|  |
| --- |
|  |

|  |
| --- |
| **SMART HEALTH MONITORING** |
| ~IOT |
|  |
| Importance of using the IOT in day to day life |
|  |
| **BEGINNERS** |
| **5/31/2019** |
|  |

*PURPOSE OF USING THIS DEVICE*

* *AFFORDABLE*
* *NO NEED OF CARETAKERS*
* *EASE OF APPLICATION*
* *ACCURATE*
* *DETAILS CAN BE RETRIEVED AT ANY TIME*
* *GIVE ALERT BEFORE CRITICAL SITUATION*

*REQUIREMENTS:*

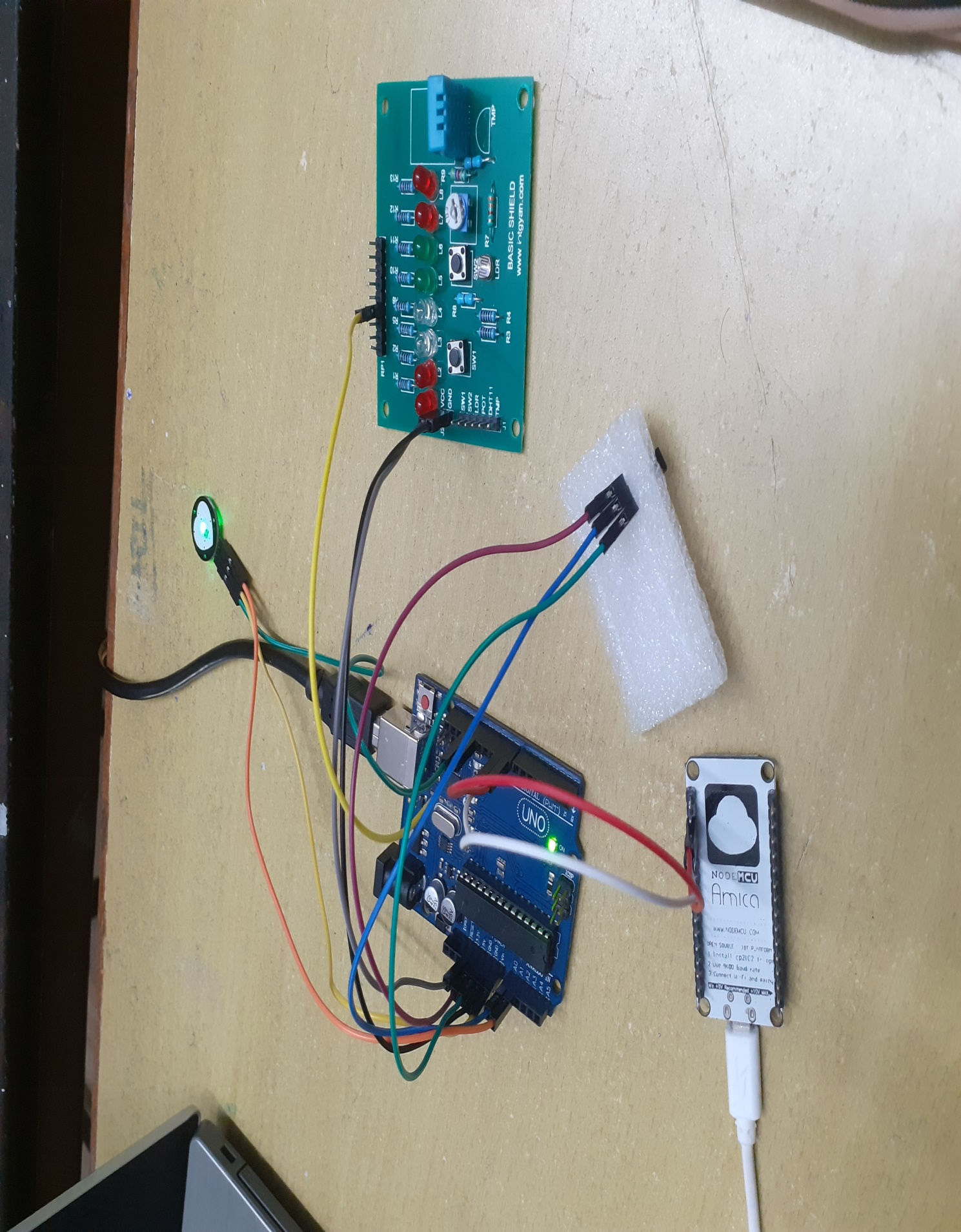
***HARDWARE:***

* *Node MCU*
* *Pulse sensor*
* *Temperature sensor (LM 35)*
* *Arduino UNO*
* *Basic shield*

***SOFTWARE:***

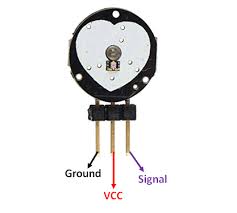
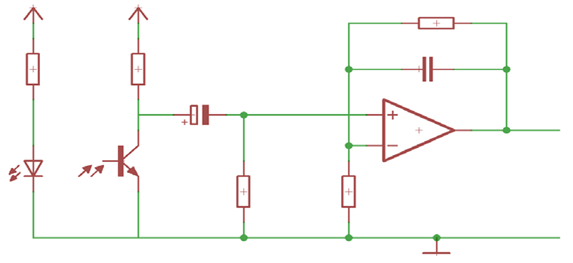
* *Arduino IDE*
* *IBM Watson Cloud platform*
* *MIT App Inventor*

*CONNECTIVITY:*

**

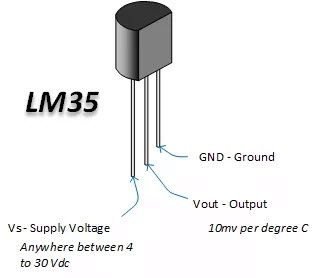
*Principle of Heartbeat Sensor:*

* *The heartbeat sensor is based on the principle of photo phlethysmography. It measures the change in volume of blood through any organ of the body which causes a change in the light intensity through that organ (a vascular region). In case of applications where heart*[*pulse rate is to be monitored*](https://www.edgefxkits.com/patient-health-monitoring-system-with-location-details-by-gps-over-gsm)*, the timing of the pulses is more important. The flow of blood volume is decided by the rate of heart pulses and since light is absorbed by blood, the signal pulses are equivalent to the heart beat pulses.*
* *The basic heartbeat sensor consists of a light emitting diode and a detector like a light detecting resistor or a photodiode. The heart beat pulses causes a variation in the flow of blood to different regions of the body. When a tissue is illuminated with the light source, i.e. light emitted by the led, it either reflects (a finger tissue) or transmits the light (earlobe). Some of the light is absorbed by the blood and the transmitted or the reflected light is received by the light detector. The amount of light absorbed depends on the blood volume in that tissue. The detector output is in form of electrical signal and is proportional to the heart beat rate.*
* *This signal is actually a DC signal relating to the tissues and the blood volume and the AC component synchronous with the heart beat and caused by pulsatile changes in arterial blood volume is superimposed on the DC signal. Thus the major requirement is to isolate that AC component as it is of prime importance.*
* ***To achieve the task of getting the AC signal, the output from the detector is first filtered using a 2 stage HP-LP circuit and is then converted to digital pulses using a comparator circuit or using simple ADC. The digital pulses are given to a microcontroller for calculating the heat beat rate, given by the formula-***
* ***BPM(Beats per minute) = 60\*f***
* ***Where f is the pulse frequency***

