IoT Based Patient Health Monitoring System using ESP8266 & Arduino

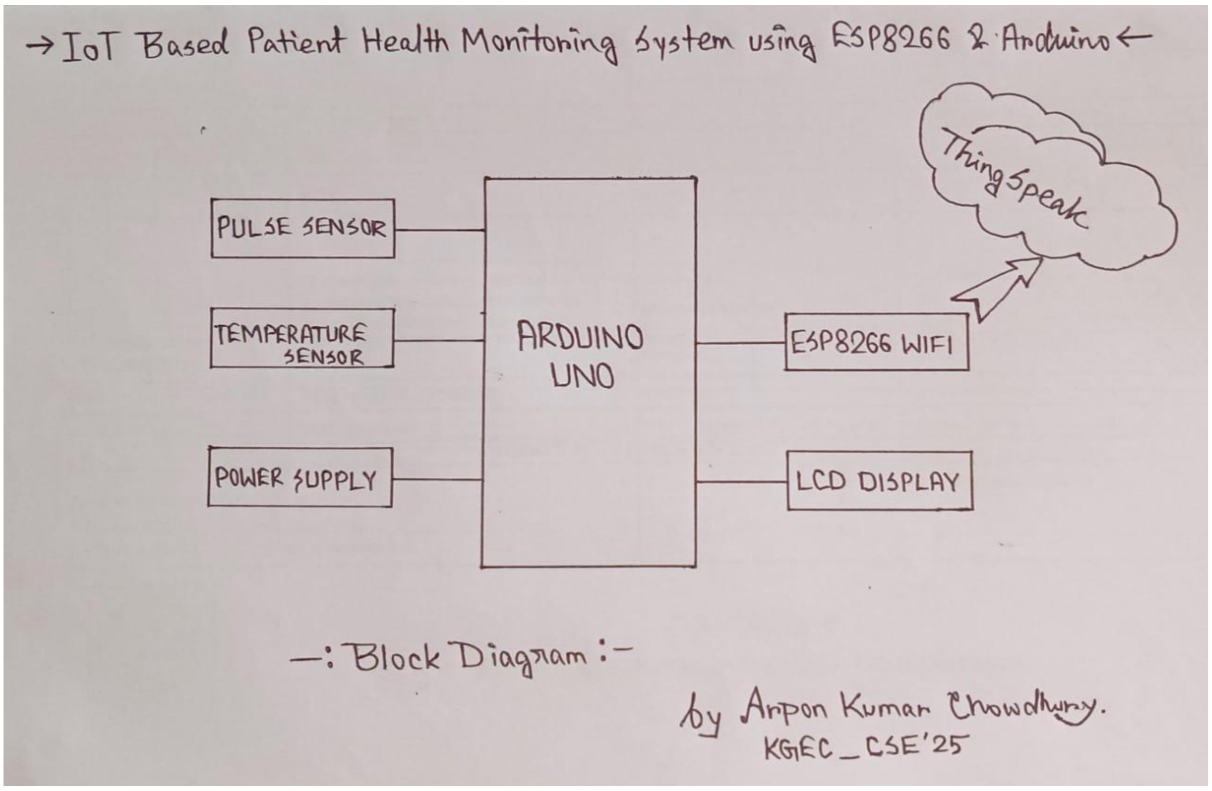
Introduction:

IoT is rapidly revolutionising the Healthcare Industry. In this project, I have designed the IoT Based Patient Health Monitoring System using ESP8266 & Arduino. The IoT platform used in this project is ThingSpeak. This IoT device could read the pulse rate and surrounding temperature and update them to an IoT platform.
This Health Monitoring System simulates two major aspects of health and wellbeing :
i) Human Body Temperature, ii) Heart Beat Rate.
It presents the numerical value of these two aspects and maintains a data logger system where the patient's data can be stored and received for medical history and check-up purpose collectively. Along with that the output values can be viewed through waveform chart. The circuit output values in the LED represent the current heartbeat, time in second, heartbeat per minute and body temperature.

Objectives:

- To design a portable health monitoring system, which measures the patient's body temperature and pulse rate.
- To provide medical assistance according to the data received from the sensors which stored in cloud server (ThingSpeak).

Block Diagram:



This is a simple block diagram that explains the IoT Based Patient Health Monitoring System using ESP8266 & Arduino. Pulse sensor and LM35 temperature sensor measure BPM & Environmental temperature respectively. The Arduino processes the code and displays it to 16*2 LCD Display. ESP8266 Wi-Fi module connects to Wi-Fi and sends the data to IoT device server. The IoT server used here is ThingSpeak. Finally, the data can be monitored from any part of the World by logging into the ThingSpeak channel.

