Title: Portable IoT Based Health Monitoring System Using ESP32

DESCRIPTION OF THE INVENTION:

This project presents a portable, IoT-enabled health monitoring system that combines multiple biomedical and environmental sensors into a single compact unit powered by the ESP32 microcontroller. The system measures **Heart Rate (BPM)**, **SpO_2**, **Temperature**, **Humidity**, and **GPS Location**, and transmits them to the **Blynk IoT cloud** for real-time monitoring.

A decision-based alert mechanism is implemented to notify emergency contacts whenever abnormal physiological conditions are detected. The prototype aims to support remote patients, elderly individuals, athletes, and outdoor users by offering reliable, low-cost, continuous health tracking.

3. Introduction

Advancements in IoT have transformed healthcare from hospital-centered monitoring into portable, personalized systems. Wearable and mobile health platforms increase accessibility, reduce delays in diagnosis, and allow caregivers to respond faster during emergencies.
This project builds an integrated portable monitoring unit using ESP32 and multiple sensors to continuously measure essential health parameters. Alerts are instantly pushed to the cloud through Wi-Fi, ensuring immediate action during abnormal conditions.

3.1 Problem Statement

Traditional monitoring systems:

- Are expensive
- Require continuous human supervision
- Are not portable
- Often lack integrated multi-sensor capabilities

3.2 Aim of the Project

To design and implement a **portable, real-time, IoT-enabled health monitoring system** with multi-sensor integration and intelligent alerting.

3.3 Objectives

- Measure physiological parameters (BPM, SpO_2) using MAX30102
- Measure temperature & humidity using DHT22
- Acquire GPS coordinates via NEO-6M
- Upload real-time data to the Blynk cloud
- Detect abnormal readings and send alert notifications

DETAILED DESCRIPTION:

This invention is a portable, multi-sensor health monitoring device built with the ESP32 microcontroller. It continuously measures vital parameters such as:

• Blood Oxygen Saturation (SpO_2)
• Heart Rate
• Body Temperature
• Ambient Humidity
• Real-time GPS Location

The ESP32 collects all sensor data, processes it, and sends it to a cloud database using Wi-Fi. An emergency algorithm monitors the readings. If any parameter exceeds a critical limit, the system automatically sends alerts to:

• Nearby hospitals
• Family members
• Emergency responders

The device is compact, battery-powered, and suitable for outdoor use. It is designed for elderly patients, athletes, and people with chronic heart or respiratory issues.

1. Intelligent Health Risk Evaluation Algorithm

The ESP32 runs a custom algorithm that constantly evaluates sensor data to detect:

- Sudden drops in SpO_2 (hypoxemia)
- Abnormal heart rate patterns (tachycardia/bradycardia)
- High fever or thermal stress
- Abnormally high humidity (risk of dehydration or heatstroke)

This multi-parameter analysis goes beyond simple threshold detection and offers more reliable emergency predictions.

2. Data Fusion Technique for Higher Accuracy

The device combines sensor data, pulling information from MAX30102, DHT22, and GPS to create a unified health profile. This reduces false alarms through:

- Moving average filtering
- Noise rejection
- Peak detection algorithms
- Calibration routines

These techniques provide more stable and useful readings.

3. Offline Working Mode (No Internet Mode)

Even when there is no internet or Wi-Fi, the system:

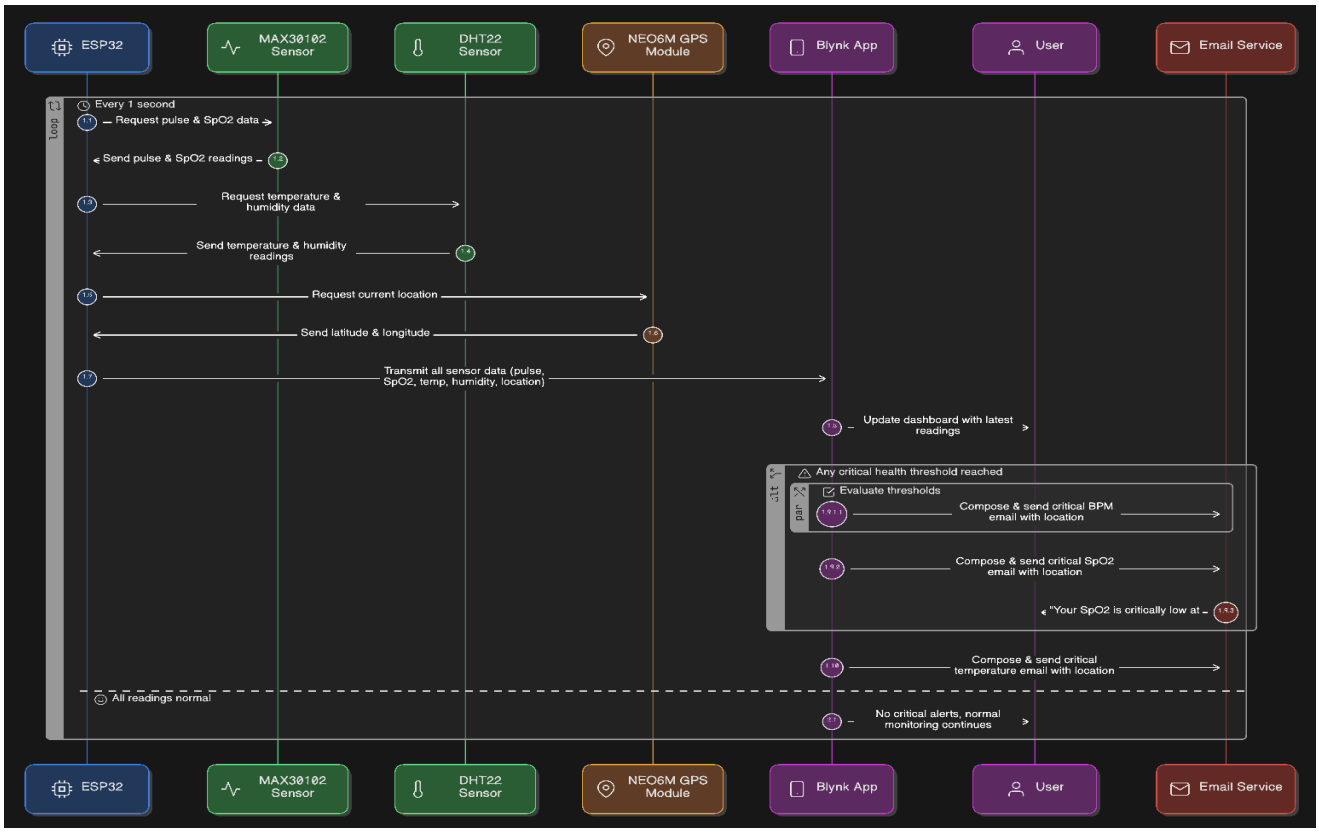
- Continues monitoring
- Stores data in internal memory (SPIFFS or SD card)
- Sends alerts via Bluetooth

