes", "y", "on")
    return False

# ------------------ pages ------------------
def home(request):
      esp32_url = ""

      if request.user.is_authenticated:
            patients = Patient.objects.filter(
                  doctor=request.user, is_archived=False
            ).order_by("name")

            archived_patients = Patient.objects.filter(
                  doctor=request.user, is_archived=True
            ).order_by("-archived_at", "name")

            esp32_url = (Patient.objects
                  .filter(doctor=request.user)
                  .exclude(last_esp32_url="")
                  .order_by("-last_esp32_seen")
                  .values_list("last_esp32_url", flat=True)
                  .first() or ""
            )

            return render(request, "medsite/home.html", {
                  "patients": patients,
                  "archived_patients": archived_patients,
                  "esp32_url": esp32_url,
            })

return render(request, "medsite/home.html", {"esp32_url": esp32_url})

@login_required
@require_POST
def archive_patient(request, patient_id):
    patient = get_object_or_404(Patient, id=patient_id, doctor=request.user, is_archived=False)
```

```
    patient.archive()
    return redirect("home")

@login_required
@require_POST
def unarchive_patient(request, patient_id):
```

```
    patient = get_object_or_404(Patient, id=patient_id, doctor=request.user, is_archived=True)
    patient.unarchive()
    return redirect("home")

@login_required
def create_patient(request):
    if request.method == "POST":
        form = PatientForm(request.POST)
        if form.is_valid():
            p = form.save(commit=False)
            p.doctor = request.user
            p.save()
            return redirect("home")
    else:
        form = PatientForm()

    return render(request, "medsite/create_patient.html", {"form": form})

@login_required
def patient_detail(request, patient_id):
    patient = get_object_or_404(Patient, id=patient_id, doctor=request.user)
    return render(request, "medsite/patient_detail.html", {"patient": patient})

@require_GET
def public_monitor(request, public_code):
    patient = get_object_or_404(Patient, public_code=public_code)

    # optional: block archived
    if getattr(patient, "is_archived", False):
        return render(request, "medsite/monitor_unavailable.html", {"patient": patient}, status=404)

    endpoint = reverse("api_latest", args=[public_code])
    share_url = request.build_absolute_uri(reverse("public_monitor", args=[public_code]))

    # ☑ show address only to the assigned doctor (logged in + owns patient)
    can_view_private = request.user.is_authenticated and patient.doctor_id == request.user.id

    return render(request, "medsite/stats.html", {
        "patient": patient,
        "endpoint": endpoint,
        "share_url": share_url,
        "can_view_private": can_view_private,
    })

@login_required
def stats_page(request, patient_id):
    patient = get_object_or_404(
        Patient, id=patient_id, doctor=request.user, is_archived=False
    )
    return render(request, "medsite/stats.html", {"patient": patient})


# ------------------ APIs ------------------
@login_required
def api_latest_patient(request, patient_id):
    patient = get_object_or_404(
        Patient, id=patient_id, doctor=request.user, is_archived=False
    )

    r = patient.readings.order_by("-created_at").first()
    if not r or (timezone.now() - r.created_at > timedelta(seconds=5)):
        return JsonResponse({"detail": "Machine unavailable"}, status=200)

    return JsonResponse({
        "created_at": r.created_at.isoformat(),
        "ir": r.ir, "red": r.red, "finger": r.finger,
```

```
        "bpm": r.bpm, "spo2": r.spo2, "pi": r.pi, "rr": r.rr,
        "sbp": r.sbp, "dbp": r.dbp, "temp": r.temp,
    })

@require_GET
def api_latest(request, public_code):
    patient = get_object_or_404(Patient, public_code=public_code)

    r = patient.readings.order_by("-created_at").first()
    if not r or (timezone.now() - r.created_at > timedelta(seconds=5)):
        return JsonResponse({"detail": "Machine unavailable"}, status=200)

    return JsonResponse({
        "created_at":  r.created_at.isoformat(),
        "ir": r.ir, "red": r.red, "finger": r.finger,
        "bpm": r.bpm, "spo2": r.spo2,
        "sbp": r.sbp, "dbp": r.dbp,
        "temp": r.temp,
    })

@csrf_exempt
@require_POST
def api_ingest(request):
    code = request.headers.get("X-PUBLIC-CODE", "").strip()
    if not code:
        return JsonResponse({"detail": "Missing X-PUBLIC-CODE"}, status=400)

    try:
        patient = Patient.objects.get(public_code=code)
        if patient.is_archived:
            return JsonResponse({"detail": "Patient is archived"}, status=403)
    except Patient.DoesNotExist:
        return JsonResponse({"detail": "Invalid patient code"}, status=403)

    try:
        payload = json.loads(request.body.decode("utf-8"))
        if not isinstance(payload, dict):
            raise ValueError("JSON must be an object")
    except Exception:
        return JsonResponse({"detail": "Invalid JSON"}, status=400)

    r = Reading.objects.create(
        patient=patient,
        ir=payload.get("ir"),
        red=payload.get("red"),
        finger=to_bool(payload.get("finger", False)),
        bpm=payload.get("bpm"),
        spo2=payload.get("spo2"),
        pi=payload.get("pi"),
        rr=payload.get("rr"),
        sbp=payload.get("sbp"),
        dbp=payload.get("dbp"),
        temp=payload.get("temp"),
    )

    return JsonResponse({"ok": True, "id": r.id})

# (Optional) keep your old public-code latest endpoint for debugging only.
# If you don't use it anymore, you can delete it.


