EMBEDDED SYSTEMS PROJECT

BCSE305L

Digital Assignment

SMART HEALTH MONITORING SYSTEM

Submitted To: Prof. Sivanesan S

Slot: A1 + TA1

Date: April 13, 2025

Submitted By:

1) Swarit Jain (22BCE0465)
2) Abhinav Goyal (22BCE0018)
3) Vedant Phalke (22BCE2210)

Contact No.: +91 96519 18555

Abhiram Goyal 22BCE0018

SMART HEALTH MONITORING SYSTEM

Abstract

What → This project presents the design and implementation of an IoT-based remote health monitoring system using multiple biomedical sensors integrated with ESP32 development board. The system monitors key health parameters including blood oxygen level (SpO2), pulse rate, body temperature, ECG signals and body weight.

How → The system utilizes MAX30102 sensor for SpO2 and pulse rate detection, MLX90614 infrared sensor for non contact body temperature measurement, AD8232 for ECG signal acquisition and HX711 module with 10 kg load cell to measure body weight. All the collected data is processed via ESP32 and transmitted online db on Supabase. A Python Flask based web app is used as front end interface for real-time monitoring.

Why → With increasing needs for continuous and remote patient monitoring. this system aims to provide an affordable, real-time, and accessible healthcare solution. It is particularly useful for elderly patients, chronic disease management & post-operative care where regular checkups may be challenging.

Problem Analysis

Problem Statement → Access to timely and accurate health monitoring remains a critical challenge, particularly in remote and underdeveloped regions. Traditional healthcare models rely heavily on hospital visits for diagnosis & monitoring which is not feasible for all patients - especially elderly, chronically ill or those recovering post surgery. Regular check ups can become burdensome, costly and impractical, resulting in delayed diagnosis or untracked health issues.

Outcomes -

1) Comprehensive Vital Sign Monitoring → Successfully implemented real-time tracking of SpO_2, pulse rate, body temperature, ECG and body weight using non-invasive biomedical sensors.
2) Reliable sensor integration with ESP32 → Achieved smooth, concurrent data acquisition from multiple sensors using ESP32 µController, ensuring efficient processing.
3) Real-time cloud connectivity → Enabled secure, continuous data transmission & storage using SupaBase, allowing remote access to health metrics from any location.
4) Interactive and intuitive user dashboard → Developed a Python Flask-based web app that displays live patient data through an easy-to-use, responsive interface for healthcare professionals.
5) Cost Effective and Scalable Healthcare Solⁿ → Designed a portable, low-cost system suitable for rural and remote deployment, contributing to improved healthcare accessibility and reduced hospital dependency.

Abhinav Goyal 22BCE0018

Objective →

1) To design a non-invasive, easy-to-use system for continuously monitoring key vital signs.

2) To reduce dependency on frequent hospital visits through remote data collection and analysis.

3) To ensure real-time health tracking by integrating sensors with ESP32 microcontroller.

4) To provide a centralized, cloud-based database for storing and accessing patient data securely.

5) To build a simple and intuitive web interface using Python Flask for healthcare providers to view patient health trends and anomalies in real-time.

Real World Scenario →

Imagine a diabetic patient with a history of heart disease living alone in a remote village. During winter, they develop mild symptoms like fatigue and chest discomfort, which might otherwise be dismissed.

→ The SpO_2 and Pulse sensor detects abnormal oxygen levels and elevated heart rate.

→ The ECG sensor captures irregular heart rhythms.

→ The Temperature sensor notes an increase in body temperature indicating possible infection.

→ The Weight sensor detects rapid weight changes - common in heart failure cases.

This data is instantly uploaded to Supabase, and an alert is generated in Flask web app, which is monitored by their physician. The doctor then arranges immediate intervention.

Objective - to - Outcome mapping :-

Objective: To develop a simple, non-invasive system to track critical health indicators.
↳ A basic, convenient embedded system based on non-invasive sensors could be created that is suitable for classes of users, such as patients and older people with no technical expertise.

Objective: To integrate multiple biomedical sensors with ESP32 microcontroller to acquire real time data.
↳ MAX30102 (pulse, SpO_2), MLX90614 (IR temperature), AD8232 (ECG), load cell and HX711 (weight sensing) were all successfully connected to ESP32 to access real-time health data.

Objective: To securely transfer health data to a cloud platform.
↳ Successful data transmission to a Supabase cloud database. It ensured continuous storage and retrieval of patient health data remotely.

Abhiram Goyal 22BCE0018

Objective: To create a user interface for watching things happen in real-time with a Python Flask app.
↳ A real-time, interactive frontend was created wherein patients and healthcare professionals can look at health trends and vital signs in real time.

Objective: To render healthcare more accessible by offering a low cost and easy-to-carry monitoring system.
↳ The final result was an affordable, mobile and scalable system that can be deployed in rural and underserved communities, enhancing access to healthcare.

Scenario →

Consider an elderly patient living in rural area with limited access to healthcare facilities. The system is installed at their home and continuously monitors their vital signs. The patient's temperature, heart rate, ECG pattern and weight are collected and uploaded to Supabase cloud db. A doctor, located miles away, logs into Flask based dashboard and can instantly view any anomalies in patient's health. If abnormal values are detected, medical action can be taken remotely.
This scenario highlights importance of remote monitoring, early detection and improved access to healthcare, particularly for vulnerable populations.

Existing IoT-based health monitoring system

In recent years, IoT has emerged as a key driver of innovation in healthcare sector. From advanced hospital telemetry systems to at-home health tracking solutions, there is a growing demand for affordable, real-time and remotely accessible patient monitoring technologies.

One such example is discussed in paper titled "IoT-based Health Monitoring System". The system described uses basic components like Arduino UNO, ESP8266, WiFi module, and simple analog sensors like LM35 (temperature) and LM358 (heart rate). These components work together to collect patient vitals and upload them to ThingSpeak, an online IoT data visualization platform.

While this system effectively demonstrates fundamental concept of remote health monitoring, it remains a basic prototype with limited functionality. Its simplicity offers valuable insights into architecture of IoT-enabled healthcare devices, but also highlights several limitations - such as limited parameters coverage, lack of clinical grade accuracy and minimal security measures. These are the very challenges that our project seeks to overcome through a more advanced, integrated and scalable solution.

Objectives of Existing System →

The core objectives of the system under review are -

1) To design a low cost health monitoring device using commonly available sensors.

2) To monitor two key health parameters: heart rate and body temperature.

3) To enable remote real-time access to health data using cloud technology (ThingSpeak).

4) To promote the integration of wireless technology (Wi-Fi) into basic patient monitoring applications.

5) To demonstrate a proof of concept for deploying IoT in small-scale health solutions.

System Illustration →

Hardware Components used →

1) Arduino UNO → Acts as primary controller to gather sensor data.

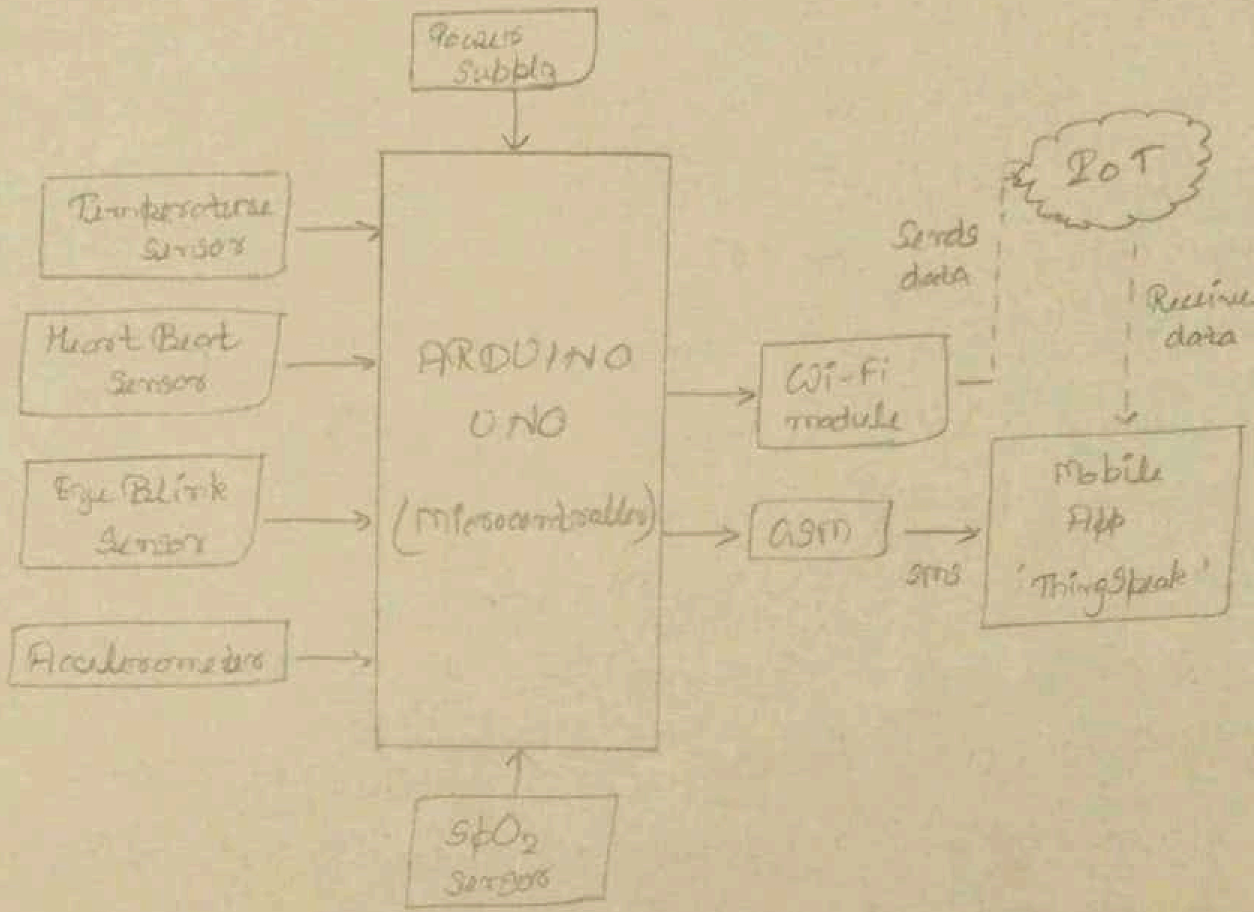
2) LM35 sensor → Analog sensors for measuring body temperature.