Requirements:

Hardware Requirements:

- Arduino Uno
- ESP8266 Wi-Fi Module
- LM35 Temperature Sensor
- Pulse Sensor
- 16*2 LCD Display
- Potentiometer (10K)
- Resistor (1K & 2K)
- Breadboard

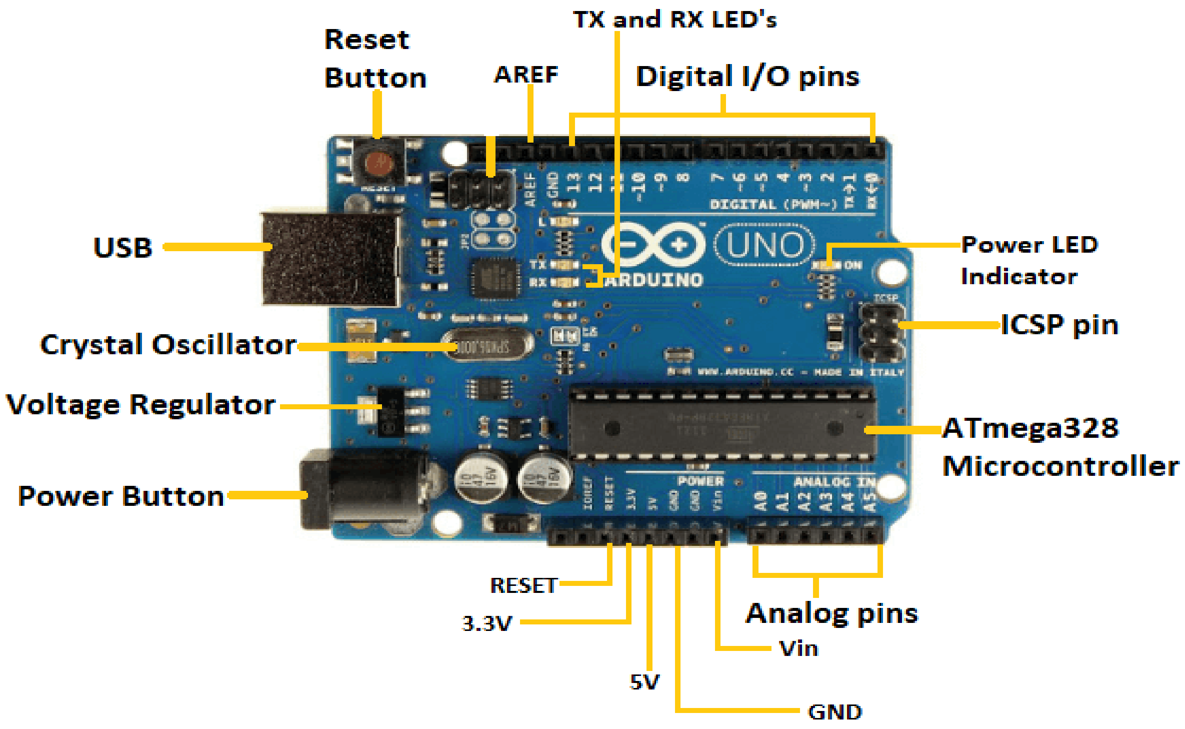
Software Requirements:

- Arduino Compiler
- ThingSpeak (Cloud Service)

Arduino Uno :

Arduino UNO is based on an ATmega328P microcontroller. The Arduino UNO includes 6 analog pin inputs, 14 digital pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. It is programmed based on IDE, which stands for Integrated Development Environment. It can run on both online and offline platforms.
The components of Arduino UNO board are shown below:

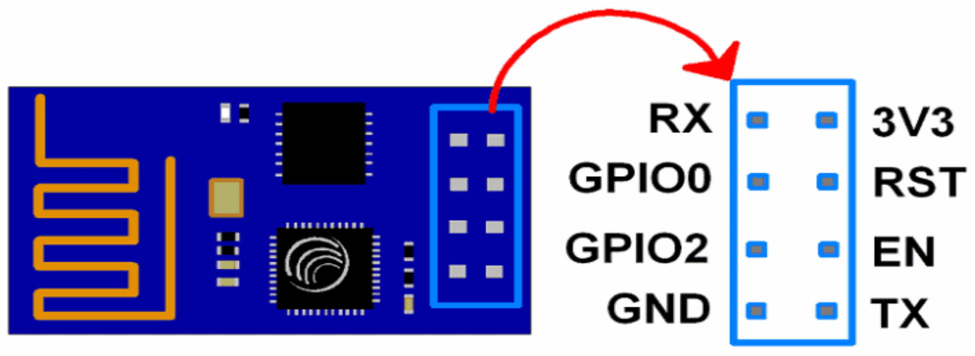
Let's discuss each component in detail :