When the internet reconnects, the data is automatically uploaded.

4. Cloud Dashboard for Doctors & Family

The device uploads all parameters to a cloud dashboard that allows:

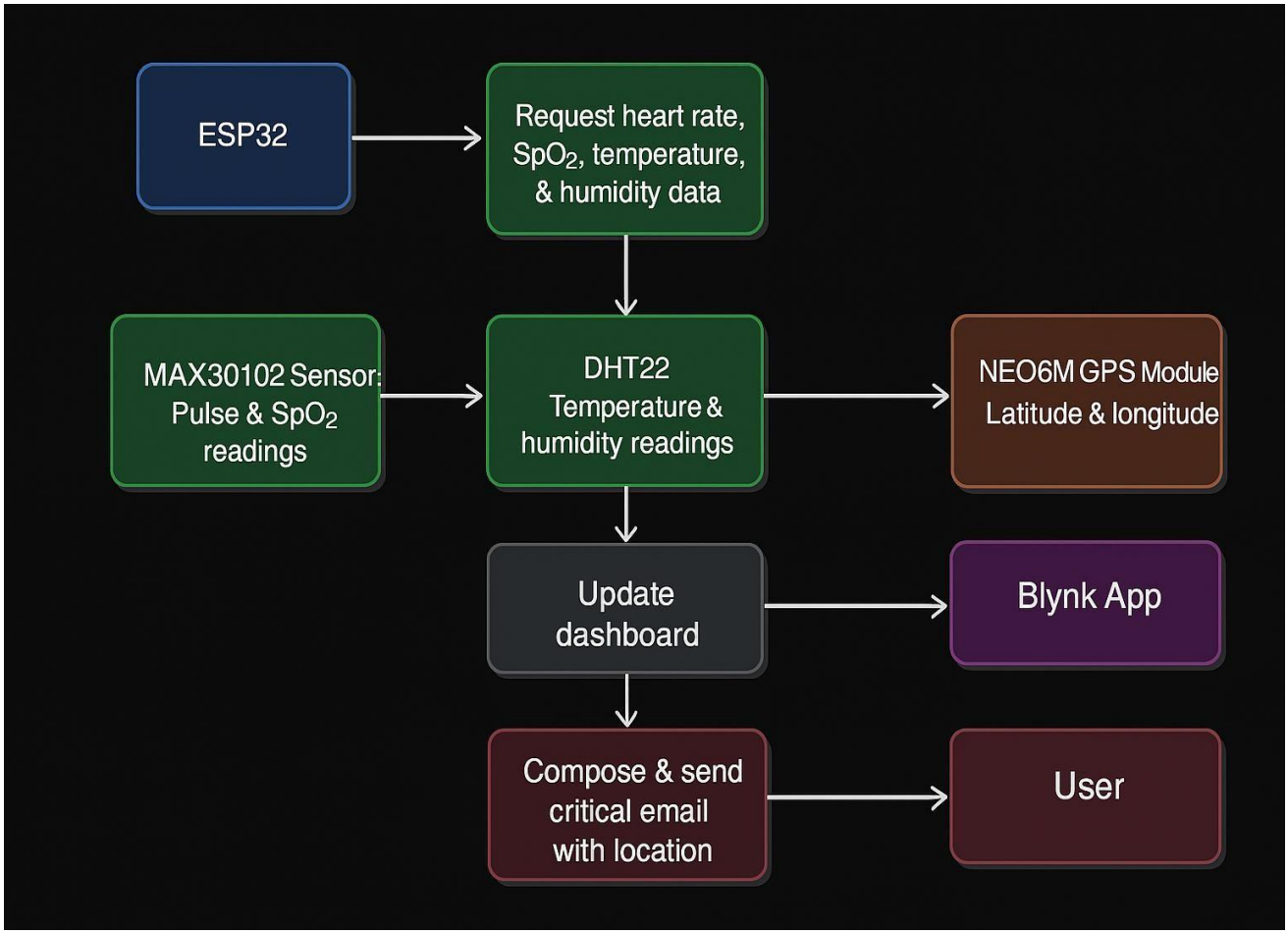
- Timestamp-based graph viewing
- Remote monitoring of the patient
- Automatic daily and weekly health reports
- Comparison with past records for early diagnos.