***[](https://www.elprocus.com/wp-content/uploads/2013/08/Heartbeat.png)***

*WORKING PRINCIPLE OF THERMAL SENSOR(LM35)*

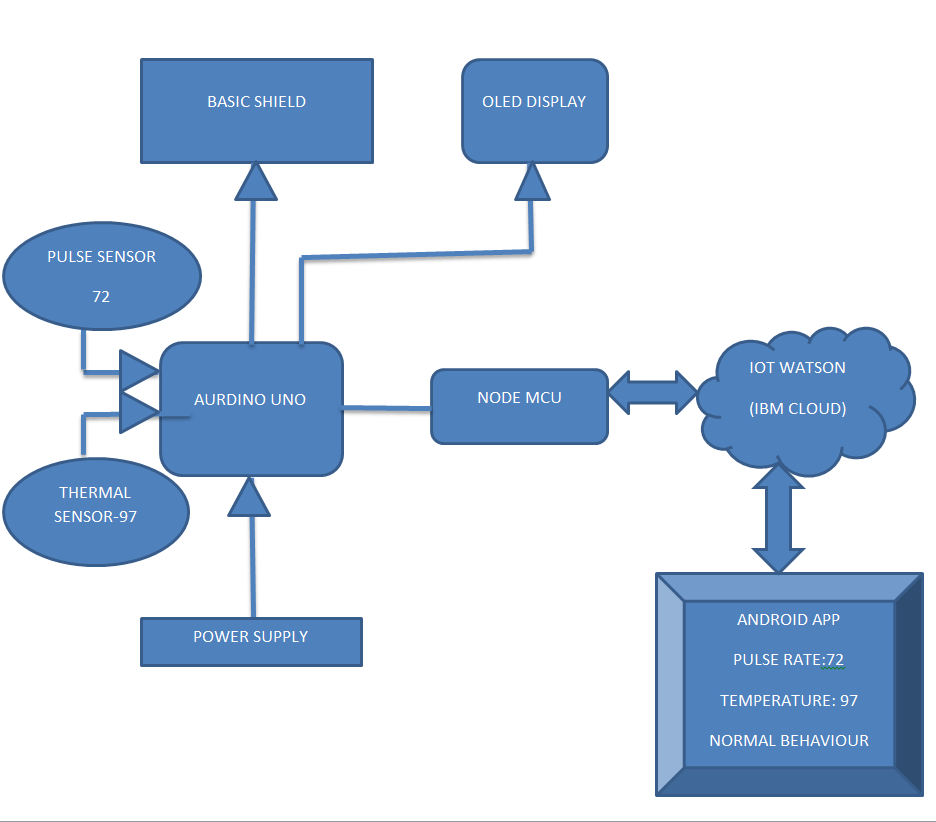
* The power supply of the LM35 temperature sensor requires 5.5V and it consists of three terminals of a material which perform the operation according to the temperature to vary resistance. When the voltage increases, then the temperature also rises. We can see this operation by using a diode.
* Temperature sensors directly connected to microprocessor input, therefore it has capable of direct and reliable communication with microprocessors. The sensor unit can communicate effectively with low-cost processors without the need of [A/D converters](https://www.efxkits.co.uk/fastest-18-bit-sar-analog-to-digital-converter/). The features of LM35 temperature sensor are explained below.



#### Features of LM35 Temperature Sensor :

* Calibrated directly in ˚ Celsius (Centigrade)
* Rated for full l −55˚ to +150˚C range
* Suitable for remote applications
* Low cost due to wafer-level trimming
* Operates from 4 to 30 volts
* Low self-heating,
* ±1/4˚C of typical nonlinearity