3) LM358 Pulse sensor → Detects heart rate through fingertip pulses.

4) ESP8266 Wi-Fi module → Sends data to cloud for remote access.

5) Power supply (9V) → Powers the microcontroller and sensors.

Software / Cloud → ThingSpeak is used as cloud IoT platform to log and visualize incoming sensor data in real time using line graph.



Working Process →

1) Patient places a finger on the pulse sensor and temperature sensor is in contact with the body.
2) Arduino reads analog data from LM35 and LM358 sensor.
3) Data is formatted and sent to ESP8266 WiFi module.
4) ESP8266 uploads data to ThingSpeak via internet.
5) Doctors or caregivers access the ThingSpeak channel to view real-time vitals.

22BCE2210 — Vedant Rahul Phalke

Issues & constraints

Despite its usefulness as basic system, several significant limitations & challenges are observed in this design:

1) Technological Constraints :-
   - Limited sensor integration ⇒ ignores SpO2, ECG & weight o
   - low processing capacity ⇒ Arduino Uno has a basic 8-bit microcontroller with limited memory & speed, which constrains expansion or concurrent sensor processing
   - Analog Signal Noise ⇒ LM358 is prone to motion-including environmental noies.

2) Security & Data Management :-
   - No Security measures ⇒ Data is transmitted in plain format over wifi with no encryption or authentication protocol.
   - No backup or redundancy : single-point failure risk with no database replication or local cache.

3) User Experience Limitation :-
   - Raw visualization: Things speak dashboard are basic and not tailored for real clinical use d
   - No Alert = system lacks real-time notification or threshold-based alerting for abnormal value
   - No mobile support = the system is not optimized for patients or doctors use on smartphones.

4) Clinical Limitation :-
   - Not clinically Callibrated : LM35 & LM358 are not approved for clinical accuracy & have ±1°C & ±bpm inaccuracies, respectively
   - No Decision support : The system does not provide analytics or decision-making assistance to the doctor or caregiver;

22BCE2210 — Vedant Rahul Phalke

* Gap Analysis

Area	Limitations in Existing system	Your Project's Adventuse.
1) Parameter monitoring	only temperature & heart rate monitered	measures SPO_2, ECG, temperature, weight, & plus rate with medical-grade sensor.
2) microcontroller	Ardino UNO (low memory / speed)	uses ESP32 with Dual-core CPU, wifi + Bluetooth & higher RAM
3) Cloude platform	uses ThingSpeak, limited customization & scalability	uses supabase (realtime DB with authentication, storage).
4) frontend Interface	Thingspeak graphs, no branding, no alerts, no patient records	full customer flask frontend with charts, using logins, alerts, & real-time dashbords.
5) Security	No encryption / authentication implemented	Flask + supabase stack allows JWT-based auth & secure HTTPS connections.
6) Alerts & Decision supports	None. No threshold monitoring or response action	Flask app can include threshold-based email/sms for abnormal reading.
7) Power & mobility	Arduino + ESP8266, high power use, not optimized for wearables	ESP32 is low-power, suitable for wearables or mobile kits.

** Additional Insigts

Project moves beyond just data collection & remote access - it embraced a user-centred, scalable, & clinically designs.

→ supabase for real-time cloud DB, authentication, Storage

→ flask for an intuitive interface with user accounts, dash bords, & patient records

→ ESP32 for robust multi-sensor integration with wifi - Bluetooth

→ High quality biomedical sensor (MAX 3012, AD8232, MLX90614) (HX 711).

Date = 13-4-2025

Sivanesan S

22BCE2210 — Vedant Rahul Phalke.

Conclusion

The reviewed system provides a strong education foundation for building IoT-based health monitoring devices. However, it is limited in terms of functionality, scalability, accuracy & user experience. Project demonstrate a significant leap by resolving these issues & adding advanced capabilities in terms of sensors integration, data visualization, security & remote health intelligence.

Thus, your system is not just an improvement - it's a next-gen solⁿ that aligns with the future of telehealth & remote patient care.

22BCE2210 — Vedant Rahul Phalke

Proposed System :-

Smart Healthcare Monitoring System.

* Introduction :-

In modern world, healthcare system are undergoing a major transformation with the integrated technology. However, traditional health monitoring methods still suffer from several limitations that hinder their effectiveness, especially in remote or resource-constrained environment.

Conventional system are :-

- Manual & Time-consuming :- Vitals like temp, pluse & weight are measured using seperate devices, increases time.
- Lack of integration :- No unified system for measuring multiple vitals, leading to inconvenience.
- No Real-Time monitoring :- Inability to transmit data prevents timely medical intervention.
- No Central Data logging & Limited Remote Access.

To overcome these limitations, we developed a Smart Health Monitoring System using the embedded system, which is capable of sensing multiple vital parameters such as:

* Body Temperature - [MLX90614]
* Pulse & SpO2 levels - [MAX30102]
* Weight "
* ECG Signals [AD8232]

The system provides

key improvements over pre-existing system includes.

1) Automation of all vital sign measurements
2) Real-time data logging & cloud integration.

Date = 13-4-2025

Sivaniesan S

22BCE2210 – Vedant Rahul Phalke

3) Remote accessibility through web interface
4) Graphical representation of ECG for better analysis
5) Error reduction through sensor-based digital data collection.

In essence, this system aim to bridge gap between patients & health care providers by offering a compact, low-cost, real-time sol^n.

* Components :-

Hardware components

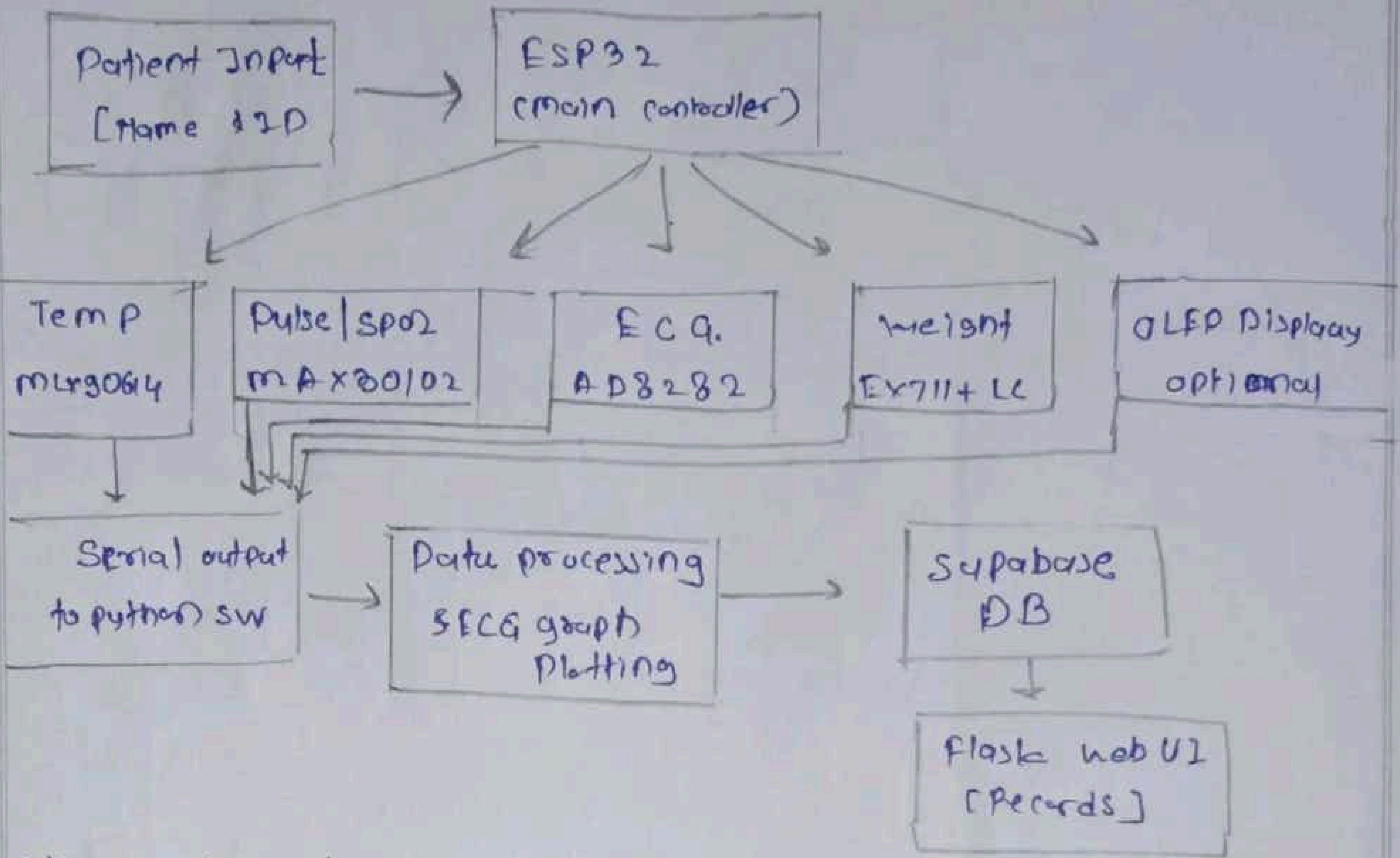
1) ESP32 Development Board = central microcontroller that collects sensor data & communication
2) MAX30102 Sensor = measures SpO_2 & pulse rate.
3) MLX90614 Temp. sensor = Contactless sensor used to measure body temperature.
4) AD8232 ECG sensor = Capture the electrical activity of the heart (ECG)
5) HX711 module + Load cell = measures body weight.
6) OLED Display = Display real-time sensor reading directly on device
7) Connecting wires, Breadboard, USB cable.