Block Diagram Explanation:

- Sensors collect raw physiological/environmental data
- ESP32 acquires sensor values using I2C & UART
- Data is processed using algorithms (peak detection, filtering)
- ESP32 connects to Wi-Fi and uploads readings to Blynk
- Dashboard updates continuously
- Alerts are pushed when thresholds are crossed

FLOW CHART OF HARDWARE WORKING:



Flowchart Explanation

1. START / Initialization

What happens

- ESP32 boots and runs setup().
- Connects to Wi-Fi (SSID: Gaurav) via wifi_connect().
- Initializes I²C and scans for the MAX30102 (i2c.scan()).
- Calls init_max30102() to configure registers and LED currents.
- Initializes UART for GPS (UART(1), 9600 baud).
- Initializes DHT22 on pin 26 and internal buffers.

Why it matters

- Ensures sensors and network are ready before entering the monitoring loop.
- If MAX30102 not detected, code prints a diagnostic message but loop continues (safe fail).

2. Main Loop — periodic cycle overview

Loop cadence

- The main loop runs continuously and uses millisecond timers (time.ticks_ms()).
- Key timing constants from the code:
  - SAMPLE_RATE_MS = 10 → MAX30102 samples every **10 ms**.
  - DHT_INTERVAL_MS = 2000 → DHT22 read every **2 s**.
  - BLYNK_UPDATE_INTERVAL_MS = 6000 → push to Blynk every **6 s**.
  - EVENT_COOLDOWN_MS = 60000 → event (alert) cooldown **60 s**.

High-level flow

- Read GPS (if available), read sensor samples at sample rate, update buffers, periodically compute BPM/SpO_2, read DHT, upload to Blynk, check thresholds and send alerts if needed.

3. Read MAX30102 (Sensor Sampling)

Inputs

- I²C FIFO registers on MAX30102.

Processing

- read_sample() reads FIFO pointers and a 6-byte packet (red + IR).
- Samples are stored in circular buffers:
  - red_buf, ir_buf, time_buf sized WINDOW = 300 (computed as (1000/10)*3).
  - idx tracks write index; filled counts how many samples are available.

Key details

- Window represents ~3 seconds of data at 10 ms per sample.
- If I²C read fails, sample stored as 0 (handled gracefully).
- DC/AC separation later uses moving averages (see below).

4. Read DHT22 (Temperature & Humidity)

When

- Every DHT_INTERVAL_MS (2 seconds).

Process

- Up to DHT_RETRIES = 3 attempts per read.
- On success: last_temp and last_hum updated and printed.
- On repeated failure: logs "DHT22 read failed".

Why

- Temperature used for monitoring fever; humidity supports environmental context (dehydration risk).

5. Parse GPS NMEA sentences

How

- UART(1) reads raw NMEA lines.
- parse_nmea_sentence() looks for $GPGGA or $GNGGA.
- Extracts latitude/longitude via ddmm.mmmm conversion to decimal degrees.
- Sets have_fix = True if fix flag indicates a valid fix; otherwise have_fix = False.

Fallback

- If no GPS fix, the system can include FALLBACK_LOCATION_LABEL in alerts if USE_FALLBACK_WHEN_NO_FIX is True.

Edge cases

- Indoor conditions often result in have_fix = False; code handles that by using the fallback label or marking location unknown.

6. Process Readings (Signal processing & feature extraction)

This is the heart of the flowchart: deriving BPM & SpO_2 from raw samples.

a) Detect finger presence

- Compute a short DC estimate of IR using MA_DC = 5 most recent IR samples.
- Compare to NO_FINGER_IR_THRESHOLD = 12000.
- If below threshold → treat as **no finger** and skip heavy computations.

b) DC removal & smoothing

- Convert the filled circular buffer into ordered arrays (ordered_red, ordered_ir, ordered_time).
- Compute DC using moving average: red_dc_series = moving_average(ordered_red, MA_DC) and similar for IR.
- AC component = raw − DC.

c) Additional smoothing

- SMOOTH_MA = 3 may be applied to AC IR series to reduce jitter before peak detection.

d) Peak detection for BPM

- detect_peaks(ac_ir, ordered_time):
  - Computes mean μ and sample standard deviation σ.
  - Threshold = μ + PEAK_STD_FACTOR * σ where PEAK_STD_FACTOR = 0.90.

    - A sample is a peak if it's greater than neighbors and above threshold and also has sufficient prominence.
    - Enforce MIN_PEAK_DISTANCE_MS = 350 to avoid double-counting (min ~171 BPM limit).
- From detected peak times, compute_bpm_from_intervals(peaks_ms) computes average interval and BPM:
    - BPM = int(60000 / avg_interval_ms).
    - Valid BPM only if within MIN_HR = 30 and MAX_HR = 180, otherwise 0.

e) SpO_2 estimation

- compute_spo2(ordered_red, ordered_ir):
    - Compute DC values: red_dc_out, ir_dc_out.
    - If ir_dc_out < NO_FINGER_IR_THRESHOLD, SpO_2 is considered invalid (no finger).
    - AC amplitude = max(AC) − min(AC) for red and IR.
    - Compute ratio R = (red_AC/red_DC) / (ir_AC/ir_DC).
    - Estimate SpO2 = int(110 - 25 * R) then clamp to [70, 100].

Outputs

- bpm and spo2 values; only updated to last_bpm / last_spo2 when valid (>0).

Why it matters

- Combining DC removal, smoothing, and robust peak detection reduces false positives and improves BPM accuracy for a small device.

7. Upload Data to Blynk (Cloud)

When

- Every BLYNK_UPDATE_INTERVAL_MS = 6000 ms.