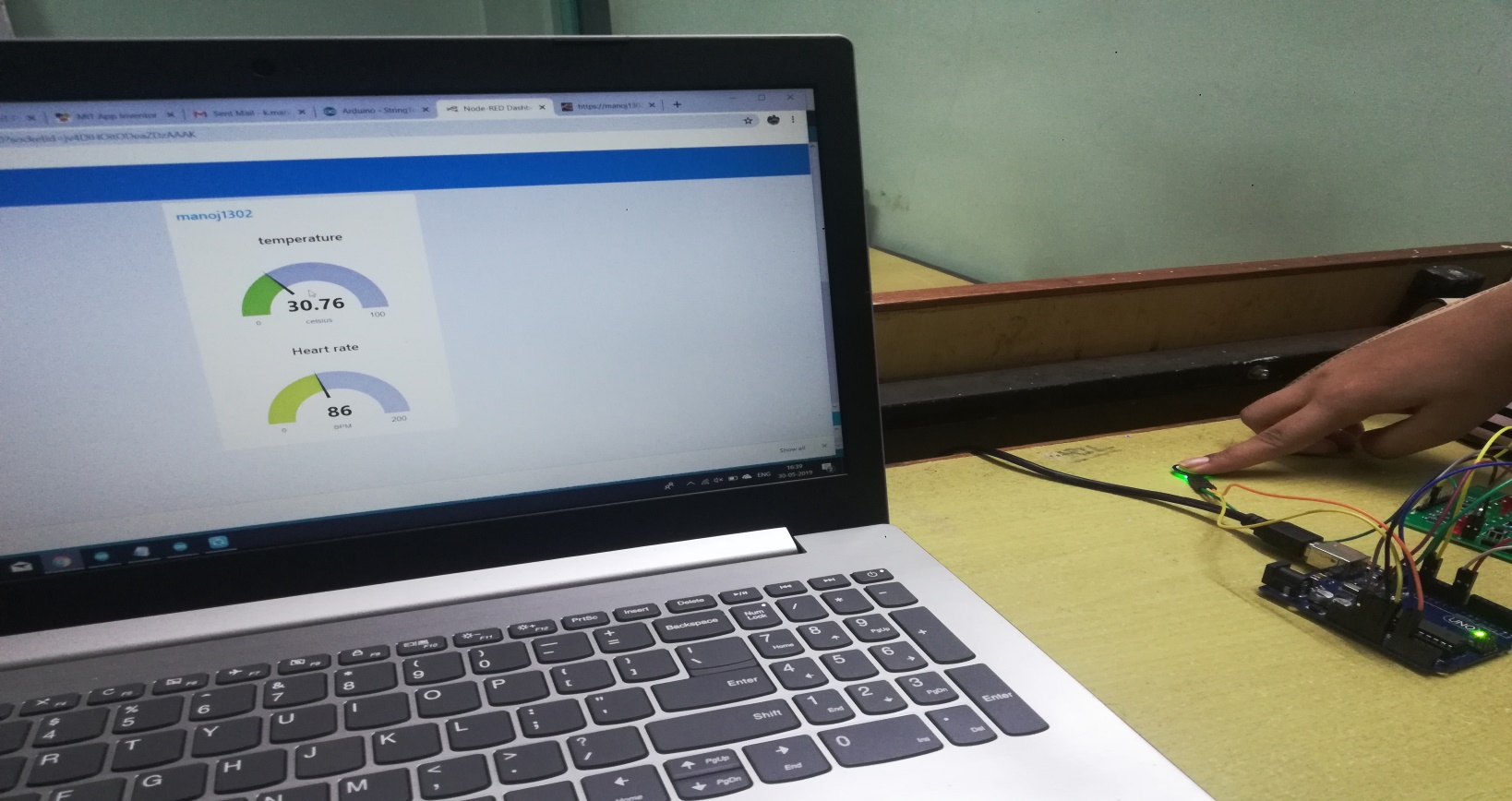
*BLOCK DIAGRAM :*

**

# Project Description :

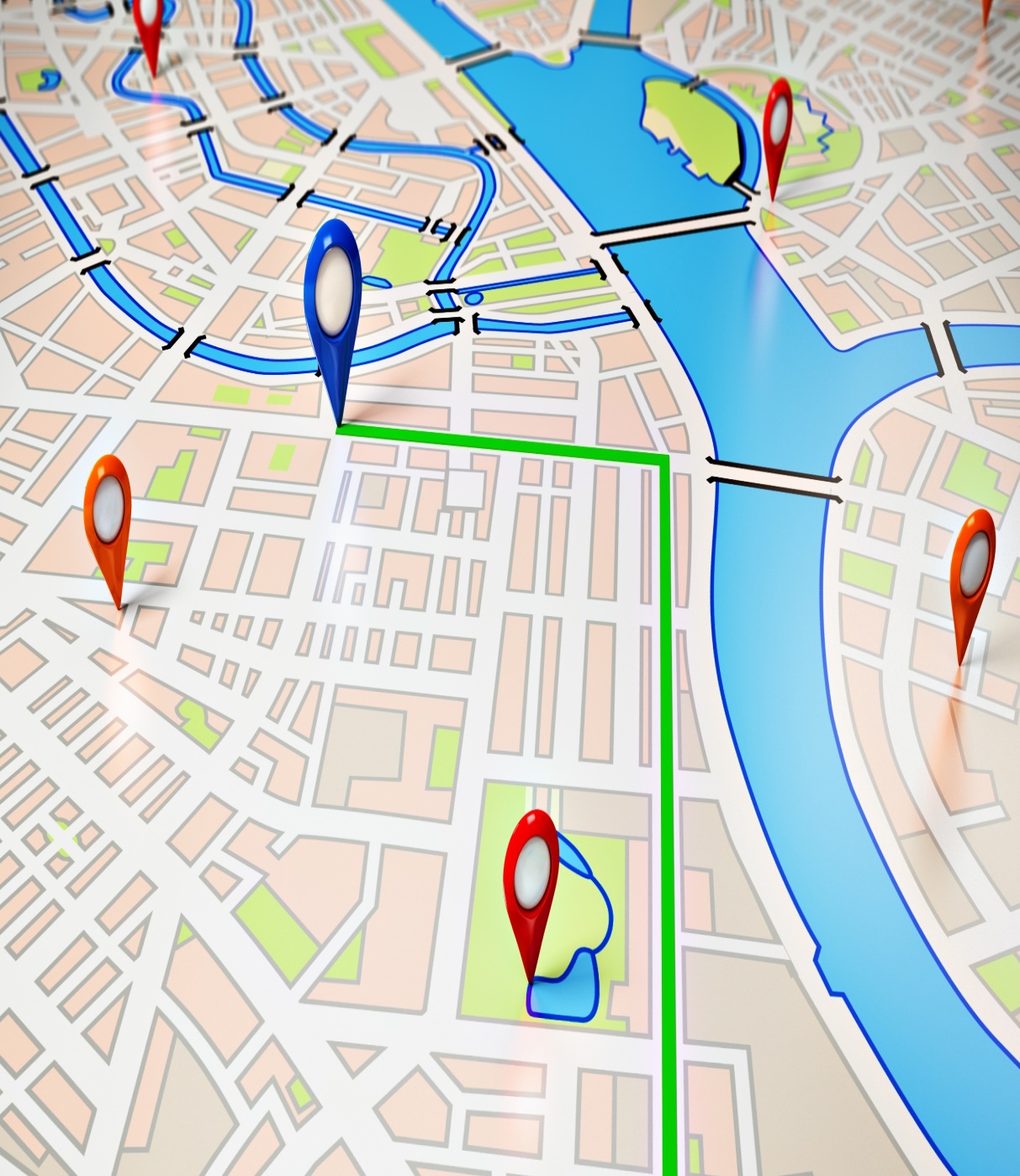
* Keeping track of the health status of the patient at home is a difficult. Especially old age patients should be periodically monitored and their loved ones need to be informed about their health status from time to time while at work. So we propose an innovative system that automated this task with ease. Our system puts forward a smart patient health tracking system that uses Sensors to track patient health and alerts the care takers in case of emergencies.
* By using sensors we can check the health parameters of the elders and these sensor values are sent to IBM Watson cloud and those values are stored in the database. These values are then visualized on the user interface which is created using node red of IBM Watson services. These values are also sent to mobile application and whenever there is case of emergency notifications are sent to desired persons.