Software components

1) Arduino IDE = used to write and upload code to ESP32 microcontroller.
2) python 3.x = Handles GUI input, serial communication, data processing.
3) Tkinter (Python module) = creates the desktop GUI for entering patient ID & name.
4) Matplotlib (Python Library) = plots ECG data & save in PNG.
5) Supabase = cloud-based backend for real-time database storage & access.
6) Flask (Python web framework) - Serial Library (pyserial)

22BCE2210 — Vedant Rahul Phalke

* Block Diagram [Conceptual Architecture]



- the system starts with PythonGUI where the user enters the Patient's name & ID. once submitted, a signal is sent to the Esp32, which activates all connected sensors:
  - → MLX90614 (Body temp)
  - → MAX30102 [SpO2 & heart rate)
  - → AD8232 [ECG signals]
  - → HX711 & load cell
  - → OLED display.

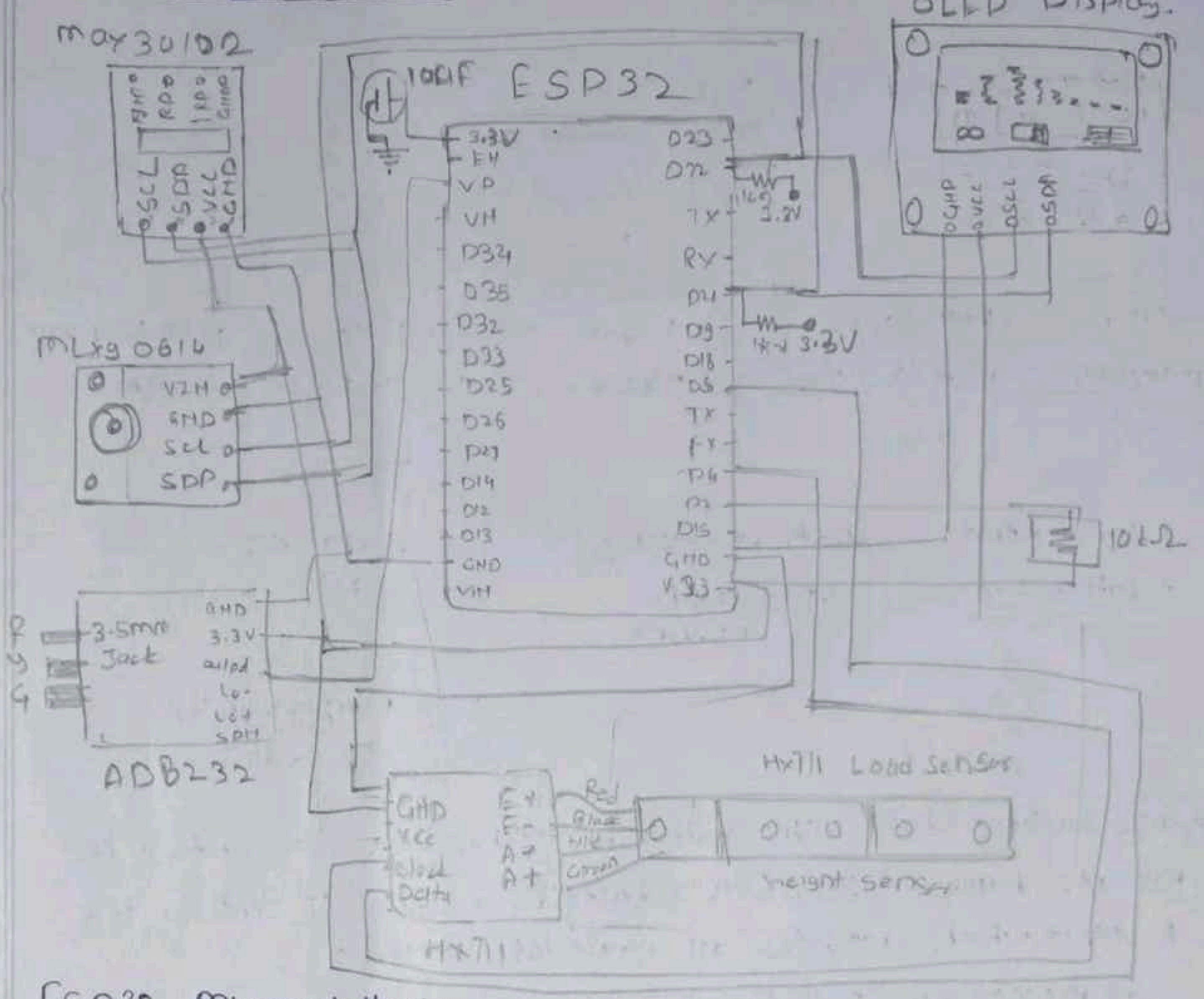
The ESP32 sends all reading to the Python application via Serial communication, which:

- processes the data
- plots the ECG graph using matplotlib
- uploads the complete data (including ECG Graph) to supabase database.

A Flask Web application then fetches this data, allowing Dr doctor to view vitals & ECG reports remotely.

22 BCE 2210 – Vedant Rahul Phalke

* Component Architecture (Circuit Architecture)



ESP32 Microcontroller

The central unit that connects & controls all sensors, processes data & sends it to python interface via Serial.

Max30102 → SPO2 & Pulse Sensor & IR Temperature

vcc → 3.3V   SCL → GPIO22 (I2C)
GND → GND   SDA → GPIO21 (I2C)

Sensor uses I2C communication to send pulse rate & SpO2 levels to the ESP32

AD8232 – ECG Sensor module

33.V → 3.3 V (ESP32)   Output → A0 | GPIO36
GND → GND

captures analog ECG waveform data from the body & sends it to ESP32 for plotting.