What is sent

- V0 → BPM (if > 0)
- V1 → SpO_2 (if > 0)
- V2 → Temperature (rounded int)
- V3 → Humidity (rounded int)
- V9 → Event text messages (used for logging alerts)

Mechanism

- HTTP GET calls to: http://blynk-cloud.com/<AUTH>/update/V<n>?value=<val> using urequests.get().

UI benefit

- Blynk dashboard shows live values and terminal/log widgets for events.

8. Threshold Check & Alert Logic

Monitored thresholds in code

- **High BPM**: last_bpm > 55 is used in the event code (note: this is lower than typical medical thresholds; adjustable).

In code's event message condition they use >55 (you may want to change to >130 or add multiple threshold levels).

- **Low SpO_2**: last_spo2 < 85 and >0 (critical alert).
- **High Temp**: last_temp > 25.0 in current code — **this is likely a development/test threshold**; for real use change to > 38.0 or as per requirements.

Event frequency control

- EVENT_COOLDOWN_MS = 60000 ensures the same event isn't sent more than once per minute.

Message contents

- Sensor reading + GPS link if have_fix True: https://maps.google.com/?q=<lat>,<lng>
- If GPS not fixed and USE_FALLBACK_WHEN_NO_FIX True, include fallback address label.

Actions

- Calls blynk_log_event(msg) which writes the text to V9 in Blynk (terminal or value display).

Edge cases & suggestions

- Ensure thresholds are medically appropriate. Current code uses a low BPM trigger (>55) and low SpO_2 <85; you might want multiple severity levels (warning vs critical).
- Include hysteresis (e.g., require consecutive abnormal readings) to reduce false alerts.

9. LED Indicator & UX Feedback

- blink_led(msDelay=80) toggles an onboard LED (LED_PIN = 2) whenever data is pushed to Blynk or an event occurs.
- update_led_state() turns LED off after led_off_time.
- Provides tactile/visual confirmation the device is active and data has been transmitted.

10. Error Handling & Robustness

I^2C / Sensor read errors

- read_bytes() and write_reg() wrap I2C ops in try/except and return None/False on failure.
- When read fails, default values (0) are used; system keeps running.

DHT retries

- Up to DHT_RETRIES = 3 attempts with small waits between retries.

GPS parsing

- Minimal GGA parser ignores malformed lines and sets have_fix = False if no valid fix.

Network

- wifi_connect() retries for up to timeout seconds and prints status. If Wi-Fi is not connected, Blynk writes will fail silently but code continues.

Data continuity

- Circular buffer preserves a sliding window of samples even if a few reads fail.

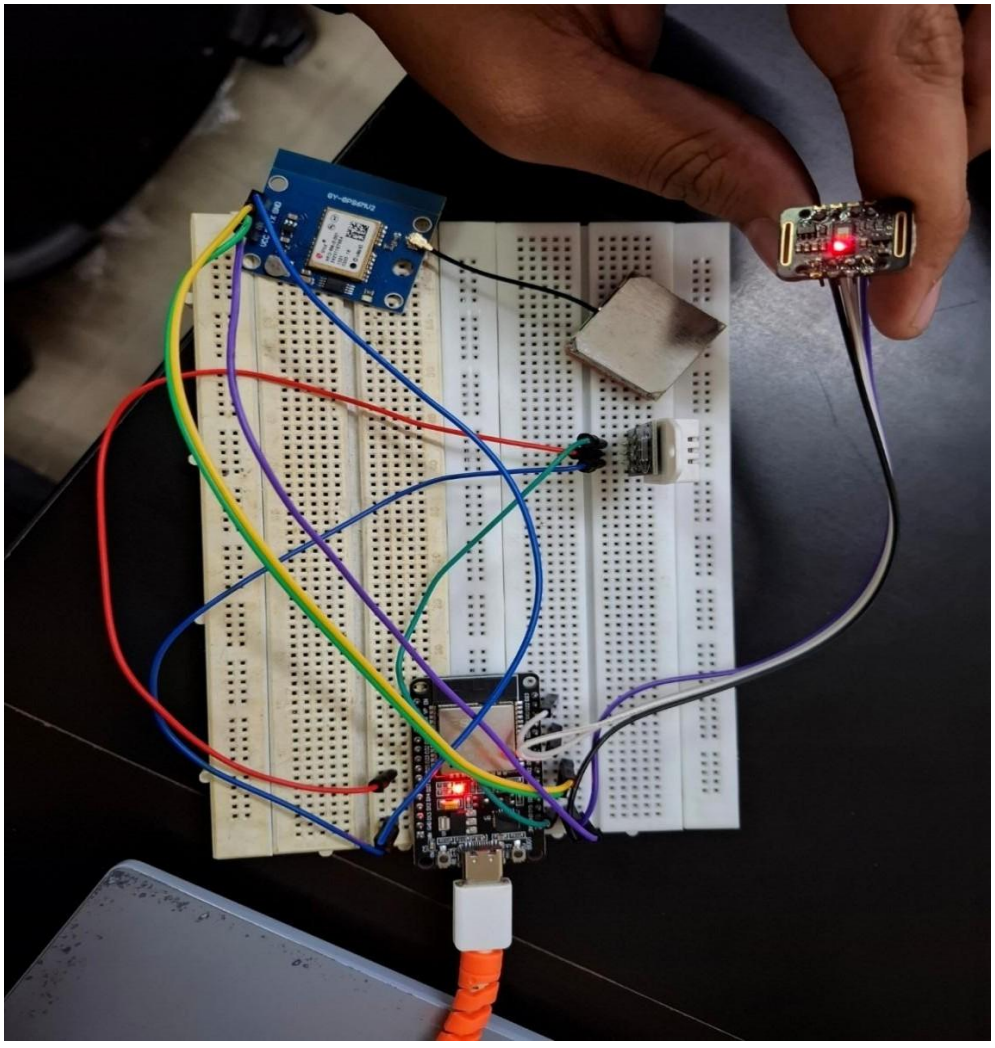
11. Practical Tips & Notes (for report / presentation)

- **Window & sampling**: WINDOW = 300 samples ≈ 3 seconds of data (10 ms sampling).