***READING THE VALUES FROM THE DEVICE :***



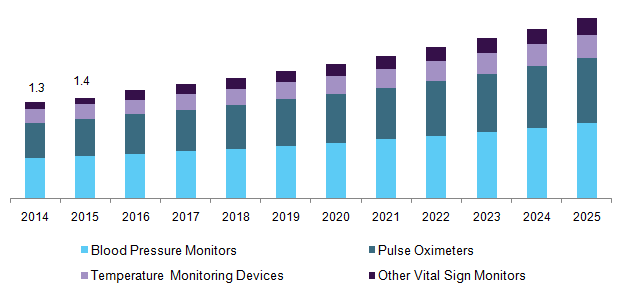
# Project Highlights :

* Working with IBM Watson cloud services
* Accessing Sensor data from anywhere in the world.
* Notifications and SMS are triggered based on sensor values automatically.
* Reduces the need for a caretaker to be assigned to each patient.
* Even the Family members can monitor the health status of their loved ones from anywhere in the world.





***GRAPHICAL REPRESENTATION :***



appendix :

CODE (nodemcu):

#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

//-------- Customise these values -----------

const char\* ssid = "YOU1";

const char\* password = "sbvizag20191";

SoftwareSerial m(D2,D3);

String url2;

#define ORG "5rxjjv"

#define DEVICE\_TYPE "IOT"

#define DEVICE\_ID "IOT0203"

#define TOKEN "17B91A0598"

//-------- Customise the above values --------

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";

char publishTopic[] = "iot-2/evt/Data/fmt/json";

char authMethod[] = "use-token-auth";

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE\_TYPE ":" DEVICE\_ID;

char topic[] = "iot-2/cmd/home/fmt/String";

WiFiClient wifiClient;

PubSubClient client(server, 1883,wifiClient);

void setup() {

Serial.begin(9600);

// put your setup code here, to run once:

m.begin(9600);

mqttConnect();

}

void loop() {

// put your main code here, to run repeatedly:

if (m.find("#"))

{

url2=m.readStringUntil('~');

Serial.println(url2);

//m.toInt(url2);

//Serial.println(url2.toInt());

delay(2000);

Serial.print("\n");

Serial.print("Sending payload: "); Serial.println(url2);

if (client.publish(publishTopic, (char\*) url2.c\_str())) {

Serial.println("Publish OK");

} else {

Serial.println("Publish FAILED");

}

}

else{

Serial.println("no data from arduino1");

}

}

void wifiConnect() {

Serial.print("Connecting to "); Serial.print(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.print("nWiFi connected, IP address: "); Serial.println(WiFi.localIP());

}

void initManagedDevice() {

if (client.subscribe(topic)) {

Serial.println("subscribe to cmd OK");

} else {

Serial.println("subscribe to cmd FAILED");

}

}

void mqttConnect() {

if (!client.connected()) {

Serial.print("Reconnecting MQTT client to "); Serial.println(server);

while (!client.connect(clientId, authMethod, token)) {

Serial.print(".");

delay(500);

}

initManagedDevice();

Serial.println();

}

}

CODE (UNO):

/\*

>> Pulse Sensor Amped 1.2 << >>> Pulse Sensor purple wire goes to Analog Pin 0 <<<

Pulse Sensor sample aquisition and processing happens in the background via Timer 2 interrupt. 2mS sample rate.

PWM on pins 3 and 11 will not work when using this code, because we are using Timer 2!

The following variables are automatically updated:

Signal : int that holds the analog signal data straight from the sensor. updated every 2mS.

IBI : int that holds the time interval between beats. 2mS resolution.