22BCE2210 — Vedant Rahul Phalke

HX711 + load cell - weight sensor

VCC → 3.3V   DT → GPIO4
GND → GND   SCK → GPIO5

used to measure weight through strain gauge signals amplified by HX711

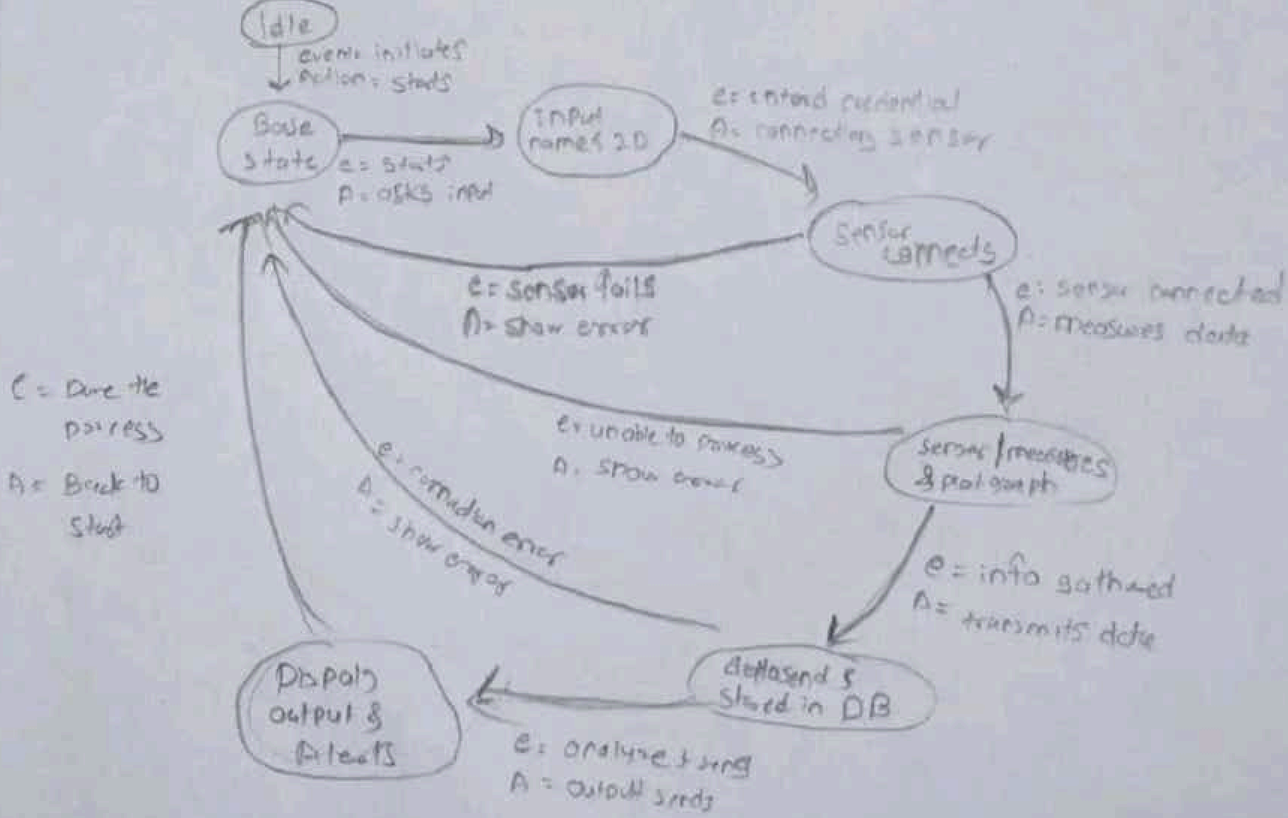
OLED - Display (output)

VCC → 3.3V   SCL → GPIO22
GND → GND   SDA → GPIO21

Display live reading like temp, pulse, SpO2, & weight (optinal for standalone view)

* Why ESP32
- Built-in wifi & Bluetooth
- Dule-core processor
- Rich I/O support
- Low power consumption
- open-source & community support
- cost effective

* FSM [finite state machine]



Sivanesan S

Step-by-Step implementation

Step 1 - Powering the system

- Connect the ESP32 to a laptop via USB.
- This powers the microcontroller and all connected sensors.
- The OLED waits for Patient Info

Step 2 :- Patient Info entry via GUI

- A python GUI (Tkinter) appears asking for :-
  - Patient name
  - Patient ID
- After clicking "submit", this data is sent via Serial (USB) to ESP32

Step 3: ESP32 receives patient data and starts sensing

- Once patient data is received, ESP32:
  - → Initialize all the sensors
  - → Displays the message "Initialising" on OLED.
  - → Starts reading value one by one

Step 4: Sensor Data Collection

1. Temperature sensor (MLX90614)
   - Measures body temperature and sends it to Serial and OLED.
2. Pulse + SpO2 Sensor (MAX30102)
   - Reads heart rate and oxygen saturation
   - Sends data to serial and OLED.
3. Weight sensor (HX711 + Load cell)
   - Measure body weight
   - Sends data to Serial and OLED
4. ECG sensor (AD8232)
   - Sends analog ECG signals.
   - Sends a marker "ECG-START" to inform the Python script.

Step 5: Python script reads serial data

- Python continuously listens to the ESP32's serial output on COM port.
- When "ECG_START" is detected:-
  - ECG data is read and stored in a list for ~15 seconds.
  - A Matplotlib ECG graph is generated and saved as a PNG image.

Step 6: Data packaging

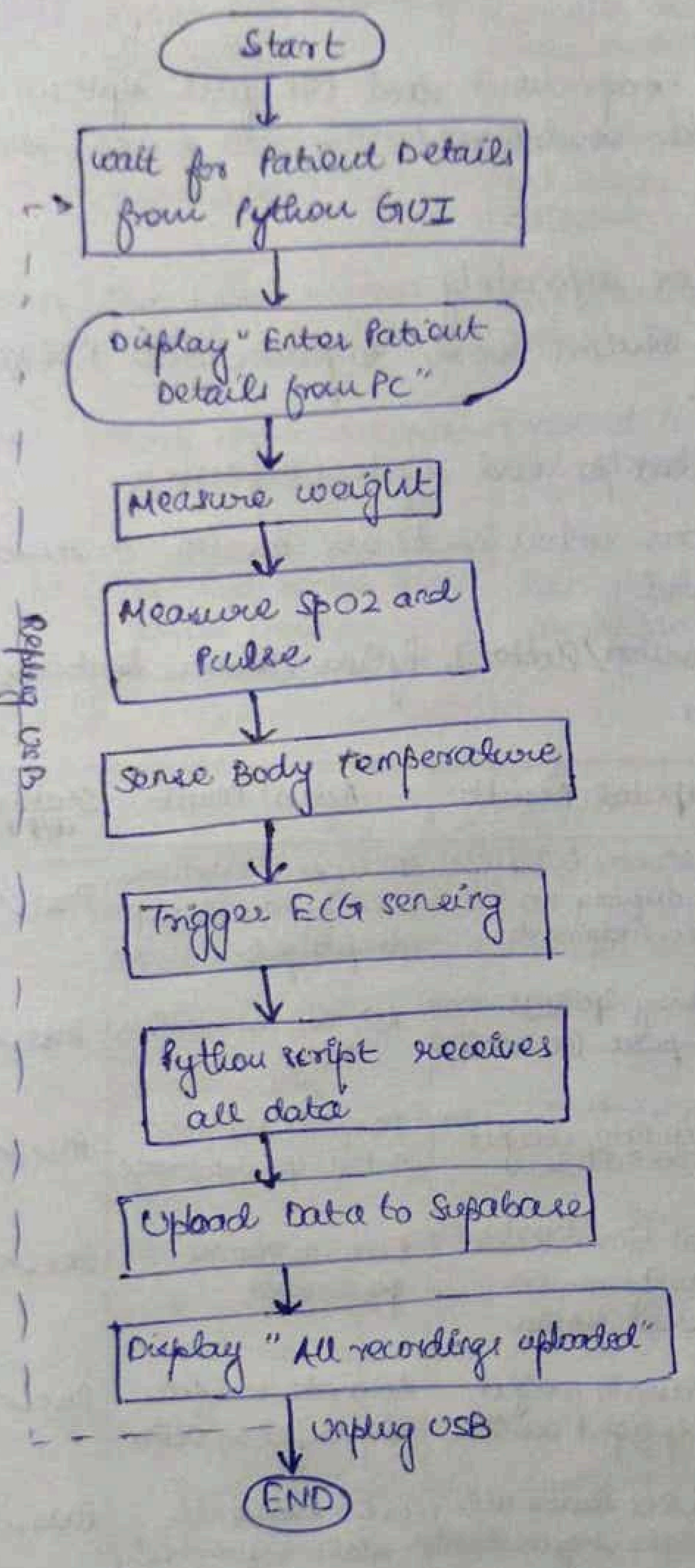
- All the following are packaged in Python as payload for API:
  - Patient name and ID
  - SpO2, Pulse, Temperature and weight
  - ECG Graph (.png)
  - Timestamp

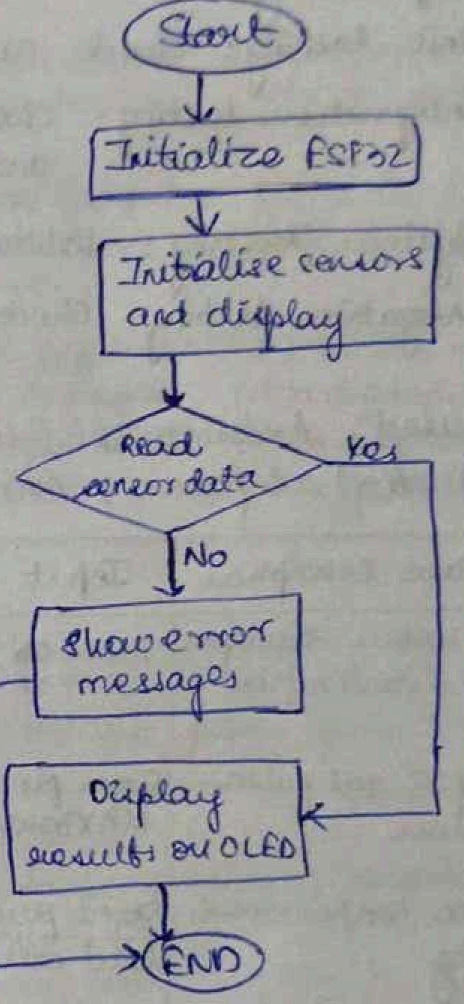
Step 7: Upload to supabase

- The Python script sends the complete package to Supabase DB:-
  - Text data as normal fields.
  - ECG graph PNG image to supabase storage from where the public URL for the stored image is generated.
  - This URL is stored in the ECG_IMAGE_URL field in the database.