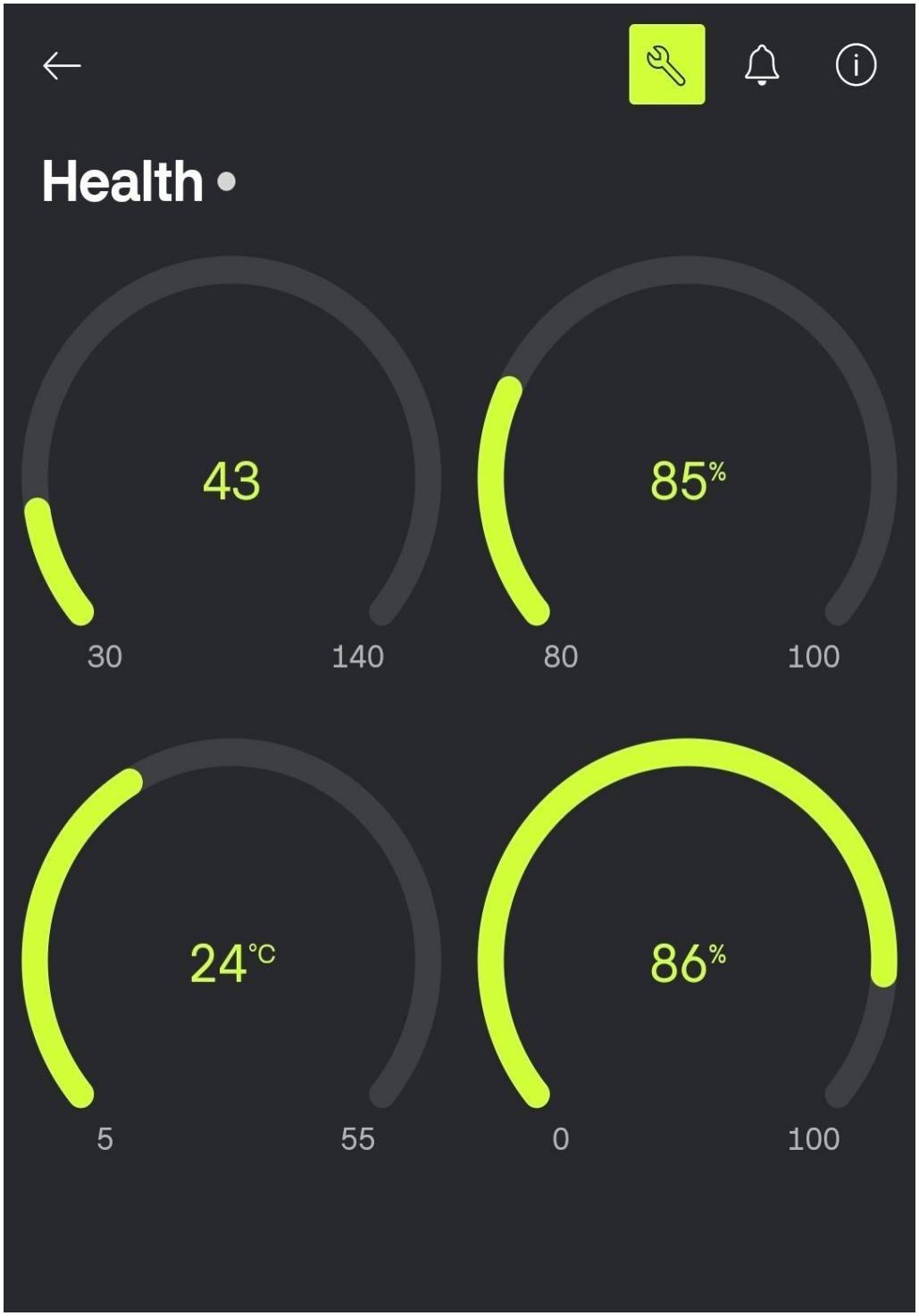
This is the sliding window used for peak detection and SpO_2 calculation — include this in your report to justify responsiveness vs stability tradeoffs.

- **Smoothing vs latency**: MA_DC = 5 and SMOOTH_MA = 3 reduce noise but add smoothing latency; explain that increasing MA sizes improves stability but can delay detection.
- **Threshold tuning**: Provide recommended thresholds in report (e.g., BPM >130 tachycardia, <50 bradycardia; SpO_2 <92% watch, <85% critical; Temp >38°C fever).
- **GPS fallback**: Explain why fallback address exists (indoor GPS weakness) and state privacy considerations when sharing precise location.
- **Power**: Mention battery supply & regulator choices; the MAX30102, ESP32, GPS have different current draws — propose a 2000–4000 mAh Li-ion for multi-hour operation.

