P.B.M.S.

Personal Bio-Monitoring System

An "IoT" approach to monitoring vitals

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	08/01/2020	Report, Documentation and Manual

USER MANUAL

EDBERT HANS, JORDY FILBERT, SAMUEL PUTRA, STEFAN LIEMAWAN

The user needs to simply strap the device and put on the ECG pads and turn on the power switch.

Full Code and Schematic:

We fully accept the changes as needed improvements and authorize initiation of work to proceed. Based on our authority and judgment, the continued operation of this system is authorized

Edbert Hans	DATE
2101684494	
Stefan Liemawan	DATE
2101694690	
Samuel Putra	DATE
2101693630	
Jordy Filbert	DATE
2101693542	

PRODUCT SUMMARY

GitHub Repository: https://github.com/thesamuelputra/P.B.M.S..git

The PBMS is an IoT-based Personal Bio-Monitoring System, that would allow an individual's vital signs to be non-invasively monitored & displayed in real time to a command center monitored by an admin. The admin would be able to check the subject's vital signs over the internet. The device would be useful for firefighters, military, biological and radioactive units, as well as other 'command center based' services.

The subject would wear the device on his left upper arm using an armband. The device contains a NodeMCU V0.9 microcontroller and three sensors connected to it to collect data from both from the subject's vitals and the surrounding environment for the administrator to monitor.

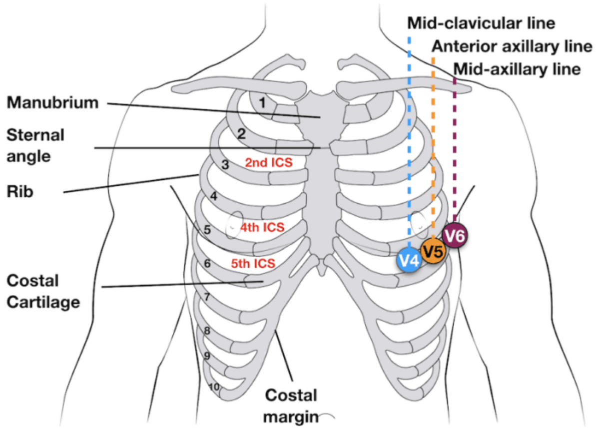
The IoT functionality of the device will be provided using the ESP-12F Wi-Fi module built within the NodeMCU. The ESP-12F is an alternative to Arduino's proprietary Wi-Fi shield, it is much smaller & comes with the NodeMCU development board and allows for a much easier troubleshooting.

The subject's heart will be monitored using an ECG sensor, specifically the AD8232 and comes with the noise reduction board. An ECG can identify several conditions, these include; respiration rate, a clog of some sort in a heart's blood supply, a past heart condition, enlargement of one side of the heart & abnormal heart rhythms. We opt to use an AD8232 ECG, as ECGs can easily be noisy and the AD8232 board acts as an op amp to help obtain a clearer signal from the heart easily.

The subject's body temperature will be monitored using a digital temperature sensor. In this case, we opt to use the LM75A digital temperature sensor. The LM75A sensor can be configured to work in different operational conditions, ambient temperature and contact temperature and has a high temperature resolution.

The surrounding air quality will be monitored using a digital air quality sensor. We chose the CCS811 sensor, as it is a low-power digital sensor that senses a wide variety of VOCs (Volatile Organic Compounds). The onboard CCS811 board has been optimized for low power consumption during an active sensor measurement & has an idle mode which further extends the battery life of the device.

RESEARCH // ELECTRODE PLACEMENTS



The ECG is one of the most useful investigations in medicine. Electrodes attached to the chest and/or limbs record small voltage changes as potential difference, which is transposed into a visual tracing.

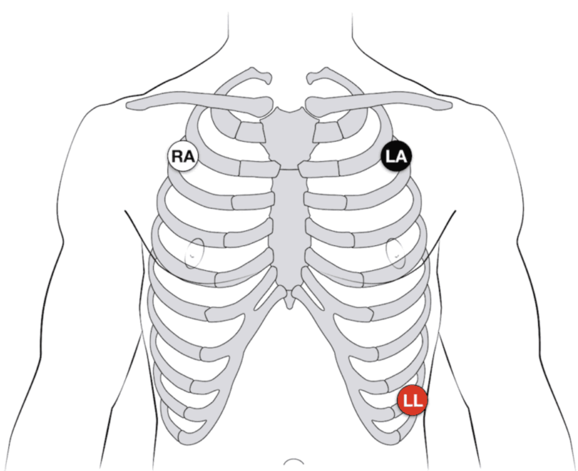


Figure 1. Optimal Electrode Placements

After some experimentation and research, we found that to get the best results with 3 leads we need to stick the electrodes on the chest wall equidistant from the heart (rather than specific limbs).