Step 8: Data visualization

- A simple Flask Web App shows all the patient records which is globally hosted at "Render" for whose link is been attached in the document.
  - Filter/search by Patient ID and name
  - View timestamped vitals and ECG image.
  - ECG image can be viewed/downloaded.
  - The app user can also download the CSV database data of a specific patient or of all patients which can be used for further data visualizations and data analytics which can help the doctor to give better treatment.


Start
wait for Patient Details from Python GUI
Display "Enter Patient Details from PC"
Measure weight
Measure SpO2 and Pulse
Sense Body temperature
Trigger ECG sensing
Python script receives all data
Upload Data to Supabase
Display "All recordings uploaded"
Unplug USB
END
Replug USB


Start
Initialize ESP32
Initialise sensors and display
Read sensor data
Yes
No
Show error messages
Display results on OLED
END

Flowcharts

Testing (testcases)

→ Purpose of testing: To ensure each component and the full system function reliably under real-world conditions.

→ Types of Testing:

- Unit testing: Check each sensor seperately.
- Integration testing: Check how sensors work together with ESP32 and python.
- System testing: Validate the end-to-end system behaviour
- Negative testing: Check behaviour when inputs are invalid or sensors are disconnected.

→ Tools used: Arduino IDE (serial monitor/Plotter), Python (Tkinter, supabase, matplotlib), supabase, GUI logger.

TC.No	Test Case Description	Input	Expected Result	Actual Result	Status (P/F)
TC01	Verify system startup and sensor initialization	Power ON the ESP32	All sensors initialize and display an acknowledgment	All sensors initialized and "system Ready" displayed.	Pass ✓
TC02	Test SpO2 and pulse detection	Finger placed on MAX30102	Display SpO2 (95-100%) and pulse (60-100 BPM)	All values within range	Pass ✓
TC03	Test IR Temperature reading	Forehead placed near MLX90614	Display temperature (~36-37.5°C)	Temp displayed in the specified range	Pass ✓
TC04	Test signal, of ECG generation	ECG electrodes attached to chest	Real time ECG waveform on serial plotter	ECG waveform generated	Pass ✓
TC05	Test weight measurement	Object placed on load cell	Accurate weight displayed on OLED	Accurate weight displayed on OLED	Pass ✓
TC06	Test OLED display update	Sensor values sensed	OLED shows all vitals sequentially	OLED shows all vitals sequentially	Pass ✓
TC07	Verify Python GUI triggers sensing	Input Patient ID and name in GUI	ESP32 receives start signals and starts sensing.	ESP32 receives "PATIENT_RECEIVED" signals and starts sensing	Pass ✓
TC08	Verify complete vitals sent to python app	Sensor readings collected	Python app should receive vitals and log them	Python script received all vitals and logged them	Pass ✓

TC09	Check ECG PNG image generation	ECG data received over serial	ECG must be saved as png image	ECG saved as .png graph image	Pass ✓
TC10	Check Data upload to Supabase	Patients details and vitals collected	New entry created in Supabase DB	New entry created in Supabase DB	Pass ✓
TC11	Test system response to missing patient inputs	Skip entering patient ID	System doesn't start sensing	System doesn't started sensing	Pass ✓
TC12	Check sensor disconnection handling	Disconnect a sensor mid-operation	Error/time out or fallback handling shown	Errors and timeout fallbacks shown	Pass ✓
TC13	Test end-to-end operational workflow	Run full workflow	All sensors read, data displayed, PNG generated, DB updated	All sensors read, data displayed, PNG generated, DB updated	Pass ✓

- Advantages of the System

1. Real-time monitoring:
   Continuously collect vitals signs (temperature, pulse, SpO2, ECG, weight) in real-time.
2. Automation:
   Minimal manual intervention, system starts sensing automatically after patient info is received.
3. Data Logging and Cloud Storage:
   All readings are stored in Supabase, ensuring secure access and future reference.
4. Compact and Integrated Design:
   Combines multiple sensors into a single place, suitable for bedside or in clinical use.
5. Graphical ECG output:
   Automatically generates ECG graphs which help in quick analysis and diagnosis.
6. User friendly Interface
   Simple Python GUI for entering patient Data, plus OLED display for local feedback.

- Challenges

→ Sensor Calibration:
To maintain accuracy, sensors may need to be calibrated regularly, especially under varying environmental conditions.

→ Noise Handling:
Some sensors, like ECG, may require additional noise reduction techniques to improve signal quality.

→ User Interface:
Further refinement of the user interface for more intuitive user interaction and ease of use can make the system more accessible.

- Future enhancements

To make the Smart Healthcare Monitoring System more scalable, reliable and user-friendly, several technological upgrades can be incorporated. These include wireless communication protocols like ZigBee and WSN for untethered sensor deployment, mobile app integration for better accessibility and cloud-based intelligence for predictive healthcare and remote monitor-ing.

1. Wireless Sensor Networks (WSN) Integration
   → Description: Replace wired sensor connections with wireless communication b/w sensors and ESP32.
   → Benefit: Allows flexible placement of sensors on the body or different rooms, improving mobility and usability.

2. Battery and Portable Power Management
   → Description: Add battery powered operation with charge monitoring
   → Benefit: Makes the system portable for field visits or patient home care.

3. AI-based Health Prediction and Alerts
   → Description: Use machine learning to analyze patient data trends and provide health risk predictions.
   → Benefit: Enables early warning systems for conditions like arrhythmia, fever spikes or oxygen drops.

4. Voice assistance and Emergency calling
   → Description: Add voice control for accessibility and emergency SMS/call feature for critical cases.
   → Benefit: Makes the system usable by elderly or differently-abled patients.

5. Cloud Integration with APIs
   → Description: Add APIs to send data to platforms like Google Fit, Apple Health, or hospital systems
   → Benefit: Ensures interoperability with existing healthcare ecosystems.

6. Data encryption and privacy controls
   → Description: Implement end-to-end encryption and patient consent mechanisms.
   → Benefit: Protects sensitive health data and ensures regulatory compliance (eg. HIPPA)

- Conclusion

The Smart Healthcare Monitoring System developed using the ESP32, various sensors (MAX30102 for SpO2 and pulse, MLX90614 for temperature, AD8232 for ECG, and HX711 with a load cell for weight), and a Python-based GUI provides an efficient and integrated solution for monitoring key vital signs in real-time. The system seamlessly collects and processes data from multiple sensors, displaying the vitals on an OLED screen and enabling further analysis through a Flask-based Python interface.

Key takeways and strengths of the system include:-

1. Multi-Sensor Integration: The system successfully integrates various sensors to monitor essential health parameters like temperature, pulse, SpO2, ECG, and weight. This comprehensive approach allow for real-time tracking of a patient's health status.

2. Real-time monitoring: The data collection is instantaneous, and the system provides real-time display on both the OLED screen (on the ESP32) and the Python GUI, ensuring immediate feedbac

This is particularly valuable for on-the-spot health checks and can support critical decision-making processes.

3. Data storage and visualization: All collected data, including sensor readings and ECG graph images, are securely uploaded to Supabase for easy access and analysis. The use of real-time graphs for ECG data enhances user interaction and helps visualize trends in health data.

4. Scalability and future enhancements: While the current system is designed for monitoring basic health vitals, it is easily scalable to incorporate additional sensors and features. Future enhancements may include the integration of other health parameters (eg. blood pressure, glucose levels), cloud-based data analysis, and real-time alerts for healthcare professionals.

5. Security and privacy: The integration of secure data transmission methods (such as encryption) ensures that patient data remains private and protected, addressing critical concerns in healthcare technology.

- References

1. Datasheet and Technical Manuals:
   - MLX90614 IR Temperature sensor datasheet:- Melexis.
   - MAX30102 Pulse oximeter - Maxim Integrated.
   - AD8232 ECG sensor datasheet - Analog Devices
   - HX711 Load cell Amplifier datasheet - Avia Semiconductors
2. ESP32 Documentation
   ESP32 (38-pin) Technical Reference Manual - Espressif systems
3. Python libraries and Tools
   - Tkinter Documentation - GUI programming
   - Matplotlib documentation - for plotting ECG graphs
4. Supabase Documentation
   - Supabase Docs - Realtime Database and API reference
5. Project based tutorials (for reference/structure):
   - Random nerd tutorials - ESP32 and sensor integration guides.