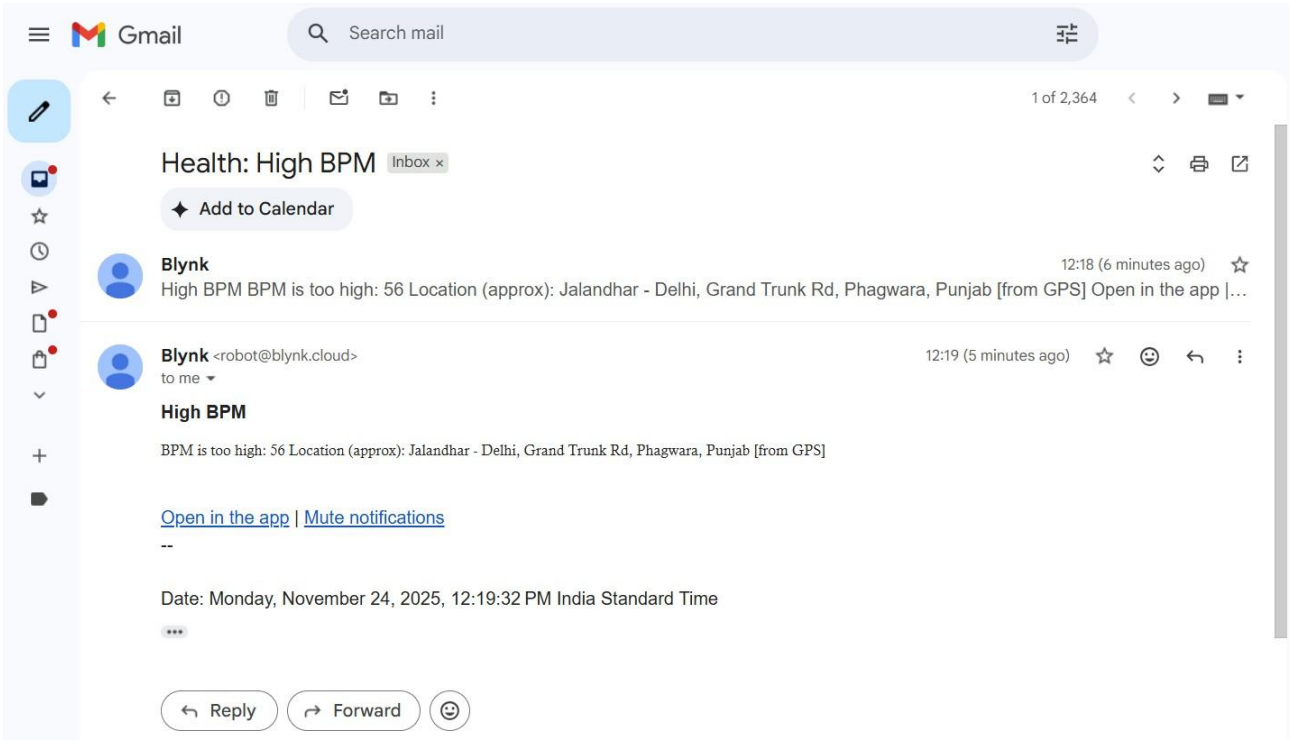
Hardware pic:

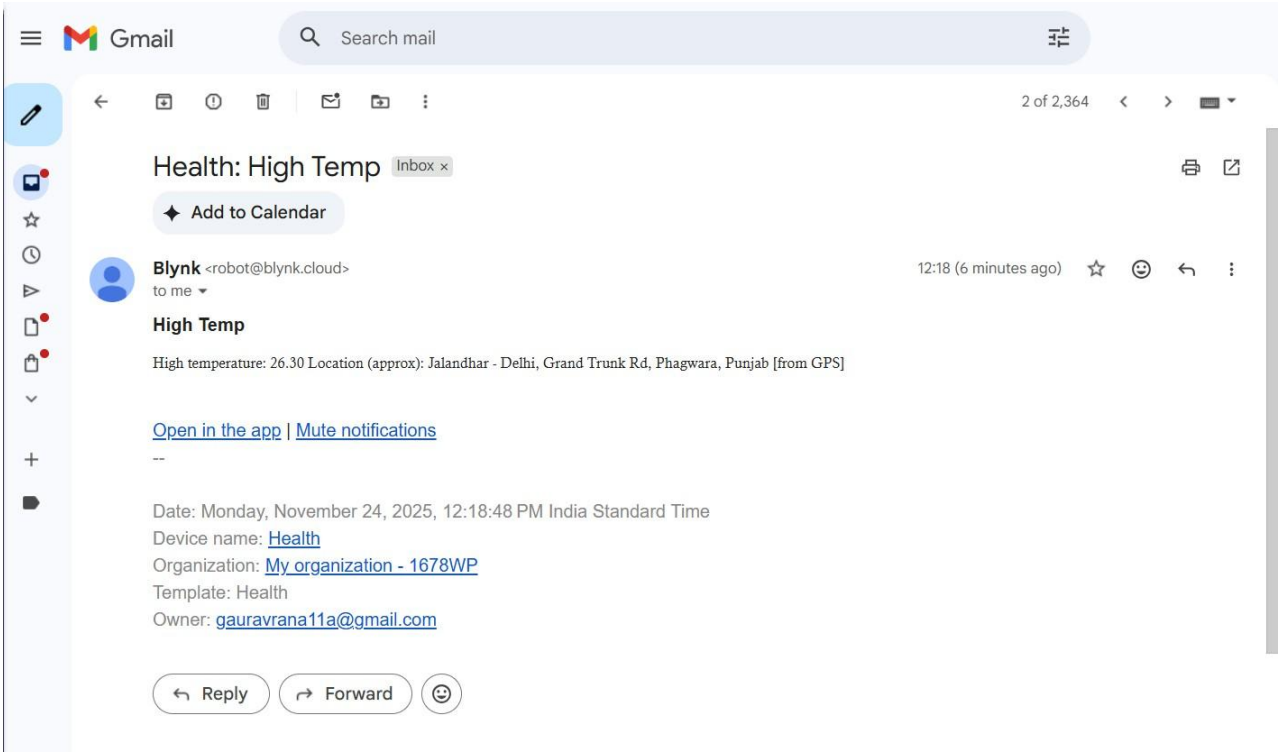


Blynk Integration:



Alert System





System Structure And Components

Power Supply Block:

A rechargeable Li-ion battery powers:

- ESP32
- MAX30102
- DHT22
- NEO-6M GPS
- OLED/LCD (optional)

A voltage regulator ensures 3.3V stable output for sensors.