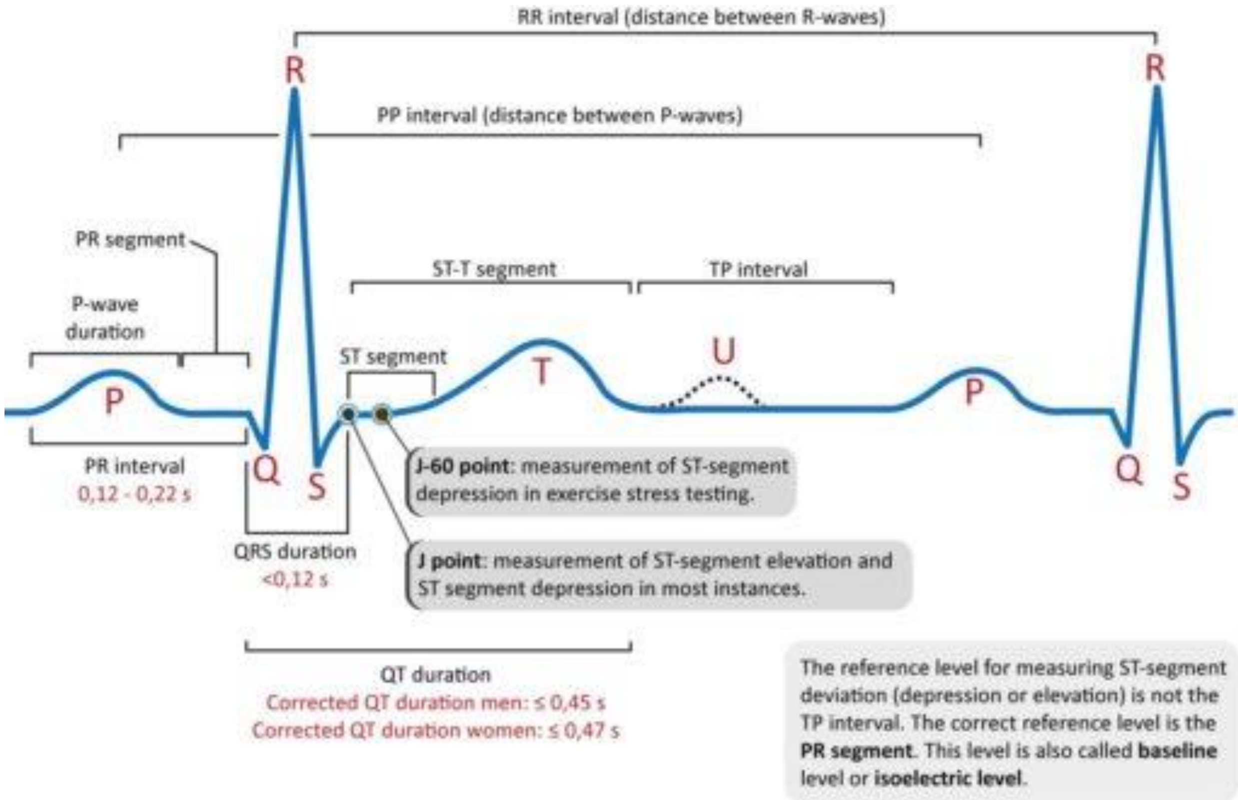


Figure 2. Different Areas of an ECG

CCS811 SPECIFICATION

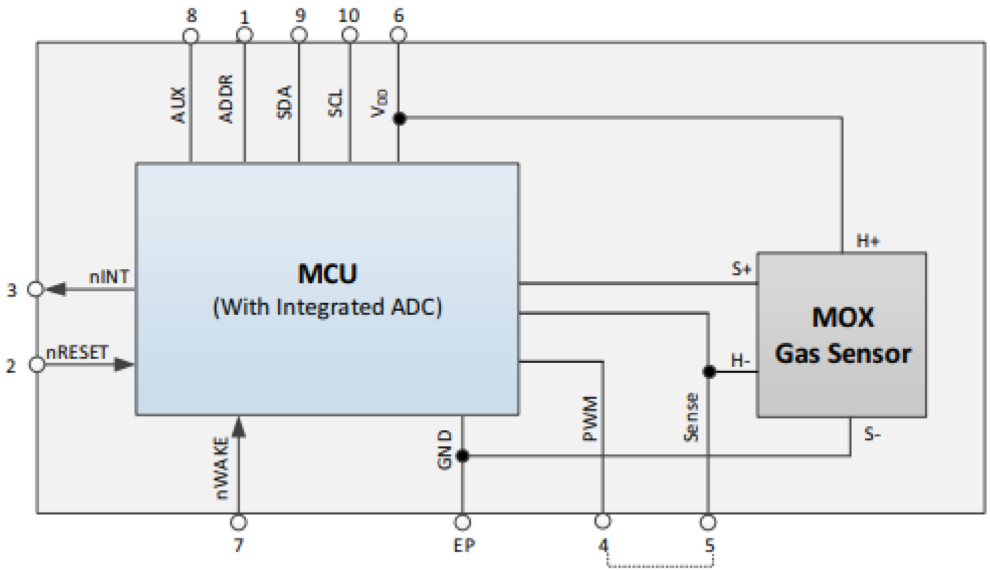


Figure 3. CSS811 Block Diagram

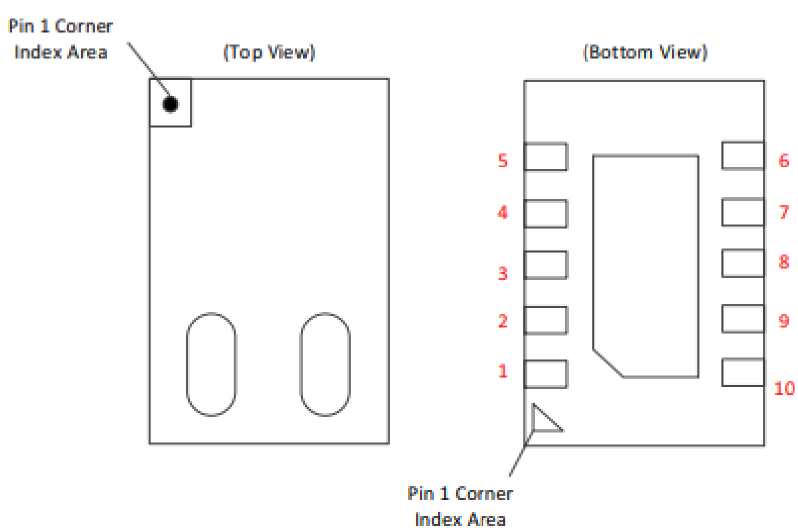


Figure 4. CSS811 Pin Diagram

LM75A SPECIFICATION

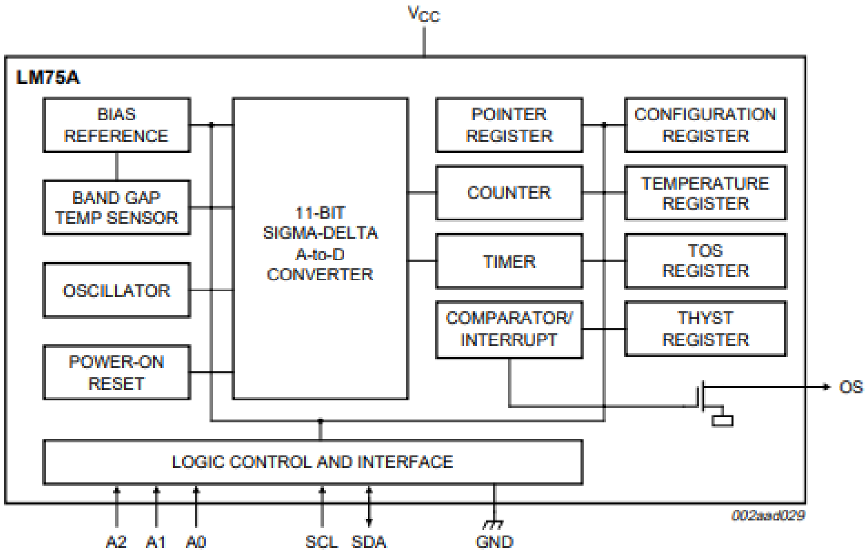


Figure 6. LM75A Block Diagram

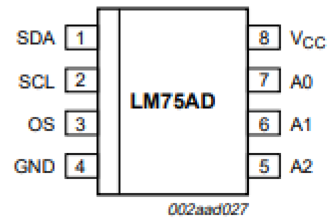


Figure 5. LM75A Pin Diagram

The LM75A is an industry-standard digital temperature sensor with an integrated sigma-delta ADC and I 2C interface. The LM75A provides 9-bit digital temperature readings with an accuracy of ±2°C from –25°C to 100°C and ±3°C over –55°C to 125°C. The LM75A operates with a single supply from +2.7 V to +5.5 V. Communication is accomplished over a 2-wire interface which operates up to 400kHz. The LM75A has three address pins, allowing up to eight LM75A devices to operate on the same 2-wire bus. The LM75A has a dedicated over-temperature output (O.S.) with programmable limit and hysteresis. This output has programmable fault tolerance, which allows the user to define the number of consecutive error conditions that must occur before O.S. is activated.