Existing (similar) system reference

Tamilselvi V., Viru P, Sribalaji S, Geetharamani J., & Vigneshwaran P. (2020). IoT based Health Monitoring System. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India. IEEE.

DOI: 10.1109/ICACCS48705.2020.9074248

ESP32 Code for Smart Healthcare Monitoring System:-

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "HX711.h"
#include <Adafruit_MLX90614.h>
#include <MAX3010x.h>
#include "filters.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define BOOT_DELAY 500

#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Load Cell
const int LOADCELL_DOUT_PIN = 4;
const int LOADCELL_SCK_PIN = 5;
HX711 scale;
float calibration_factor = 205.8;

// Temperature Sensor
Adafruit_MLX90614 mlx = Adafruit_MLX90614();

// MAX30102 Configuration
MAX30105 sensor;
const auto kSamplingRate = sensor.SAMPLING_RATE_400SPS;
const float kSamplingFrequency = 400.0;

const unsigned long kFingerThreshold = 10000;
const unsigned int kFingerCooldownMs = 500;

const float kEdgeThreshold = -2000.0;

const float kLowPassCutoff = 5.0;
const float kHighPassCutoff = 0.5;

const bool kEnableAveraging = true;
const int kAveragingSamples = 5;
const int kSampleThreshold = 5;

// State Management
enum SensorState
{
  WAIT_PATIENT,
  WEIGHT,
  SPO2,
  TEMP,
  ECG,
  IDLE
};
SensorState currentState = WAIT_PATIENT;
```

```
unsigned long stateStartTime = 0;
String ecgData = "";
String patientID = "";
String patientName = "";
const int ECG_PIN = 35;

void showMessage(String msg, unsigned long delayTime = 2000)
{
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println(msg);
  display.display();
  if (delayTime > 0) {
    delay(delayTime);
  }
}

void setup()
{
  delay(BOOT_DELAY);
  Serial.begin(115200);
  Wire.begin();
  while (!Serial);

  Serial.println("\n\nBOOT:Initializing system...");
  Serial.flush();

  // OLED Initialization
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED allocation failed");
    while (1) { delay(100); }
  }

  display.setTextSize(1);
  display.setTextColor(WHITE);
  showMessage("Welcome to Smart Health Monitoring System.", 1000);
  showMessage("Enter patient ID\nand Name via PC", 0);

  // Load Cell Initialization
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  scale.set_scale(calibration_factor);
  if (!scale.wait_ready_timeout(3000)) {
    Serial.println("ERROR:LOADCELL_FAIL");
    while (1) { delay(100); }
  }
  scale.tare();

  // Temperature Sensor Initialization
  if (!mlx.begin())
  {
    showMessage("MLX90614 Error!", 3000);
    while (1) { delay(100); }
  }
```

```
  // MAX30102 Initialization (Daniel Wiese's library)
  if (sensor.begin() && sensor.setSamplingRate(kSamplingRate))
  {
    Serial.println("Sensor initialized");
  }
  else
  {
    showMessage("MAX30102 Error!", 3000);
    while (1) { delay(100); }
  }

  Serial.flush();
  while (Serial.available()) { Serial.read(); }

  Serial.println("READY");
  Serial.flush();
  delay(100);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("System Ready");
  display.println("Awaiting Data...");
  display.display();
  currentState = WAIT_PATIENT;
  stateStartTime = millis();
}

LowPassFilter low_pass_filter_red(kLowPassCutoff, kSamplingFrequency);
LowPassFilter low_pass_filter_ir(kLowPassCutoff, kSamplingFrequency);
HighPassFilter high_pass_filter(kHighPassCutoff, kSamplingFrequency);
Differentiator differentiator(kSamplingFrequency);
MovingAverageFilter<kAveragingSamples> averager_bpm;
MovingAverageFilter<kAveragingSamples> averager_r;
MovingAverageFilter<kAveragingSamples> averager_spo2;

// Statistic for pulse oximetry
MinMaxAvgStatistic stat_red;
MinMaxAvgStatistic stat_ir;

// R value to SpO2 calibration factors
// See https://www.maximintegrated.com/en/design/technical-documents/app-
notes/6/6845.html
float kSpO2_A = 1.5958422;
float kSpO2_B = -34.6596622;
float kSpO2_C = 112.6898759;

// Timestamp of the last heartbeat
long last_heartbeat = 0;

// Timestamp for finger detection
long finger_timestamp = 0;
bool finger_detected = false;

// Last diff to detect zero crossing
```

```
float last_diff = NAN;
bool crossed = false;
long crossed_time = 0;

unsigned long spo2StartTime = 0;

void handleWaitPatient()
{
  showMessage("Enter patient ID\nand Name via PC", 0);
  if (Serial.available())
  {
    if (patientID == "")
    {
      patientID = Serial.readStringUntil('\n');
      patientID.trim();
    }
    else if (patientName == "")
    {
      patientName = Serial.readStringUntil('\n');
      patientName.trim();
    }

    if (patientID != "" && patientName != "")
    {
      Serial.println("PATIENT_RECEIVED");
      showMessage("Patient Data Received.\nStarting Vitals...", 2000);
      currentState = WEIGHT;
      stateStartTime = millis();
    }
  }
}

void handleWeight()
{
  showMessage("Weight meas in 2 sec...", 2000);
  delay(2000);

  unsigned long startTime = millis();
  float weightSum = 0;
  int count = 0;

  while (millis() - startTime < 5000)
  {
    if (scale.is_ready())
    {
      weightSum += scale.get_units(5);
      count++;
    }
    delay(100);
  }

  float avgWeight = count > 0 ? weightSum / count : 0;
  avgWeight *= 240;
  display.clearDisplay();
```

```
  display.setCursor(0, 0);
  display.setTextSize(2);
  display.print("Weight:");
  display.setCursor(0, 30);
  display.print(-avgWeight/1000, 2);
  display.print(" g");
  display.display();
  delay(3000);

  Serial.print("WEIGHT:");
  Serial.println(-avgWeight/1000, 2);
  showMessage("Weight recorded.\nNext: SpO2 in 2 sec", 2000);
  currentState = SPO2;
  stateStartTime = millis();
}

void initDrawScreen(void)
{
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.println(F(""));
  display.setCursor(5, display.getCursorY());
  display.setTextSize(2);
  display.println(F("BPM  %SpO2"));
  display.display();
}
bool display_reset = true;
void displayMeasuredValues(bool no_finger, int32_t beatAvg, int32_t spo2)
{
  display.setCursor(5, 35);
  display.setTextColor(WHITE, BLACK);
  if (no_finger)
  {
    display.setTextSize(2);
    display.println(F("NO Finger            "));
    display_reset = true;
    display.display();
  }
  else if (beatAvg < 30 && display_reset)
  {
    display.setTextSize(2);
    display.println(F("Pls.  Wait             "));
    display_reset = false;
    display.display();
  }
  else if (beatAvg >= 30)
  {
    display.setTextSize(2);
    display.println(F("               "));
    display.setCursor(5, 35);
    display.setTextSize(3);
```

```
    display.print(beatAvg);
    display.print(F(" "));
    if (spo2 >= 20 && spo2 < 100)
    {
      display.print(spo2);
    }
    else if (spo2 >= 100) {
      display.print(F("99"));
    }
    else
    {
      display.print(F("--"));
    }
    display.println(F("    "));
    display.display();
  }
}

int average_bpm;
int average_r;
int average_spo2;

void handleSPO2()
{
  if (spo2StartTime == 0)
  {
    spo2StartTime = millis();
    display.clearDisplay();
    initDrawScreen();
  }

  // 20-second measurement window
  while (millis() - spo2StartTime < 20000)
  {
    auto sample = sensor.readSample(1000);
    float current_value_red = sample.red;
    float current_value_ir = sample.ir;

    // Original finger detection and processing logic
    if (sample.red > kFingerThreshold)
    {
      if (millis() - finger_timestamp > kFingerCooldownMs)
      {
        finger_detected = true;
      }
    }
    else
    {
      differentiator.reset();
      averager_bpm.reset();
      averager_r.reset();
      averager_spo2.reset();
      low_pass_filter_red.reset();
      low_pass_filter_ir.reset();
```