BPM : int that holds the heart rate value, derived every beat, from averaging previous 10 IBI values.

QS : boolean that is made true whenever Pulse is found and BPM is updated. User must reset.

Pulse : boolean that is true when a heartbeat is sensed then false in time with pin13 LED going out.

\*/

#include <SoftwareSerial.h>

SoftwareSerial mySerial(9,8);

// VARIABLES

int pulsePin = A1; // Pulse Sensor purple wire connected to analog pin 0

int tempPin = A0; // Temperature Sensor connected to analog pin 1

int blinkPin = 13; // pin to blink led at each beat

float sensorValue1;

int sensorValue[2]={0,0};

int voltageValue[2] = {0, 0};

char inbyte = 0;

// these variables are volatile because they are used during the interrupt service routine!

volatile int BPM; // used to hold the pulse rate

volatile int Signal; // holds the incoming raw data

volatile int IBI = 600; // holds the time between beats, must be seeded!

volatile boolean Pulse = false; // true when pulse wave is high, false when it's low

volatile boolean QS = false; // becomes true when Arduoino finds a beat.

void setup(){

pinMode(blinkPin,OUTPUT); // pin that will blink to your heartbeat!

Serial.begin(9600); // we agree to talk fast!

mySerial.begin(9600); //

interruptSetup(); // sets up to read Pulse Sensor signal every 2mS

// UN-COMMENT THE NEXT LINE IF YOU ARE POWERING The Pulse Sensor AT LOW VOLTAGE,

// AND APPLY THAT VOLTAGE TO THE A-REF PIN

//analogReference(EXTERNAL);

}

void loop(){

getSensorValues();

getVoltageValue();

sendAndroidValues();

delay(20); // take a break

}

void getSensorValues()

{

// read the analog in value to the sensor array

sensorValue[0] = ( 5 \* analogRead(tempPin) \* 100.0) / 1024.0;

Serial.println(sensorValue[0]);

sensorValue1=(5\*analogRead(tempPin)\*100.0)/1024;

Serial.println(sensorValue1);

if (QS == true){ // Quantified Self flag is true when arduino finds a heartbeat

Serial.print("Signal=");

Serial.println(Signal); // Print pulse sensor raw data

Serial.print("HeartBeat=");

Serial.println(BPM); // Print BPM (beats per Minut)

Serial.print("Time Interval=");

Serial.println(IBI); // Print IBI (Interval between beats)

sensorValue[1] = BPM; // read the BPM to the sensor array

QS = false; // reset the Quantified Self flag for next time

}

}

void sendAndroidValues()

{

//puts # before the values so our app knows what to do with the data

mySerial.print('#');

Serial.print('#');

//for loop cycles through 2 sensors and sends values via serial

String payload ="{\"d\":{\"temperature\":";

payload +=sensorValue1;

payload+=",""\"signal\":";

payload+=BPM;

payload+="}}";

Serial.println(payload);

mySerial.println(payload);

mySerial.println('~'); //used as an end of transmission character - used in app for string length

Serial.println('~');

delay(2000); //added a delay to eliminate missed transmissions

}

void getVoltageValue()

{

if (BPM>=60 && BPM<=100){

for (int x = 0; x < 2; x++)

{

voltageValue[x] = sensorValue[x];

}

}

else {

voltageValue[1] = 0;

}

}

INTERRUPT FUNCTION :

volatile int rate[10]; // array to hold last ten IBI values

volatile unsigned long sampleCounter = 0; // used to determine pulse timing

volatile unsigned long lastBeatTime = 0; // used to find IBI

volatile int P =512; // used to find peak in pulse wave, seeded

volatile int T = 512; // used to find trough in pulse wave, seeded

volatile int thresh = 512; // used to find instant moment of heart beat, seeded

volatile int amp = 100; // used to hold amplitude of pulse waveform, seeded

volatile boolean firstBeat = true; // used to seed rate array so we startup with reasonable BPM

volatile boolean secondBeat = false; // used to seed rate array so we startup with reasonable BPM

void interruptSetup(){

// Initializes Timer2 to throw an interrupt every 2mS.