AD8232 BOARD SPECIFICATION

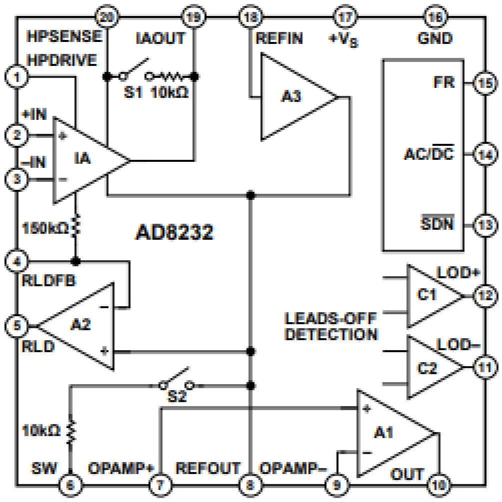


Figure 7. AD8232 Functional Block Diagram

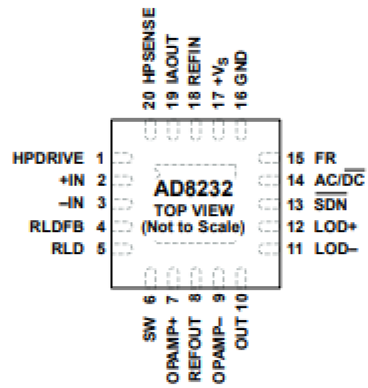


Figure 8. AD8232 Pin Diagram

Pin No.	Mnemonic	Description
1	HPDRIVE	High-Pass Driver Output. Connect HPDRIVE to the capacitor in the first high-pass filter. The AD8232 drives this pin to keep HPSENSE at the same level as the reference voltage.
2	+IN	Instrumentation Amplifier Positive Input. +IN is typically connected to the left arm (LA) electrode.
3	−IN	Instrumentation Amplifier Negative Input. −IN is typically connected to the right arm (RA) electrode.
4	RLDFB	Right Leg Drive Feedback Input. RLDFB is the feedback terminal for the right leg drive circuit.
5	RLD	Right Leg Drive Output. Connect the driven electrode (typically, right leg) to the RLD pin.
6	SW	Fast Restore Switch Terminal. Connect this terminal to the output of the second high-pass filter.
7	OPAMP+	Operational Amplifier Noninverting Input.
8	REFOUT	Reference Buffer Output. The instrumentation amplifier output is referenced to this potential. Use REFOUT as a virtual ground for any point in the circuit that needs a signal reference.
9	OPAMP−	Operational Amplifier Inverting Input.
10	OUT	Operational Amplifier Output. The fully conditioned heart rate signal is present at this output. OUT can be connected to the input of an ADC.
11	LOD−	Leads Off Comparator Output. In dc leads off detection mode, LOD− is high when the electrode to −IN is disconnected, and it is low when connected. In ac leads off detection mode, LOD− is always low.
12	LOD+	Leads Off Comparator Output. In dc leads off detection mode, LOD+ is high when the +IN electrode is disconnected, and it is low when connected. In ac leads off detection mode, LOD+ is high when either the −IN or +IN electrode is disconnected, and it is low when both electrodes are connected.
13	$\overline{\text{SDN}}$	Shutdown Control Input. Drive $\overline{\text{SDN}}$ low to enter the low power shutdown mode.
14	AC/$\overline{\text{DC}}$	Leads Off Mode Control Input. Drive the AC/$\overline{\text{DC}}$ pin low for dc leads off mode. Drive the AC/$\overline{\text{DC}}$ pin high for ac leads off mode.
15	FR	Fast Restore Control Input. Drive FR high to enable fast recovery mode; otherwise, drive it low.
16	GND	Power Supply Ground.
17	$+V_S$	Power Supply Terminal.
18	REFIN	Reference Buffer Input. Use REFIN, a high impedance input terminal, to set the level of the reference buffer.
19	IAOUT	Instrumentation Amplifier Output Terminal.
20	HPSENSE	High-Pass Sense Input for Instrumentation Amplifier. Connect HPSENSE to the junction of R and C that sets the corner frequency of the dc blocking circuit.
	EP	Exposed Pad. Connect the exposed pad to GND or leave it unconnected.

NODEMCU BOARD SPECIFICATION

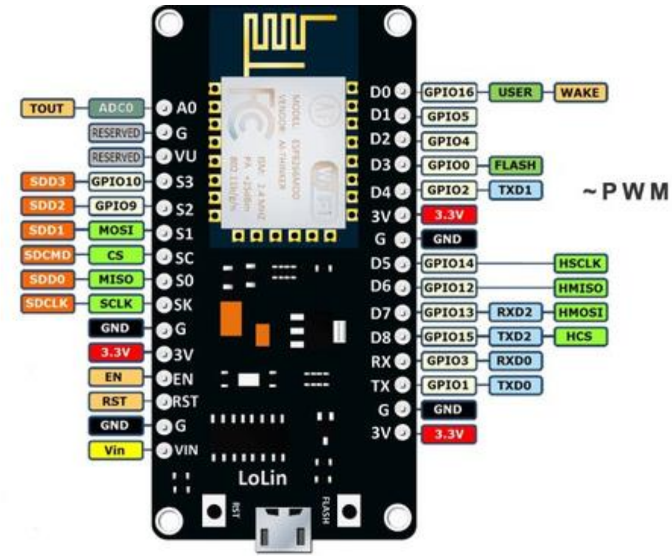
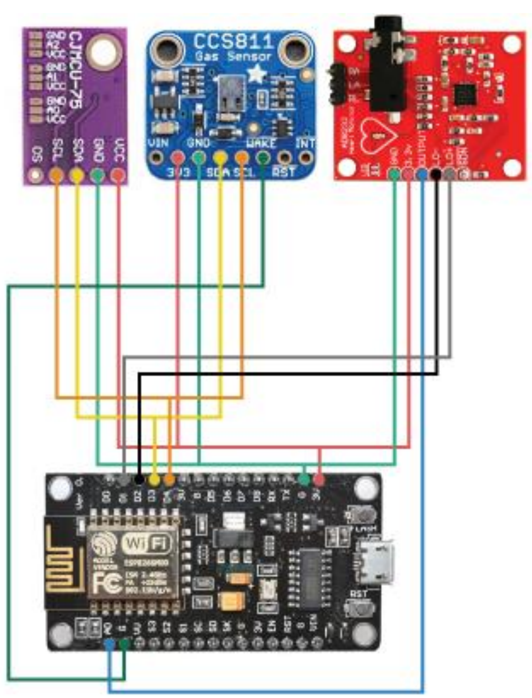