- **ATmega328 Microcontroller** : It is a single chip Microcontroller of the ATmel family. The processor code inside it is 8-bit. It combines **Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timer, external and internal interrupts, and oscillator.**
- **ICSP pin** : The In-Circuit Serial Programming pin allows the user to program using the firmware of the Arduino board.
- **Power LED Indicator** : The ON status of LED shows the power is activated. When the power is OFF, the LED will not light up.
- **Digital I/O pins** : The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
- **TX and RX LED's** : The successful flow of data is represented by the lighting of these LED's.
- **AREF :** The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
- **Reset button** : It is used to add a Reset button to the connection.
- **USB** : It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
- **Crystal Oscillator** : The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.
- **Voltage Regulator** : The voltage regulator converts the input voltage to 5V.
- **GND** : Ground pins. The ground pin acts as a pin with zero voltage.
- **Vin** : It is the input voltage.
- **Analog Pins** : The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pins.

ESP8266 :

The ESP8266 is a very user-friendly and low-cost device to provide internet connectivity to our projects. The module can work both as an Access point (can create hotspot) and as a station (can connect to Wi-Fi), hence it can easily fetch data and upload it to the internet making the Internet of Things as easy as possible. It can also fetch data from the internet using API's hence our project could access any information that is available on the internet, thus making it smarter. Another exciting feature of this module is that it can be programmed using the Arduino IDE which makes it a lot more user-friendly.



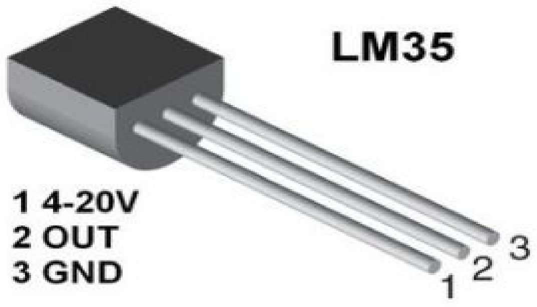
The ESP8266 module works with 3.3V only, anything more than 3.7V would kill the module hence be cautious with our circuits.

Here is its pins description.

Pin 1: Ground: Connected to the ground of the circuit
Pin 2: Tx/GPIO – 1: Connected to Rx pin of programmer/uC to upload program
Pin 3: GPIO – 2: General purpose Input/output pin
Pin 4 : CH_EN: Chip Enable/Active high
Pin 5: Flash/GPIO – 0: General purpose Input/output pin
Pin 6 : Reset: Resets the module
Pin 7: RX/GPIO – 3: General purpose Input/output pin
Pin 8: Vcc: Connect to +3.3V only

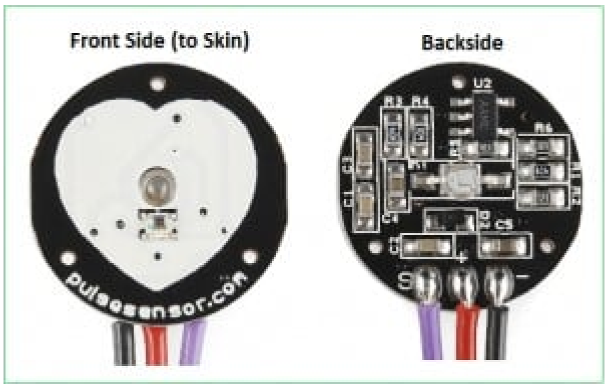
LM35 Temperature Sensor :

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly-proportional to the Centigrade temperature. The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range.

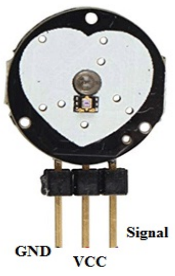


Pulse Sensor :

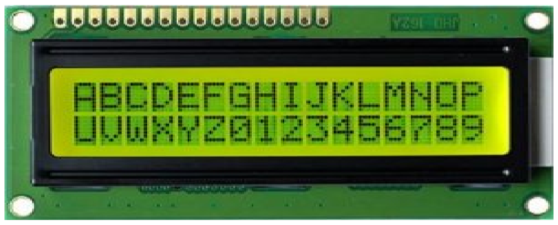
The Pulse Sensor is a plug-and-play heart-rate sensor for Arduino. It can be used by students, artists, athletes, makers, and game & mobile developers who want to easily incorporate live heart-rate data into their projects. The essence is an integrated optical amplifying circuit and noise eliminating circuit sensor. Clip the Pulse Sensor to our earlobe or fingertip and plug it into our Arduino, we can ready to read heart rate. Also, it has an Arduino demo code that makes it easy to use.



- **Pin-1(GND) :** Black Colour Wire - It is connected to the GND terminal of the system.
- **Pin-2(VCC) :** Red Colour Wire - It is connected to the supply voltage (+5V otherwise +3.3V) of the system.
- **Pin-3(Signal) :** Purple Colour Wire - It is connected to the pulsating o/p signal.