TCCR2A = 0x02; // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE

TCCR2B = 0x06; // DON'T FORCE COMPARE, 256 PRESCALER

OCR2A = 0X7C; // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE

TIMSK2 = 0x02; // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A

sei(); // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED

}

// THIS IS THE TIMER 2 INTERRUPT SERVICE ROUTINE.

// Timer 2 makes sure that we take a reading every 2 miliseconds

ISR(TIMER2\_COMPA\_vect){ // triggered when Timer2 counts to 124

cli(); // disable interrupts while we do this

Signal = analogRead(pulsePin); // read the Pulse Sensor

sampleCounter += 2; // keep track of the time in mS with this variable

int N = sampleCounter - lastBeatTime; // monitor the time since the last beat to avoid noise

// find the peak and trough of the pulse wave

if(Signal < thresh && N > (IBI/5)\*3){ // avoid dichrotic noise by waiting 3/5 of last IBI

if (Signal < T){ // T is the trough

T = Signal; // keep track of lowest point in pulse wave

}

}

if(Signal > thresh && Signal > P){ // thresh condition helps avoid noise

P = Signal; // P is the peak

} // keep track of highest point in pulse wave

// NOW IT'S TIME TO LOOK FOR THE HEART BEAT

// signal surges up in value every time there is a pulse

if (N > 250){ // avoid high frequency noise

if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)\*3) ){

Pulse = true; // set the Pulse flag when we think there is a pulse

digitalWrite(blinkPin,HIGH); // turn on pin 13 LED

IBI = sampleCounter - lastBeatTime; // measure time between beats in mS

lastBeatTime = sampleCounter; // keep track of time for next pulse

if(secondBeat){ // if this is the second beat, if secondBeat == TRUE

secondBeat = false; // clear secondBeat flag

for(int i=0; i<=9; i++){ // seed the running total to get a realisitic BPM at startup

rate[i] = IBI;

}

}

if(firstBeat){ // if it's the first time we found a beat, if firstBeat == TRUE

firstBeat = false; // clear firstBeat flag

secondBeat = true; // set the second beat flag

sei(); // enable interrupts again

return; // IBI value is unreliable so discard it

}

// keep a running total of the last 10 IBI values

word runningTotal = 0; // clear the runningTotal variable

for(int i=0; i<=8; i++){ // shift data in the rate array

rate[i] = rate[i+1]; // and drop the oldest IBI value

runningTotal += rate[i]; // add up the 9 oldest IBI values

}

rate[9] = IBI; // add the latest IBI to the rate array

runningTotal += rate[9]; // add the latest IBI to runningTotal

runningTotal /= 10; // average the last 10 IBI values

BPM = 60000/runningTotal; // how many beats can fit into a minute? that's BPM!

QS = true; // set Quantified Self flag

// QS FLAG IS NOT CLEARED INSIDE THIS ISR

}

}

if (Signal < thresh && Pulse == true){ // when the values are going down, the beat is over

digitalWrite(blinkPin,LOW); // turn off pin 13 LED

//Serial.println(BPM);

Pulse = false; // reset the Pulse flag so we can do it again

amp = P - T; // get amplitude of the pulse wave

thresh = amp/2 + T; // set thresh at 50% of the amplitude

P = thresh; // reset these for next time

T = thresh;

}

if (N > 2500){ // if 2.5 seconds go by without a beat

thresh = 512; // set thresh default

P = 512; // set P default

T = 512; // set T default

lastBeatTime = sampleCounter; // bring the lastBeatTime up to date

firstBeat = true; // set these to avoid noise

secondBeat = false; // when we get the heartbeat back

}

sei(); // enable interrupts when youre done!

}// end isr

Limitations:

* Compatibility
* complexity