Figure 10. NodeMCU V 0.9 Pin Layout

SCHEMATIC AND PIN ROUTING



PIN Routes	
Sensors	NodeMCU
VCC / 3.3V	3.3V
GND / WAK	GND
SDA	D3
SCL	D4
OUTPUT	A0
LO+	D1
LO-	D2

Figure 9. Schematic of Sensors to NodeMCU

CODE SNIPPET

```
#include <SoftwareSerial.h>
#include <LM75A.h>
// Create I2C LM75A instance
LM75A lm75a_sensor(false,  //A0 LM75A pin state
                   false,  //A1 LM75A pin state
                   false); //A2 LM75A pin state

// Setup air quality sensor to pin (D1, D2) --> (SDA, SCL)
#include "Adafruit_CCS811.h"
Adafruit_CCS811 ccs;

#include <ESP8266WiFi.h>;
#include <WiFiClient.h>;
#include <ThingSpeak.h>;
const char* ssid = "..."; // Replace the ... with your Network SSID
const char* password = "..."; // Replace the ... your Network Password

// Initiate variables for updating to Thingspeak
float temperature_in_degrees;
float temp;
int ecg;
int co2;
int TVOC;



WiFiClient client;
unsigned long myChannelNumber = [...]; // Replace the ... with your Channel Number (Without Brackets)
const char * myWriteAPIKey = "..."; // Replace the ... with your Write API key

void setup() {
  // Enable Serial Monitor, NodeMCU runs on 115600 by default
  Serial.begin(9600);
  delay(10);

  // Connect to WiFi network
  WiFi.begin(ssid, password);
  ThingSpeak.begin(client);

  // Setting up the air quality sensor
  Serial.println("Starting Up CCS811");
  if (!ccs.begin()) {
    Serial.println("Failed to start sensor! Please check your wiring.");
    while (1);



  // Calibrate the air temp. sensor
  while (!ccs.available());
  float temp = ccs.calculateTemperature();
  ccs.setTempOffset(temp - 25.0);

  //  Set digital pins to input for ECG sensor
  pinMode(D3, INPUT); // Setup for leads off detection LO +
  pinMode(D4, INPUT); // Setup for leads off detection LO -
}

void loop() {
  //  Run functions for LM75A sensor
  temperature_in_degrees = lm75a_sensor.getTemperatureInDegrees();
  if (temperature_in_degrees == INVALID_LM75A_TEMPERATURE) {
    Serial.println("Error while getting temperature");
  } else {
    Serial.println("");
    Serial.print("Body Temperature: ");
    Serial.print(temperature_in_degrees);
    Serial.println(" C°");
  }
```

```
  //  Run functions for CSS811 sensor
  if (ccs.available()) {
    temp = ccs.calculateTemperature();
    co2 = ccs.geteCO2();
    TVOC = ccs.getTVOC();
    if (!ccs.readData()) {
      Serial.print("CO2: ");
      Serial.print(co2);
      Serial.println(" ppm");
      Serial.print("TVOC (Total Volatile Organic Compound): ");
      Serial.print(TVOC);
      Serial.println(" ppb");
      Serial.print("Air Temperature: ");
      Serial.print(temp);
      Serial.println(" C°");
    }
    else {
      Serial.println("ERROR!");
      while (1);
    }
  }

  //  Run functions for AD8232 sensor
  if ((digitalRead(D3) == 1) || (digitalRead(D4) == 1)) {
    Serial.println('!');
  }
  else {
    ecg = analogRead(A0);
    Serial.println(ecg);
  }
  delay(10);

  //  Set fields through Thingspeak library, to enable multiple data transmission
  ThingSpeak.setField(1,temperature_in_degrees);
  ThingSpeak.setField(2,co2);
  ThingSpeak.setField(3,TVOC);
  ThingSpeak.setField(4,temp);
  ThingSpeak.setField(5,ecg);

  // Update all the fields to Thingspeak
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

  // Delay to prevent saturation of data
  delay(100);

}
```