16*2 LCD Display :



The features of this LCD mainly :

- The operating voltage of this LCD is 4.7V-5.3V
- It includes two rows where each row can produce 16-characters
- The utilization of current is 1mA with no backlight
- Every character can be built with a 5×8 pixel box
- The alphanumeric LCDs alphabets & numbers
- Is display can work on two modes like 4-bit & 8-bit
- These are obtainable in Blue & Green Backlight
- It displays a few custom generated characters

ThingSpeak :

ThingSpeak allows you to aggregate, visualize and analyze live data streams in the cloud. Some of the key capabilities of ThingSpeak include the ability to:

- Easily configure devices to send data to ThingSpeak using popular IoT protocols.
- Visualize our sensor data in real-time.

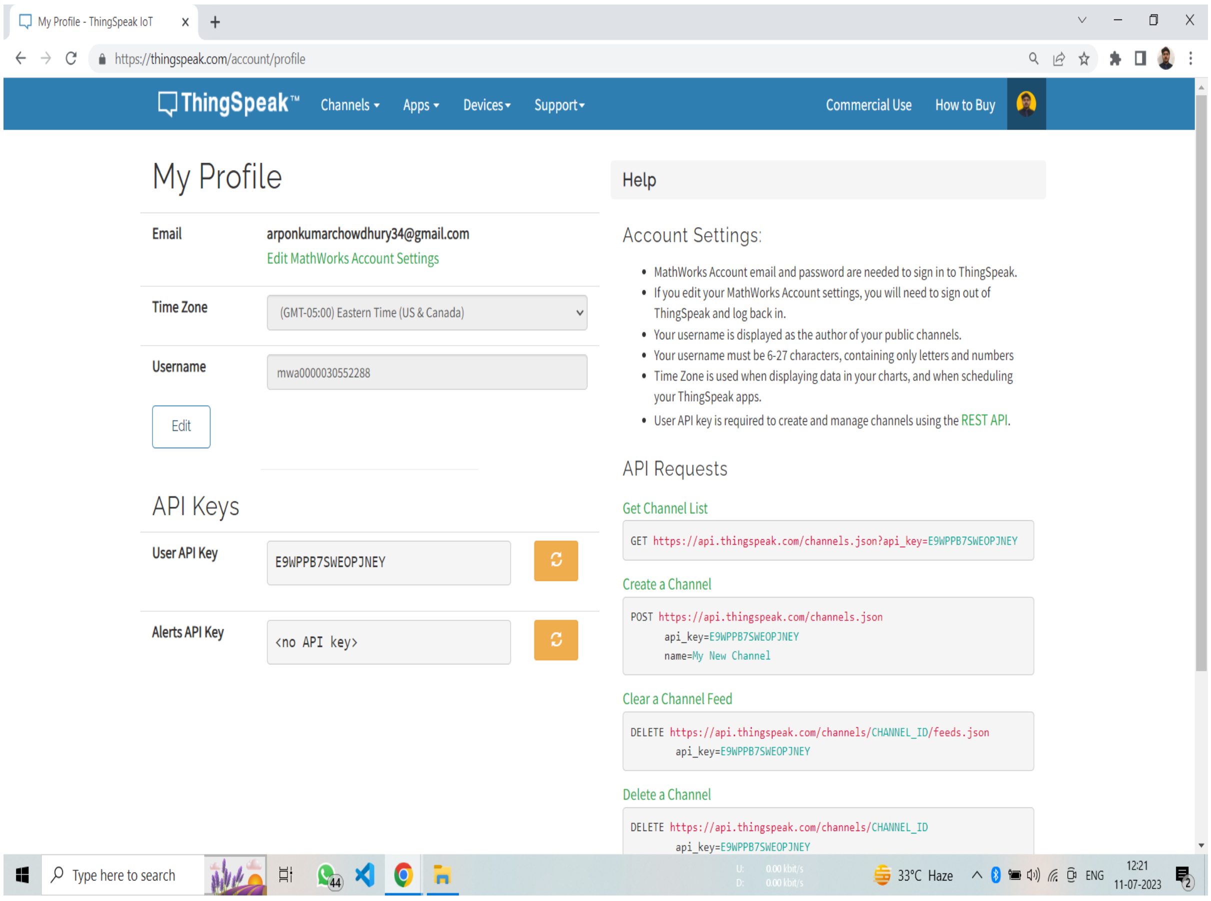
- Aggregate data on-demand from third-party sources.
- Use the power of MATLAB to make sense of our IoT data.
- Run our IoT analytics automatically based on schedules or events.
- Prototype and build IoT systems without setting up servers or developing web software.
- Automatically act on our data and communicate using third-party services like Twilio or Twitter

To learn how you can collect, analyze and act on our IoT data with ThingSpeak, explore the topics below:

Collect : Send sensor data privately to the cloud

Analyze : Analyze and visualize our data with matlab

Act: Trigger a reaction



My Profile

Email	arponkumarchowdhury34@gmail.com Edit MathWorks Account Settings
Time Zone	(GMT-05:00) Eastern Time (US & Canada)
Username	mwa0000030552288

Edit

API Keys

User API Key	E9WPPB7SWEOPJNEY
Alerts API Key	<no API key>

Help

Account Settings:

- MathWorks Account email and password are needed to sign in to ThingSpeak.
- If you edit your MathWorks Account settings, you will need to sign out of ThingSpeak and log back in.
- Your username is displayed as the author of your public channels.
- Your username must be 6-27 characters, containing only letters and numbers
- Time Zone is used when displaying data in your charts, and when scheduling your ThingSpeak apps.
- User API key is required to create and manage channels using the REST API.

API Requests

Get Channel List

```
GET https://api.thingspeak.com/channels.json?api_key=E9WPPB7SWEOPJNEY
```

Create a Channel

```
POST https://api.thingspeak.com/channels.json
     api_key=E9WPPB7SWEOPJNEY
     name=My New Channel
```

Clear a Channel Feed

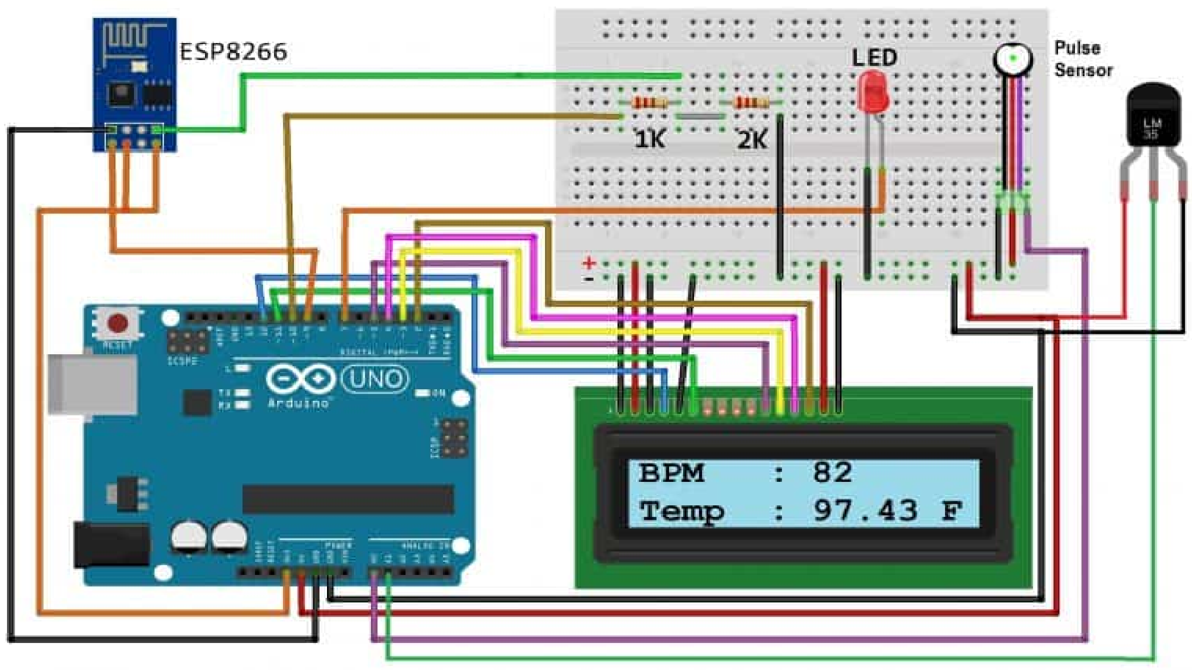
```
DELETE https://api.thingspeak.com/channels/CHANNEL_ID/feeds.json
       api_key=E9WPPB7SWEOPJNEY
```

Delete a Channel

```
DELETE https://api.thingspeak.com/channels/CHANNEL_ID
       api_key=E9WPPB7SWEOPJNEY
```

Circuit Diagram & Connections:

For designing IoT based patient health monitoring system using ESP8266 Wi-Fi module & Arduino, assemble the circuit as shown in the below figure :



1. Connect Pulse Sensor output pin to A0 of Arduino and other two pins to VCC and GND.

2. Connect LM35 Temperature Sensor output pin to A1 of Arduino and other two pins to VCC & GND.

3. Connect the LED to Digital pin 7 of Arduino via a 220-ohm resistor.

4.Connect pin 1,3,5,16 of LCD to GND.