# ------------------ auth ------------------
def register_view(request):
    if request.user.is_authenticated:
        return redirect("home")
```

```
    if request.method == "POST":
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect("home")
    else:
        form = RegisterForm()

    return render(request, "medsite/register.html", {"form": form})

def login_view(request):
    if request.user.is_authenticated:
        return redirect("home")

    form = AuthenticationForm(request, data=request.POST or None)
    if request.method == "POST" and form.is_valid():
        login(request, form.get_user())
        return redirect("home")

    return render(request, "medsite/login.html", {"form": form})

@require_POST
def logout_view(request):
    logout(request)
    return redirect("home")
```

3) Models (Database Tables)

The file medsite/models.py defines the database structure.

Patient model

- Stores patient info: name, age, address, emergency_number
- Auto-generates a unique public_code (example: PUB-XXXXXXXX)
- Supports archiving using:
    - is_archived and archived_at

Reading model

- Stores each uploaded vitals snapshot:
    - created_at
    - raw signals: ir, red, finger
    - computed values: bpm, spo2, pi, rr, sbp, dbp, temp
- Linked to a patient using:
    - patient = ForeignKey(Patient, ...)

MODELS CODE:

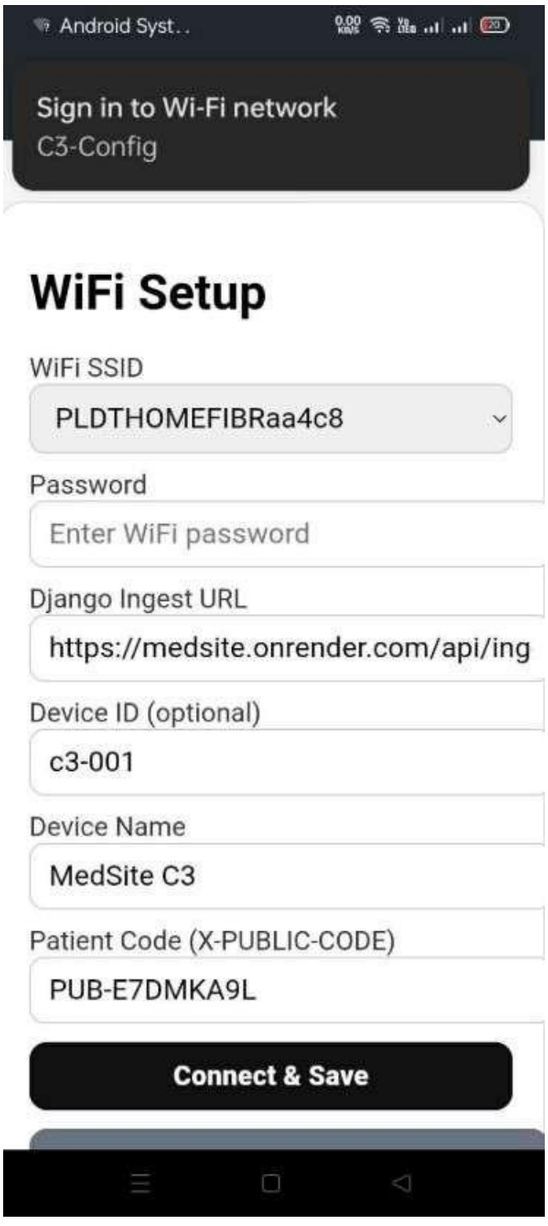
```
# medsite/models.py
from django.db import models
from django.conf import settings
import secrets
from django.utils import timezone

ALPHABET = "ABCDEFGHJKLMNPQRSTUVWXYZ23456789"  # no confusing 0/O, 1/I

def generate_public_code(length=8):
    return "PUB-" + "".join(secrets.choice(ALPHABET) for _ in range(length))

class Patient(models.Model):
    doctor = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name="patients",
    )

    name = models.CharField(max_length=120)
    age = models.PositiveIntegerField(null=True, blank=True)
```

```
    address = models.TextField(blank=True)
    emergency_number = models.CharField(max_length=30, blank=True)

    public_code = models.CharField(max_length=20, unique=True, blank=True, editable=False)
    last_esp32_url = models.URLField(blank=True)
    last_esp32_seen = models.DateTimeField(null=True, blank=True)