MAX30102 Pulse Oximeter Sensor Block:

The MAX30102 sensor measures:

- SpO_2 levels
- Heart rate (BPM)

It sends digital data to ESP32 via I^2C.

Temperature Sensor (DHT22):

Measures:

- Body/ambient temperature
- Environmental humidity

Used for detecting fever or dehydration risks.

GPS Module (NEO-6M)

Provides continuous:

- Latitude
- Longitude
- Speed

Used for emergency location tracking.

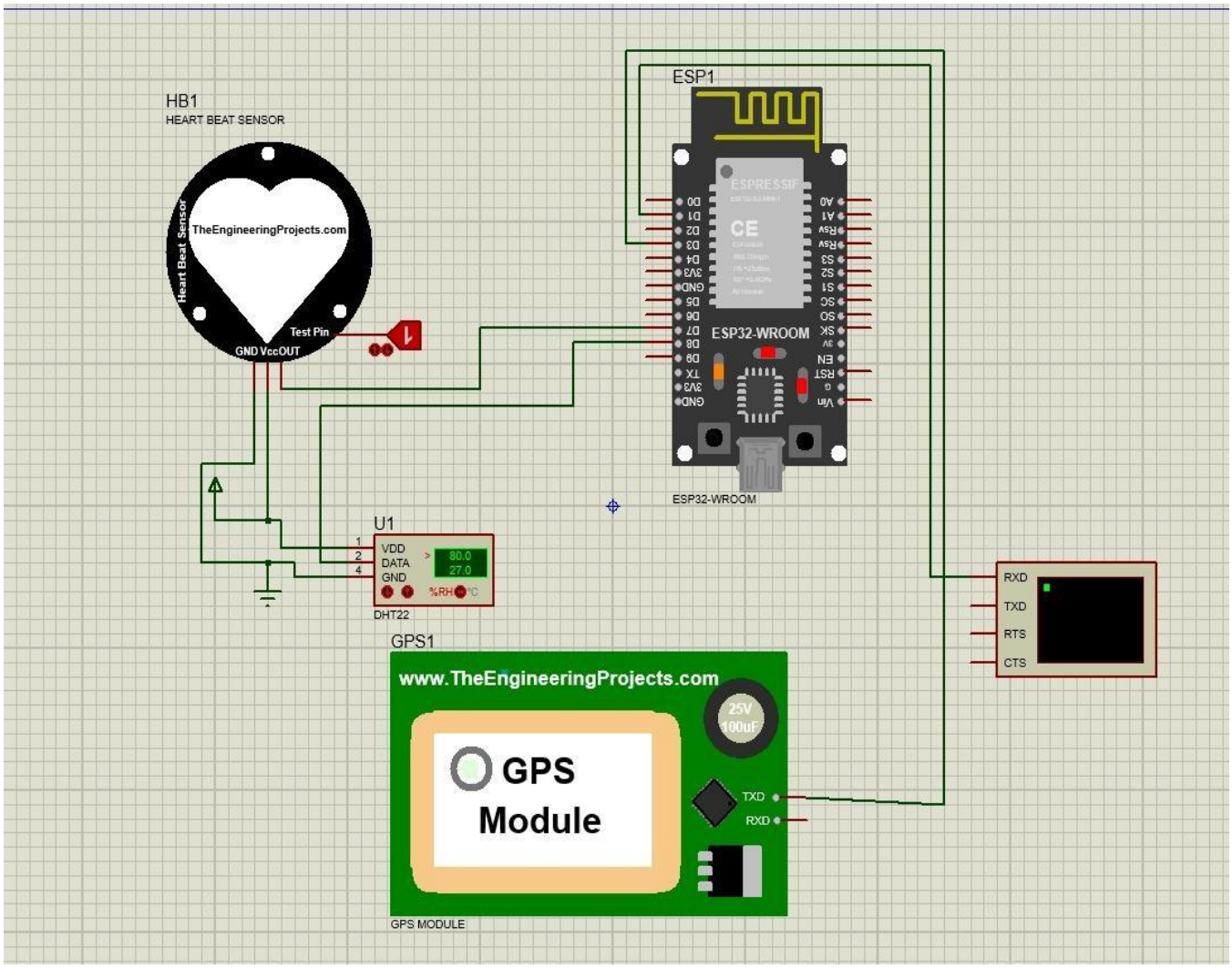
Controller (ESP 32 MCU)
Main functions include:
• Collecting sensor data
• Running alert detection algorithms
• Sending data to cloud IoT platforms
• Triggering SMS/Email alerts
• Logging long-term data
Built-in Wi-Fi enables real-time monitoring.

F) Data Logging & IoT Cloud Block

ESP32 uploads data to:
•Thingspeak/Firebase/Blynk/MQTT platforms
• Mobile application dashboard
Benefits include remote monitoring and medical analysis.