5. Connect pin 2,15 of LCD to VCC.

6. Connect pin 4,6,11,12,13,14 of LCD to Digital pin 12,11,5,4,3,2 of Arduino.

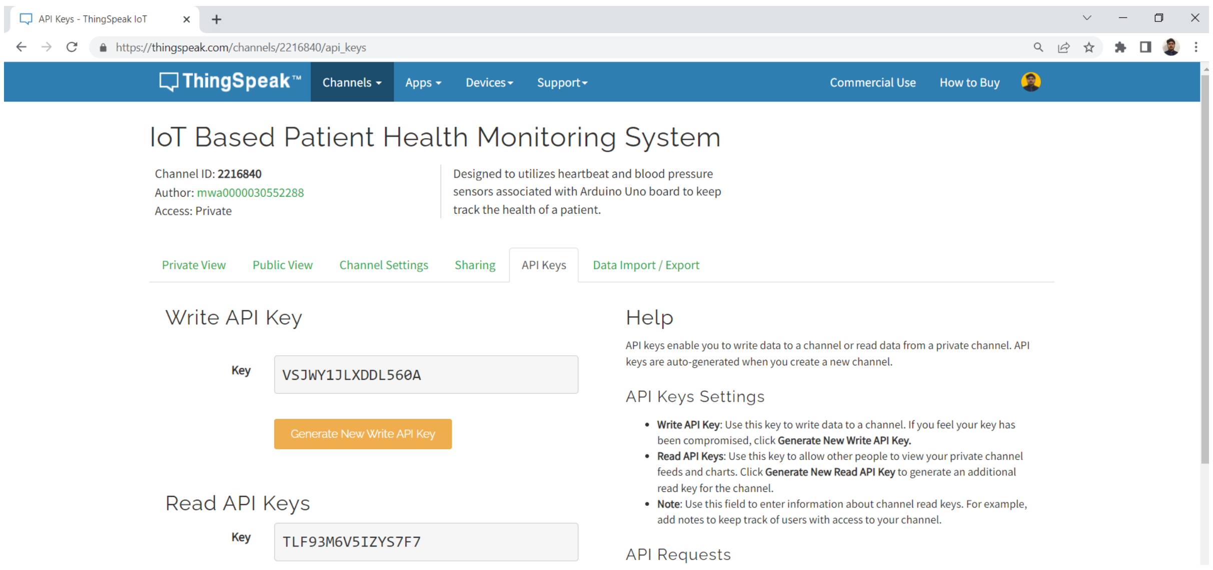
7. The RX pin of ESP8266 works on 3.3V and it will not communicate with the Arduino when we will connect it directly to the Arduino. So, we will have to make a voltage divider for it which will convert the 5V into 3.3V. This can be done by connecting the 2.2K & 1K resistor. Thus the RX pin of the ESP8266 is connected to pin 10 of Arduino through the resistors.

8. Connect the TX pin of the ESP8266 to pin 9 of the Arduino.

Setting the ThingSpeak:

ThingSpeak provides a very good tool for IoT based projects. By using the ThingSpeak site, we can monitor our data and control our system over the Internet, using the Channels and web pages provided by ThingSpeak. So first we need to sign up for ThingSpeak. So visit **https://thingspeak.com** and create an account.

Then create a New Channel and set up. After that, the API keys. The key is required for programming modifications and setting our data.



API Keys - ThingSpeak IoT
https://thingspeak.com/channels/2216840/api_keys
ThingSpeak™
Channels
Apps
Devices
Support
Commercial Use
How to Buy

IoT Based Patient Health Monitoring System

Channel ID: **2216840**
Author: mwa0000030552288
Access: Private

Designed to utilizes heartbeat and blood pressure sensors associated with Arduino Uno board to keep track the health of a patient.