    # ☑ archive
    is_archived = models.BooleanField(default=False, db_index=True)
    archived_at = models.DateTimeField(null=True, blank=True)

    def save(self, *args, **kwargs):
        if not self.public_code:
            while True:
                code = generate_public_code(8)
                if not Patient.objects.filter(public_code=code).exists():
                    self.public_code = code
                    break
        super().save(*args, **kwargs)

    def archive(self):
        self.is_archived = True
        self.archived_at = timezone.now()
        self.save(update_fields=["is_archived", "archived_at"])

    def unarchive(self):
        self.is_archived = False
        self.archived_at = None
        self.save(update_fields=["is_archived",  "archived_at"])

class Reading(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE, related_name="readings")
    created_at = models.DateTimeField(auto_now_add=True)

    ir = models.BigIntegerField(null=True, blank=True)
    red = models.BigIntegerField(null=True, blank=True)
    finger = models.BooleanField(default=False)

    bpm = models.IntegerField(null=True, blank=True)
    spo2 = models.FloatField(null=True, blank=True)
    pi = models.FloatField(null=True, blank=True)
    rr = models.FloatField(null=True, blank=True)
    sbp = models.IntegerField(null=True, blank=True)
    dbp = models.IntegerField(null=True, blank=True)
    temp = models.FloatField(null=True, blank=True)
```

How it Works

1. First connect to “C3-Config on your mobile device

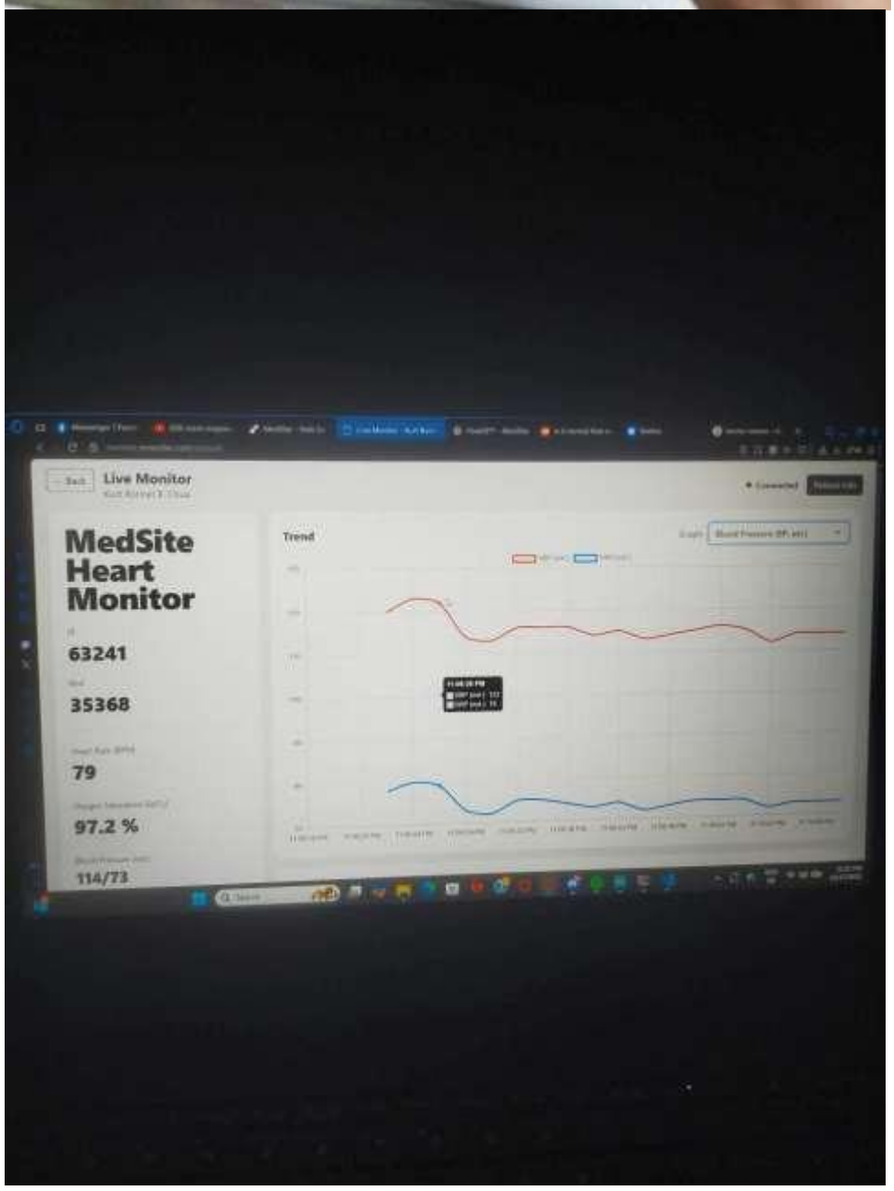
2. You will be redirected to this page, where you connect the device to your WIFI network



3. To check if your device has been connected, go to the website https://medsite.onrender.com, and after creating an account and logging in you will be met with the home screen. Click the blue link “ESP32 Page”, and if a portal opens up, your device is successfully connected, if not, then you may want to reboot it (turn it off and on)

This is the page where you can input patient codes, and tinker with the device’s settings. When creating a patient, they are automatically created with a “Patient Code”, said code can be inputted in the config page to tell the device to display the data on said patient’s monitor.

CREATION PROCESS (IMAGES):


Live Monitor
MedSite
Heart
Monitor
63241
35368
79
97.2 %
114/73
Trend


RAYMAND