G) Emergency Alert Module
When abnormal readings occur:
•ESP32 sends alert messages
•Includes health data + live GPS location
• Notifies hospital or family instantly
Thresholds example:
•HR < 50 or > 130
•SpO_2<92%
• Temperature > 38°C (fever)

Circuit Diagram (Proteus):



Circuit Diagram Description:

ESP32 Microcontroller (Centre of Circuit):

The ESP32-WROOM acts as the main control unit, interfacing with all sensors and peripherals.
Important features in the circuit:
It receives input signals from the Heart Beat Sensor / MAX30102, DHT22 sensor, and GPS module.
It processes all sensor data and runs threshold-evaluation algorithms.
Its GPIO pins are used for sensor communication:
Digital pins for DHT22.

The ESP32 is powered through its 3.3V regulator supplied from Vcc.
Its built-in Wi-Fi module (shown on the top) is responsible for cloud data uploading and alert messaging.

B) Heart Rate Sensor / MAX30102 (Left Top)

The heartbeat sensor module is connected to the ESP32 via:
Vcc pin → 3.3V supply
GND pin → Common ground
OUT / DATA pin → ESP32 GPIO pin
This sensor measures:
Pulse rate using infrared light
Blood oxygen saturation (SpO_2)
Photoplethysmogram (PPG) waveforms
The test output pin indicates heartbeats visually using an LED (shown in red on the diagram).
The ESP32 continuously collects heartbeat data every second.

C) DHT22 Temperature–Humidity Sensor (Bottom Middle)

The DHT22 module has three terminals:
VDD – Connected to 3.3V supply
DATA – Connected to a GPIO pin on ESP32
GND – Connected to system ground
This sensor provides:
Body/ambient temperature
Relative humidity
The circuit shows a capacitor connected to stabilize the power and ensure accurate temperature readings.
The sensor's real-time temperature and humidity values appear on its module display.

D) GPS Module NEO-6M (Bottom Right)

The GPS module is connected to the ESP32 using UART serial communication:
TXD of GPS → RX pin of ESP32
RXD of GPS → TX pin of ESP32
Vcc pin → 5V supply
GND pin → Ground
The GPS continuously provides:
Latitude
Longitude

Time data
The diagram also includes a decoupling capacitor (25V 100uF) near the GPS module to filter noise and ensure stable operation.

E) Virtual Terminal (Bottom Right Corner)

All modules share a common GND line connecting:
ESP32
Heartbeat sensor
DHT22
GPS module
This common reference ensures stable and noise-free signal communication across the system.

F) Power Supply Connections

The power supply is divided into:
3.3V for ESP32, Heartbeat sensor (MAX30102), and DHT22
5V for GPS NEO-6M module
Regulated power ensures each sensor receives its required voltage for proper operation.

Working Principle

The Portable Health Monitoring and Emergency Alert System operates by continuously sensing the user's vital health parameters and environmental conditions, processing the collected data in real time, evaluating potential health risks, and transmitting both normal and emergency information through an IoT platform. The major functional blocks work together as described below:

Sensor Data Acquisition

Heart Rate & SpO_2 Measurement (MAX30102 / Heartbeat Sensor)
The ESP32 requests pulse and oxygen saturation readings from the MAX30102 sensor at fixed intervals.
The sensor emits infrared light and measures the reflected intensity through a photodiode.
The raw photoplethysmogram (PPG) waveform is processed to extract:
• Heartbeat rate (BPM)
• Blood oxygen saturation (SpO_2)
These readings are transferred digitally to the ESP32.

Temperature & Humidity Reading (DHT22 Sensor)

The ESP32 then requests temperature and humidity data from the DHT22 module.
The DHT22 sends calibrated digital readings.
Temperature helps detect fever, while humidity indicates dehydration or environmental stress.

Real-Time Location Detection (NEO-6M GPS)

The ESP32 establishes UART communication with the GPS module.
It requests the current coordinates.
The GPS module sends valid latitude and longitude.
This location data is used for emergency alerts and patient tracking.

2. Pre-Processing and Filtering

After receiving the data, the ESP32 performs:
Noise removal from MAX30102 signals
Smoothing using moving averages
Filtering of temperature spikes
Verification of valid GPS lock
This ensures accuracy and reliability of all health parameters.

Real-Time Data Integration

The ESP32 combines:
SpO_2
Heart rate
Temperature
Humidity
GPS coordinates
into a single structured data packet.
This packet represents the user's complete health status at that moment.

Transmission to IoT Cloud (Blynk App / Server)

The ESP32 sends the integrated data packet to the cloud through Wi-Fi.
The readings are displayed instantly on the user's dashboard.
Trends and graphs are updated continuously.
Doctors or family members can view the data remotely.

Threshold Evaluation & Emergency Detection

The ESP32 continuously checks if any reading exceeds a preset medical threshold:

SpO_2 < 92% → Low oxygen alert
HR < 50 or > 130 BPM → Heart rate abnormality
Temperature > 38°C → Fever alert
The microcontroller evaluates each sensor output and determines if the user is in a potentially dangerous condition.

Automated Alert Generation

If any critical threshold is crossed:
ESP32 composes an emergency alert message.
It attaches:
Current SpO_2 / HR / Temperature
GPS coordinates
Timestamp
The alert is sent through:
Email service
SMS (if compatible gateway is used)
App notifications
This ensures immediate help from hospitals or emergency contacts.

RESULTS AND ADVANTAGES

1. Hardware Implementation Results

- MAX30102 accurately measures heart rate and SpO_2.
- DHT22 provides precise temperature/humidity readings.
- GPS module gives stable location data.
- ESP32 successfully uploads real-time data to the cloud.

By:-

- **Gaurav Singh Rana (12324801)**
- **Hemant Kumar (12324886)**
- **Vivek Kumar (12325500)**
- **Shubham Kumar (12326211)**