```
    high_pass_filter.reset();
    stat_red.reset();
    stat_ir.reset();
    finger_detected = false;
    finger_timestamp = millis();
  }

  if (finger_detected)
  {
    displayMeasuredValues(false, 0, 0);
    current_value_red = low_pass_filter_red.process(current_value_red);
    current_value_ir = low_pass_filter_ir.process(current_value_ir);
    stat_red.process(current_value_red);
    stat_ir.process(current_value_ir);

    float current_value = high_pass_filter.process(current_value_red);
    float current_diff = differentiator.process(current_value);

    if (!isnan(current_diff) && !isnan(last_diff))
    {
      if (last_diff > 0 && current_diff < 0)
      {
        crossed = true;
        crossed_time = millis();
      }
      if (current_diff > 0)
        crossed = false;

      if (crossed && current_diff < kEdgeThreshold)
      {
        if (last_heartbeat != 0 && crossed_time - last_heartbeat > 300)
        {
          int bpm = 60000 / (crossed_time - last_heartbeat);
          float rred = (stat_red.maximum() - stat_red.minimum()) / stat_red.average();
          float rir = (stat_ir.maximum() - stat_ir.minimum()) / stat_ir.average();
          float r = rred / rir;
          float spo2 = kSpO2_A * r * r + kSpO2_B * r + kSpO2_C;

          if (bpm > 50 && bpm < 250)
          {
            if (kEnableAveraging)
            {
              average_bpm = averager_bpm.process(bpm);
              average_r = averager_r.process(r);
              average_spo2 = averager_spo2.process(spo2);
              if (averager_bpm.count() >= kSampleThreshold)
              {
                displayMeasuredValues(false, average_bpm, average_spo2);
              }
            }
            else
            {
              displayMeasuredValues(false, bpm, spo2);
            }
```

```
            }
            stat_red.reset();
            stat_ir.reset();
          }
          crossed = false;
          last_heartbeat = crossed_time;
        }
      }
      last_diff = current_diff;
    } else {
      displayMeasuredValues(true, 0, 0);
    }
  }

  // After 20 seconds, finalize and move to next state
  Serial.print("SPO2:");
  if(average_spo2 > 99) {average_spo2 = 99;}
  Serial.println(average_spo2, 1);
  Serial.print("PULSE:");
  Serial.println(average_bpm);
  displayMeasuredValues(true, average_bpm, average_spo2);
  currentState = TEMP;
  spo2StartTime = 0;
  stateStartTime = millis();
}

void handleTemperature()
{
  showMessage("Temp sense in 2 sec", 2000);
  delay(2000);
  showMessage("Sensing Temp: ");
  delay(1000);

  unsigned long startTime = millis();
  float tempSum = 0;
  int count = 0;

  while (millis() - startTime < 10000)
  {
    float tempC = mlx.readObjectTempC();
    display.clearDisplay();
    display.setCursor(0, 0);
    display.setTextSize(2);
    display.print("Temp:");
    display.setCursor(0, 30);
    display.print(tempC, 1);
    display.print(" C");
    display.display();
    tempSum += tempC;
    count++;
    delay(50);
  }

  float avgTemp = count > 0 ? tempSum / count : 0;
```

```
  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(2);
  display.print("Temp:");
  display.setCursor(0, 30);
  display.print(avgTemp, 1);
  display.print(" C");
  display.display();
  delay(1000);

  Serial.print("TEMP:");
  Serial.println(avgTemp, 1);
  showMessage("Temp recorded.\nNext: ECG in 2 sec", 2000);
  currentState = ECG;
  stateStartTime = millis();
}

void handleECG()
{
  showMessage("ECG meas in 2 sec...", 2000);
  delay(2000);

  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Recording ECG");
  display.display();

  Serial.println("ECG_START");
  ecgData = "";
  unsigned long startTime = millis();

  while (millis() - startTime < 30000)
  {
    int ecgValue = analogRead(ECG_PIN) / 4;
    Serial.println(ecgValue);
    ecgData += String(ecgValue) + ",";
    delay(20);
  }

  if (ecgData.endsWith(","))
  {
    ecgData.remove(ecgData.length() - 1);
  }

  Serial.println("DATA_END");

  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.println("Data uploaded.");
  display.println("Please unplug");
  display.println("device and replug");
  display.display();
```

```
  currentState = IDLE;
}

void loop()
{
  switch (currentState)
  {
  case WAIT_PATIENT:
    handleWaitPatient();
    break;
  case WEIGHT:
    handleWeight();
    break;
  case SPO2:
    handleSPO2();
    break;
  case TEMP:
    handleTemperature();
    break;
  case ECG:
    handleECG();
    break;
  case IDLE:
    while (true);
    break;
  default:
    break;
  }
}
```

Python Script Code:-

```
import tkinter as tk
import serial
import time
import threading
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from datetime import datetime
from supabase import create_client, Client

# ----------------- Supabase Configuration -----------------
SUPABASE_URL = "https://wjqlervxbljhkscrteld.supabase.co"
SUPABASE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6IndqcWxlcnZ4YmxqaGt
zY3J0ZWxkIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NDM4MjgxODUsImV4cCI6MjA1OTQwNDE4NX0.UJiOIJPtD4sY
6Tgscur8RouPi-0jPIIYJuC18uerqyQ"
supabase: Client = create_client(SUPABASE_URL, SUPABASE_KEY)

# ----------------- Serial Setup -----------------
SERIAL_PORT = 'COM5'
BAUD_RATE = 115200
```

```python
print(f"Connecting to {SERIAL_PORT} at {BAUD_RATE} baud...")
try:
    ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
    ser.reset_input_buffer()
    # Send a wake-up pulse (optional)
    ser.write(b'\r\n')
    time.sleep(5)
except Exception as e:
    print(f"Serial connection failed: {str(e)}")
    exit(1)

# Improved ready check
print("Waiting for ESP32 ready signal", end='', flush=True)
start_time = time.time()
while time.time() - start_time < 30:  # 30 second timeout
    if ser.in_waiting:
        line = ser.readline().decode('utf-8').strip()
        print(f"\nReceived: '{line}'")
        if "READY" in line.upper():
            break
    print(".", end='', flush=True)
    time.sleep(0.5)
else:
    print("\nError: Timeout waiting for ready signal")
    print("Possible solutions:")
    print("1. Verify ESP32 is powered and programmed correctly")
    print("2. Check USB cable connection")
    print("3. Press ESP32 reset button")
    ser.close()
    exit(1)

print("ESP32 synchronization complete")


# ---------------- Global Variables ----------------
vitals = {}
ecg_values = []

# ---------------- Function Definitions ----------------
def read_sensor_data(patient_id, patient_name):
    global ecg_values, vitals
    print("Waiting for sensor data from ESP32...")

    recording_ecg = False

    while True:
        try:
            line = ser.readline().decode('utf-8').strip()
            if not line:
                continue
            print("Received:", line)

            if line.startswith("WEIGHT:"):
                vitals["weight"] = float(line.split("WEIGHT:")[1])
```

```
            elif line.startswith("SPO2:"):
                vitals["spo2"] = float(line.split("SPO2:")[1])
            elif line.startswith("PULSE:"):
                vitals["pulse"] = int(line.split("PULSE:")[1])
            elif line.startswith("TEMP:"):
                vitals["temperature"] = float(line.split("TEMP:")[1])
            elif line == "ECG_START":
                print("ECG recording started...")
                recording_ecg = True
                continue
            elif line == "DATA_END":
                break
            elif recording_ecg:
                try:
                    ecg_values.append(int(line))
                except ValueError:
                    pass
        except Exception as e:
            print("Error reading serial:", e)
            break

    print("Sensor data capture complete.")
    print("Vitals:", vitals)
    print("ECG data length:", len(ecg_values))
    ecg_image_filename = generate_ecg_graph(patient_id, patient_name, ecg_values)
    upload_to_supabase(patient_id, patient_name, vitals, ecg_image_filename)
    ser.close()

def generate_ecg_graph(patient_id, patient_name, ecg_data):
    if not ecg_data:
        print("No ECG data captured.")
        return None

    # Calculate sampling parameters
    sampling_rate = 50  # 50 samples/second (1500 samples/30s)
    total_seconds = len(ecg_data) / sampling_rate

    # Create larger figure (width in inches)
    plt.figure(figsize=(20, 6))  # Width x Height (20" wide for better detail)

    # Create time axis in seconds
    time_axis = [i/sampling_rate for i in range(len(ecg_data))]

    # Plot with time axis
    plt.plot(time_axis, ecg_data, label="ECG Signal", linewidth=0.8)

    # Formatting
    plt.xlabel("Time (seconds)", fontsize=12)
    plt.ylabel("ECG Value (ADC units)", fontsize=12)
    plt.title(f"ECG Data for {patient_name} ({patient_id}) - {total_seconds:.1f}s 
recording", fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)

    # Set x-axis ticks every 1 second
```

```
    max_time = int(total_seconds) + 1
    plt.xticks(range(0, max_time, 1))

    # Add vertical lines every 1 second
    for sec in range(0, max_time):
        plt.axvline(x=sec, color='gray', linestyle=':', alpha=0.5)

    plt.legend(fontsize=12)
    plt.tight_layout()  # Prevent label cutoff

    filename = f"ecg_graph_{patient_id}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
    plt.savefig(filename, dpi=150, bbox_inches='tight')  # Higher DPI for clarity
    plt.close()
    print(f"Detailed ECG graph saved as {filename}")
    return filename

def upload_to_supabase(patient_id, patient_name, vitals, ecg_image_filename):
    try:
        with open(ecg_image_filename, "rb") as img_file:
            image_data = img_file.read()

        try:
            storage_response = supabase.storage.from_("ecg-
images").upload(path=ecg_image_filename, file=image_data, file_options={"content-type":
"image/png"})
            ecg_image_url = f"{SUPABASE_URL}/storage/v1/object/public/ecg-
images/{ecg_image_filename}"
            print("ECG image uploaded. URL:", ecg_image_url)
        except Exception as e:
            print("Exception during image upload:", e)
            return

        data = {
            "patient_id": patient_id,
            "patient_name": patient_name,
            "weight": vitals.get("weight", None),
            "spo2": vitals.get("spo2", None),
            "pulse": vitals.get("pulse", None),
            "temperature": vitals.get("temperature", None),
            "ecg_image": ecg_image_url,
            "timestamp": datetime.now().isoformat()
        }
        try:
            response = supabase.table("patient_data").insert(data).execute()
            if not response.data:
                print("Database Insert Warning: Empty response")
            else:
                print("Data uploaded successfully!")
                exit(0)
        except Exception as db_error:
            print("Database Insert Error:", str(db_error))
    except Exception as e:
        print("General Exception:", str(e))
```