Write API Key

Key VSJWY1JLXDDL560A

Read API Keys

Key TLF93M6V5IZYS7F7

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

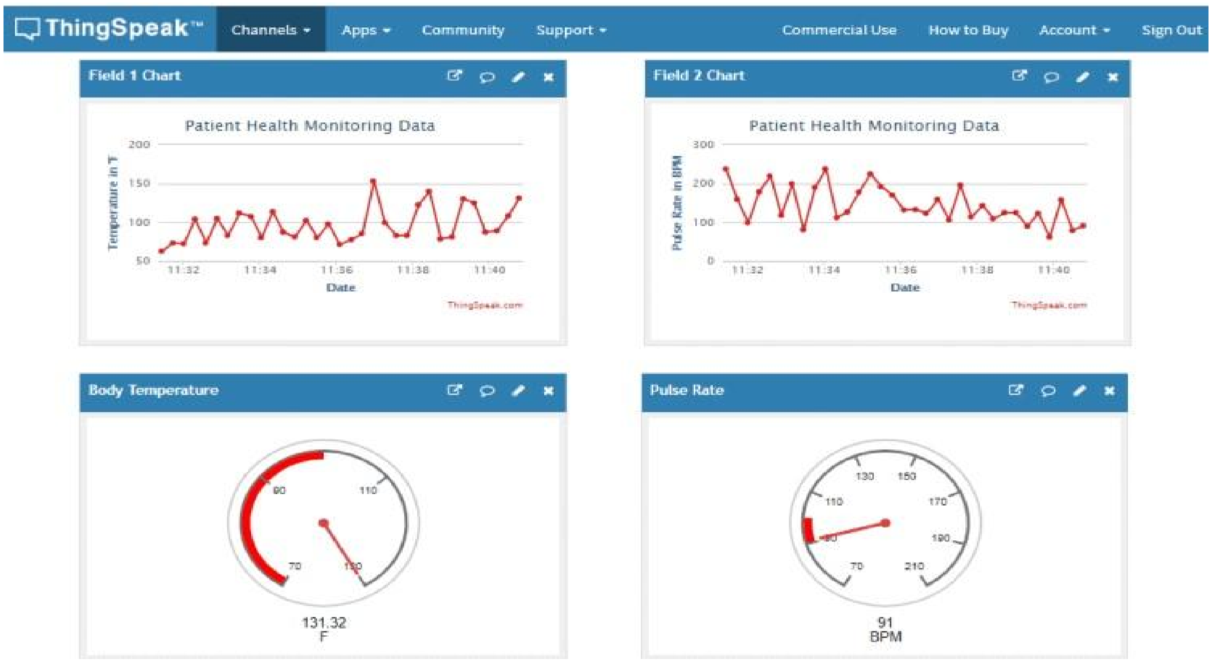
API Keys Settings

- **Write API Key**: Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key.**
- **Read API Keys**: Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- **Note**: Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

API Requests

Then upload the code to the Arduino Uno by assembling the circuit shown above. Open the serial monitor and it will automatically connected to Wi-Fi and set up everything.

Now click on channels so that we can see the online data streaming, i.e IoT Based Patient Health Monitoring System using ESP8266 & Arduino as shown in the figure here.



Program/Source Code:

The source code for the project IoT Based Patient Health Monitoring System using ESP8266 & Arduino is given below. We need to write it to Arduino IDE , then compile it and upload it to our Arduino Uno board.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);
#include <SoftwareSerial.h>
float pulse = 0;
float temp = 0;
SoftwareSerial ser(9,10);
String apiKey = "VSJWY1JLXDDL560A";

// Variables
int pulsePin = A0; // Pulse Sensor purple wire connected to analog pin
0
int blinkPin = 7 ; // pin to blink led at each beat
int fadePin = 13; // pin to do fancy classy fading blink at each beat
int fadeRate = 0; // used to fade LED on with PWM on fadePin

// Volatile Variables, used in the interrupt service routine!

volatile int BPM; // int that holds raw Analog in 0. updated every 2mS
volatile int Signal; // holds the incoming raw data
volatile int IBI = 600; // int that holds the time interval between
beats! Must be seeded!
volatile boolean Pulse = false; // "True" when User's live heartbeat is
detected. "False" when no "live beat".
volatile boolean QS = false; // becomes true when Arduoino finds a
beat.

// Regards Serial OutPut -- Set This Up to our needs
static boolean serialVisual = true; // Set to 'false' by Default.
Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse
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 = 525; // 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 setup()
{
lcd.begin(16, 2);
pinMode(blinkPin,OUTPUT); // pin that will blink to your heartbeat!
pinMode(fadePin,OUTPUT); // pin that will fade to your heartbeat!
Serial.begin(115200); // we agree to talk fast!
interruptSetup(); // sets up to read Pulse Sensor signal every 2mS

// UN-COMMENT THE NEXT LINE AND APPLY THAT VOLTAGE TO THE A-REF PIN

// analogReference(EXTERNAL);

lcd.clear();
lcd.setCursor(0,0);
lcd.print(" Patient Health");
lcd.setCursor(0,1);
lcd.print(" Monitoring ");
delay(4000);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Initializing....");
delay(5000);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Getting Data....");
ser.begin(9600);
ser.println("AT");
delay(1000);
ser.println("AT+GMR");
delay(1000);
ser.println("AT+CWMODE=3");
delay(1000);
ser.println("AT+RST");
delay(5000);
ser.println("AT+CIPMUX=1");
delay(1000);

String cmd="AT+CWJAP=\"Alexahome\",\"98765432\"";
ser.println(cmd);
delay(1000);
ser.println("AT+CIFSR");
delay(1000);
}

// Where the Magic Happens
void loop()
{
serialOutput();
if (QS == true) // A Heartbeat Was Found
{