UI/UX VIA THINGSPEAK

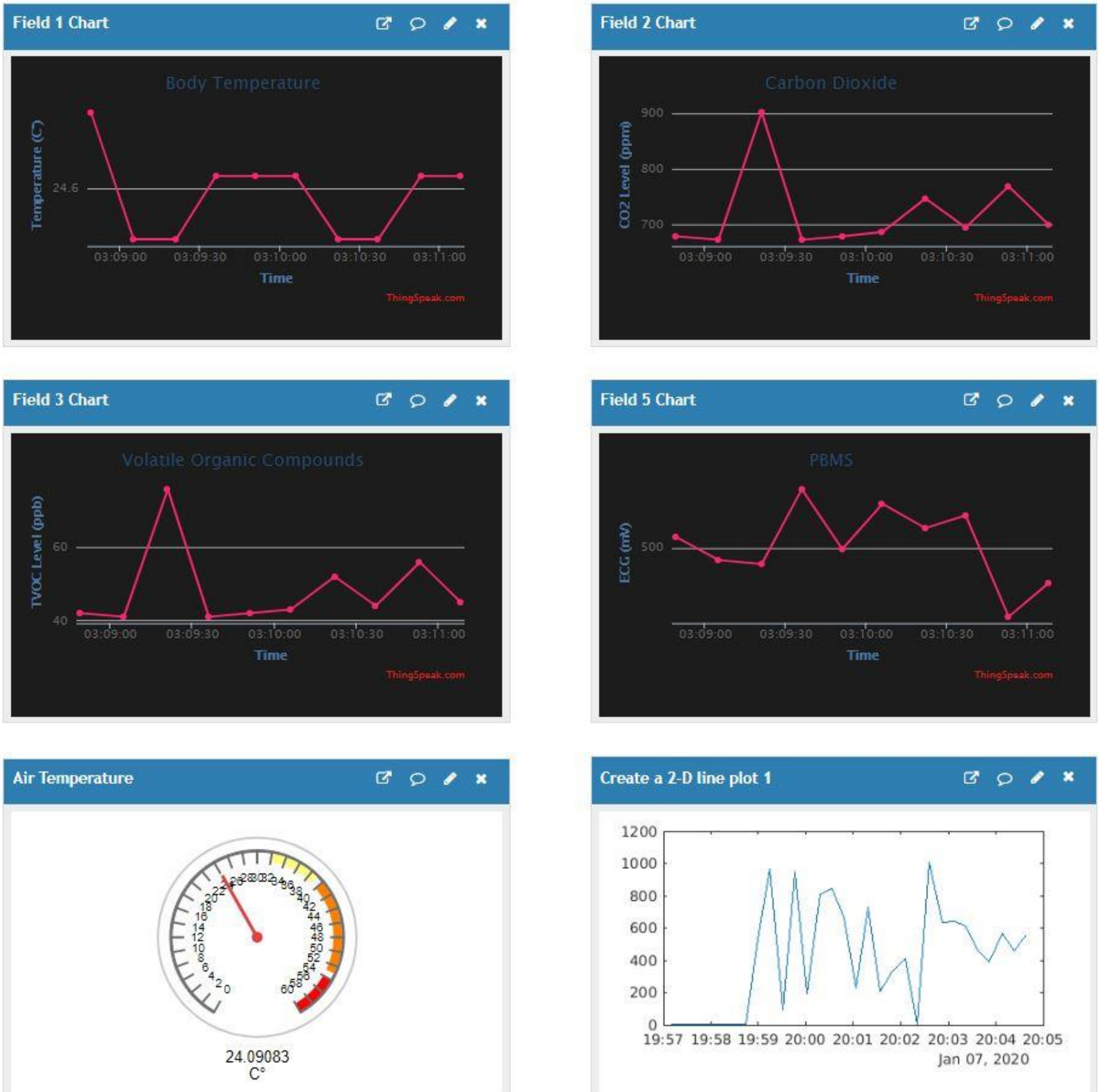


Figure 11. Channel View on Dashboard (15 Seconds per Update)