```python
def send_patient_info():
    patient_id = entry_id.get().strip()
    patient_name = entry_name.get().strip()
    if patient_id and patient_name:
        time.sleep(1)  # Small buffer

        ser.write((patient_id + "\n").encode())
        time.sleep(0.5)  # Important delay
        ser.write((patient_name + "\n").encode())

        print(f"Sent patient info: {patient_id} | {patient_name}")

        # Wait for confirmation with timeout
        start_time = time.time()
        while time.time() - start_time < 5:
            line = ser.readline().decode('utf-8').strip()
            if line == "PATIENT_RECEIVED":
                break
        else:
            print("Warning: Didn't receive confirmation from ESP32")

        root.destroy()
        threading.Thread(target=read_sensor_data, args=(patient_id,
patient_name)).start()

# ---------------- GUI Setup ----------------
root = tk.Tk()
root.title("Enter Patient Information")

tk.Label(root, text="Patient ID:").grid(row=0, column=0, padx=10, pady=5)
entry_id = tk.Entry(root)
entry_id.grid(row=0, column=1, padx=10, pady=5)

tk.Label(root, text="Patient Name:").grid(row=1, column=0, padx=10, pady=5)
entry_name = tk.Entry(root)
entry_name.grid(row=1, column=1, padx=10, pady=5)

tk.Button(root, text="Submit", command=send_patient_info).grid(row=2, column=0,
columnspan=2, pady=10)
root.mainloop()
```

Ques – 1) Identify uniqueness/novelty of your project with respect to current scenario.

What distinguishes this project today is the full utilization of various health sensors within a single portable and web-connected system. In contrast to most existing solutions that monitor one or two medical signs (such as heart rate or temperature) only, the system monitors SpO_2, pulse rate, body temperature, ECG signals, and body weight simultaneously—all in real-time. By combining these measurements, the system presents a comprehensive picture of a patient's health and assists in discovering issues more accurately and earlier.

Another significant innovation is the utilization of the ESP32, a high-performance microcontroller that can process a large number of sensor inputs and possesses in-built Wi-Fi and Bluetooth. This is superior to previous boards such as the Arduino UNO. Secondly, the project utilizes Supabase, a new scalable cloud database that can execute in real time. It is far more flexible and secure than standard IoT dashboards such as ThingSpeak. Data is stored securely and displayed interactively via a custom-built Python Flask web application. This enables healthcare professionals to view and monitor patient data remotely.

Considering the increasing emphasis on remote care, particularly post the COVID-19 pandemic, the system's design places priority on portability, affordability, real-time accessibility, and usability—suiting the system for implementation in rural and underserved areas. All of these considerations combined make the system innovative and highly relevant, bridging the gap between clinical-grade monitoring and cost-effective, community-level healthcare technology.

Ques – 2) Identify and justify the reasons for choosing all sensors and actuators used in your project.

1. MAX30102 – SpO_2 and Pulse Sensor

Function: Measures blood oxygen saturation (SpO_2) and pulse rate using photoplethysmography (PPG).
Justification:

- **Dual functionality:** Combines pulse and SpO_2 in a single module, reducing the need for multiple sensors.
- **Compact and Non-invasive:** Ideal for continuous and wearable health monitoring.
- **High sensitivity:** Provides accurate readings of pulse and blood oxygen levels even with small variations in blood flow.
- **I^2C communication:** Ensures easy and stable integration with the ESP32.

2. MLX90614 – Infrared Temperature Sensor

Function: Measures non-contact body temperature using infrared sensing.
Justification:

- **Non-invasive monitoring:** Safer and more hygienic, especially important for infectious disease scenarios.
- **Precision:** Offers medical-grade accuracy with ±0.2°C error margin in body mode.
- **Wide temperature range:** Suitable for both body and ambient temperature measurements.
- **I^2C interface:** Seamless compatibility with the ESP32.

3. AD8232 – ECG Sensor

Function: Captures electrical activity of the heart (ECG waveform).
Justification:

- **Medical relevance:** ECG data is vital for detecting arrhythmias, heart stress, and other cardiac issues.
- **Analog output:** Offers real-time ECG signals, which can be visualized and analyzed.
- **Low power consumption:** Suitable for wearable, battery-powered applications.
- **Noise filtering:** Integrated signal conditioning improves accuracy in portable use.

4. HX711 + 10kg Load Cell – Weight Measurement

Function: Measures patient body weight through strain gauge deformation.
Justification:

- **Cost-effective and accurate:** Combines affordability with high sensitivity in measuring small weight changes.
- **Ease of calibration:** Can be zeroed and scaled for accurate medical weight monitoring.
- **Useful for detecting fluid retention:** Especially helpful for patients with heart or kidney issues.
- **Simple digital interface (HX711):** Provides stable readings with 24-bit resolution when connected to ESP32.

Ques – 3) Suggest alternative solution for your idea/project.

1. Alternative Processing Platform: Raspberry Pi

An alternative to using the ESP32 microcontroller is the **Raspberry Pi Zero W** or **Raspberry Pi 4 Model B**. These are single-board computers that support a full Linux operating system and offer significantly higher processing power, memory, and storage capabilities.

This allows local processing of data, advanced visualization, and the use of Python libraries for **on-device analytics**. Raspberry Pi also supports camera modules and USB peripherals, making it ideal for more advanced or clinic-based setups. However, it comes with increased power consumption, size, and cost—making it more suitable for fixed installations than wearable or portable devices.

2. Alternative Cloud Backend: Firebase or AWS IoT Core

Instead of Supabase, cloud services such as **Firebase Realtime Database** or **AWS IoT Core** can be used for storing and synchronizing patient health data.

- **Firebase** offers real-time data synchronization, user authentication, and seamless mobile app integration.
- **AWS IoT Core** provides secure communication, device management, and integration with cloud analytics tools.

These platforms offer high reliability and scalability, particularly for commercial applications or large-scale deployments. However, they introduce complexity in setup and may incur costs beyond free usage tiers.

3. Alternative Frontend Interface: Mobile Application

In place of a Flask-based web application, a **cross-platform mobile application** developed using **Flutter** or **React Native** can serve as the user interface.

Mobile apps improve accessibility for both patients and healthcare providers, offering features like **push notifications, local data caching, offline access**, and **platform-native user experience**. This alternative is especially suitable for users in remote or resource-limited settings who may not have access to desktop devices.

While mobile apps increase development time and maintenance requirements, they offer a more robust and user-friendly interface for long-term use.

4. Alternative Communication Method: Bluetooth Low Energy (BLE)

For scenarios where Wi-Fi is unreliable or power conservation is critical, **Bluetooth Low Energy (BLE)** can be used in place of Wi-Fi. BLE-enabled sensors can transmit data to a nearby **gateway device** such as a smartphone or Raspberry Pi, which then uploads the data to the cloud.

BLE drastically reduces energy consumption, making it suitable for **wearable health monitoring**. However, its limited range and dependence on a constant gateway device may restrict its use in some environments.

5. Alternative Sensor Modules

The individual sensors used (e.g., MAX30102 for SpO_2 and pulse, AD8232 for ECG) can be replaced with more integrated or specialized alternatives:

- **MAX86150** integrates pulse, SpO_2, and ECG in a single module.
- **TMP117** offers high-precision body temperature readings.
- **Smart weight sensors** or compact load sensors can be embedded into wearables for continuous body mass tracking.

These alternatives can reduce wiring complexity and board size, making the system more compact and suitable for wearable applications.

Ques – 4) Coin/Prepare a single liner to promote your idea/project.

Smart care in real time, right from your fingertips.

Ques – 5) Briefly try to sell your product.

Imagine being able to check on a loved one's health—even from miles away. Our smart health monitoring system makes that possible.

It tracks key vital signs like oxygen level, heart rate, temperature, ECG, and weight using non-invasive sensors, and sends the data instantly to a secure online platform. Doctors and caregivers can view everything in real time through a simple dashboard—no hospital visit needed.

Whether it's for elderly parents, patients in recovery, or people living in remote areas, this system helps catch problems early, reduces hospital trips, and gives peace of mind—all in a compact, affordable device.

It's healthcare that watches over you—even when no one's around.