// BPM and IBI have been Determined
// Quantified Self "QS" true when arduino finds a heartbeat
fadeRate = 255; // Makes the LED Fade Effect Happen, Set 'fadeRate'
Variable to 255 to fade LED with pulse
serialOutputWhenBeatHappens(); // A Beat Happened, Output that to
serial.
QS = false; // reset the Quantified Self flag for next time
}
ledFadeToBeat(); // Makes the LED Fade Effect Happen
delay(20); // take a break
read_temp();
esp_8266();
}
void ledFadeToBeat()
{
fadeRate -= 15; // set LED fade value
fadeRate = constrain(fadeRate,0,255); // keep LED fade value from going
into negative numbers!
analogWrite(fadePin,fadeRate); // fade LED
}
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
}
```

```
void serialOutput()
{ // Decide How To Output Serial.
if (serialVisual == true)
{
arduinoSerialMonitorVisual('-', Signal); // goes to function that makes
Serial Monitor Visualizer
}
else
{
sendDataToSerial('S', Signal); // goes to sendDataToSerial function
}
}
void serialOutputWhenBeatHappens()
{
if (serialVisual == true) // Code to Make the Serial Monitor Visualizer
Work
{
Serial.print("*** Heart-Beat Happened *** "); //ASCII Art Madness
Serial.print("BPM: ");
Serial.println(BPM);
}
else
{
sendDataToSerial('B',BPM); // send heart rate with a 'B' prefix
sendDataToSerial('Q',IBI); // send time between beats with a 'Q' prefix
}
}
void arduinoSerialMonitorVisual(char symbol, int data )
{
const int sensorMin = 0; // sensor minimum, discovered through
experiment
const int sensorMax = 1024; // sensor maximum, discovered through
experiment
int sensorReading = data; // map the sensor range to a range of 12
options:
int range = map(sensorReading, sensorMin, sensorMax, 0, 11);
// do something different depending on the
// range value:
switch (range)
{
case 0:
Serial.println(""); /////ASCII Art Madness
break;
case 1:
Serial.println("---");
break;
case 2:
Serial.println("------");
break;
case 3:
Serial.println("---------");
break;
case 4:
Serial.println("------------");
break;
case 5:
Serial.println("--------------|-");
break;
case 6:
Serial.println("--------------|---");
break;
case 7:
Serial.println("--------------|-------");
break;
case 8:
Serial.println("--------------|----------");
break;
case 9:
Serial.println("--------------|----------------");
break;
case 10:
Serial.println("--------------|-------------------");
break;
case 11:
Serial.println("--------------|-----------------------");
break;
}
}

void sendDataToSerial(char symbol, int data )
{
Serial.print(symbol);
Serial.println(data);
}
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
pulse = BPM;
}
}
if (Signal < thresh && Pulse == true)
{ // when the values are going down, the beat is over
digitalWrite(blinkPin,LOW); // turn off pin 13 LED
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 we are done!
}// end is
```

```
void esp_8266()
{
// TCP connection AT+CIPSTART=4,"TCP","184.106.153.149",80
String cmd = "AT+CIPSTART=4,\"TCP\",\"";
cmd += "184.106.153.149"; // api.thingspeak.com
cmd += "\",80";
ser.println(cmd);
Serial.println(cmd);
if(ser.find("Error"))
{
Serial.println("AT+CIPSTART error");
return;
}
String getStr = "GET /update?api_key=";
getStr += apiKey;
getStr +="&field1=";
getStr +=String(temp);
getStr +="&field2=";
getStr +=String(pulse);
getStr += "\r\n\r\n";
// send data length
cmd = "AT+CIPSEND=4,";
cmd += String(getStr.length());
ser.println(cmd);
Serial.println(cmd);
delay(1000);
ser.print(getStr);
Serial.println(getStr); //thingspeak needs 15 sec delay between updates
delay(3000);
}
void read_temp()
{
int temp_val = analogRead(A1);
float mv = (temp_val/1024.0)*5000;
float cel = mv/10;
temp = (cel*9)/5 + 32;
Serial.print("Temperature:");
Serial.println(temp);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("BPM :");
lcd.setCursor(7,0);
lcd.print(BPM);
lcd.setCursor(0,1);
lcd.print("Temp.:");
lcd.setCursor(7,1);
lcd.print(temp);
lcd.setCursor(13,1);
lcd.print("F");
}
```

Conclusion:

- Our proposed system aims at simplifying health monitoring.
- This system of ours has been proven effective enough in aiding medical assistance.
- This approach of ours contributes towards Smart India.

References:

❖ Marathe, Sachi, et al. “A Wireless Patient Monitoring System using Integrated ECG module, Pulse Oximeter, Blood Pressure and Temperature Sensor.” 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN). IEEE, 2019.

❖ Mishra, Vaibhav, Durgesh Kumar Mishra and Ass. Prof. Ankit Trivedi. “Health Monitoring System using IoT using Arduino Uno Microcontroller.” Health 6.08 (2019).

By **Arpon Kumar Chowdhury, KGEC_CSE’25**