PROTOTYPE DOCUMENTATION

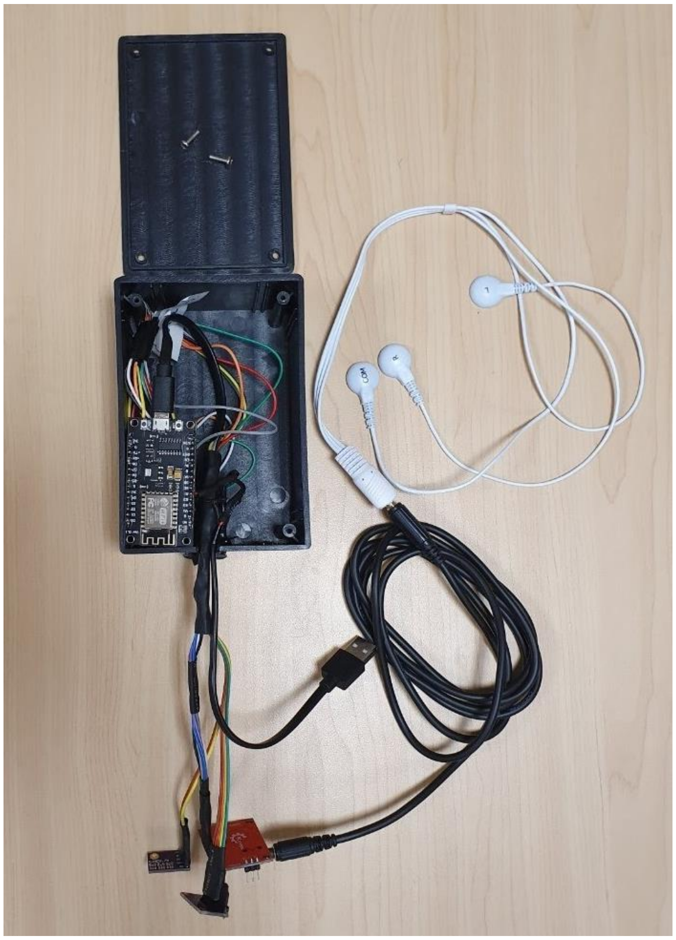


Figure 12. Prototype Overview

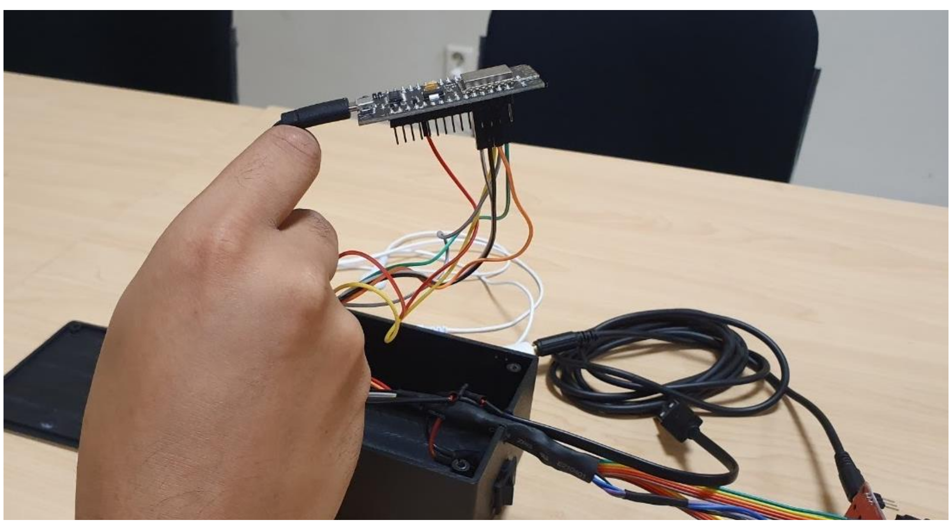


Figure 13. NodeMCU Wiring

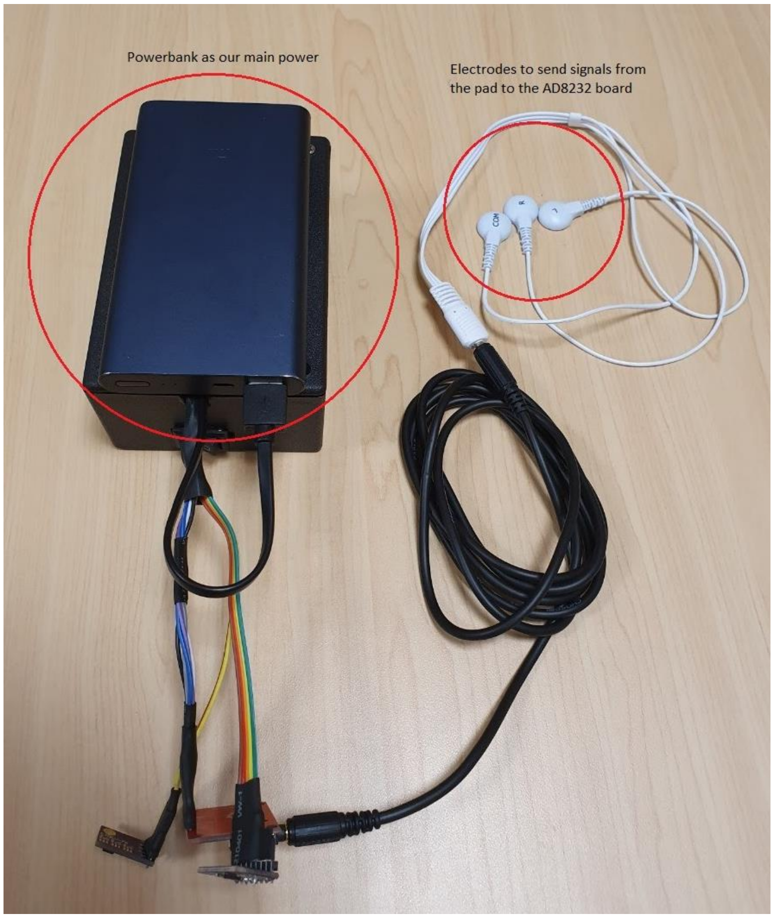


Figure 14. Closed Prototype Overview

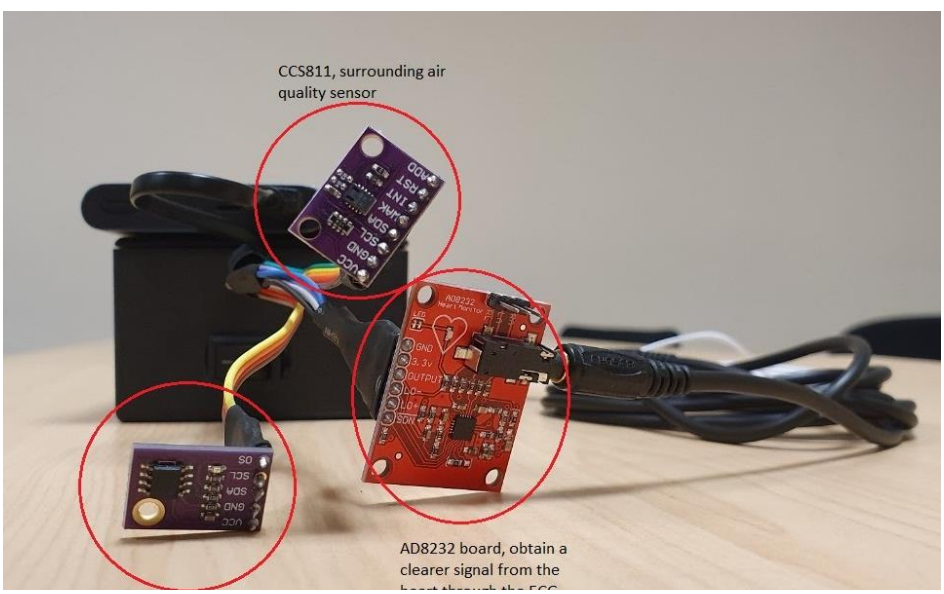


Figure 15. Sensors Overview

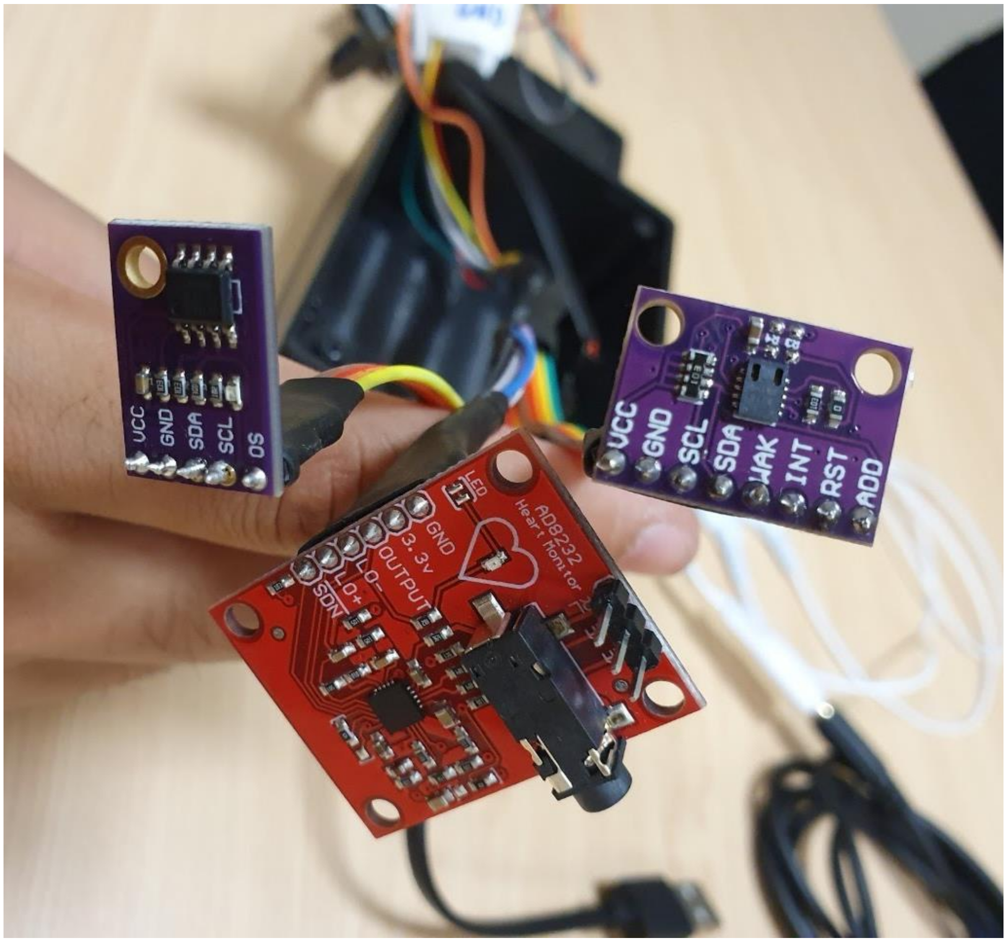


Figure 16. Sensors Overview