Internet of Things

TEMPERATURE BASED FAN SPEED CONTROLLER

Team Members:

Sl no	Student ID	Student Name
1	20181CSE0211	G VIVEKANANDHAN
2	20181CSE0264	HARSHA HASADYA SON NT
3	20181CSE0265	HARSHA K
4	20181CSE9046	PALTERU KAUSHIK

Aim:

To control the speed of the fan based on the surrounding temperature automatically.

OBJECTIVE:

The main objective of our project is to control the speed of the fan automatically based on the surrounding room temperature without any human intervention and monitor the parameters such as temperature, humidity and fan speed on the OLED screen. Our other objectives include sending the data of these parameters to a cloud platform and visualize the data in the form of graphs, and sending the notifications to our mobile phones using the Blynk application.

ABSTRACT:

Often, we are tired of reducing the fan speed manually when we are sleeping uncomfortably during the chilly winter nights. We also feel annoyed when the fan is running slow during the scorching summer afternoons. We are also worried about how much power we're wasting everyday regardless of the weather.

Say no more hassle to changing the fan speed manually or worrying about the power consumption. We are presenting our project **A Temperature-Based Automatic Fan Speed Controller** which controls the speed of the fan based on the room temperature automatically without us having a need to shed a drop of sweat.

We also wonder if we can facilitate the idea of monitoring the fan speed at a particular temperature on our devices like computers and mobile phones. So, we decided to send this data i.e., the values of temperature, humidity and fan speed to an online cloud platform called **ThingSpeak** and to our mobile phones in the form of notifications using a mobile application called **Blynk,** to monitor the fan speed at a particular temperature.

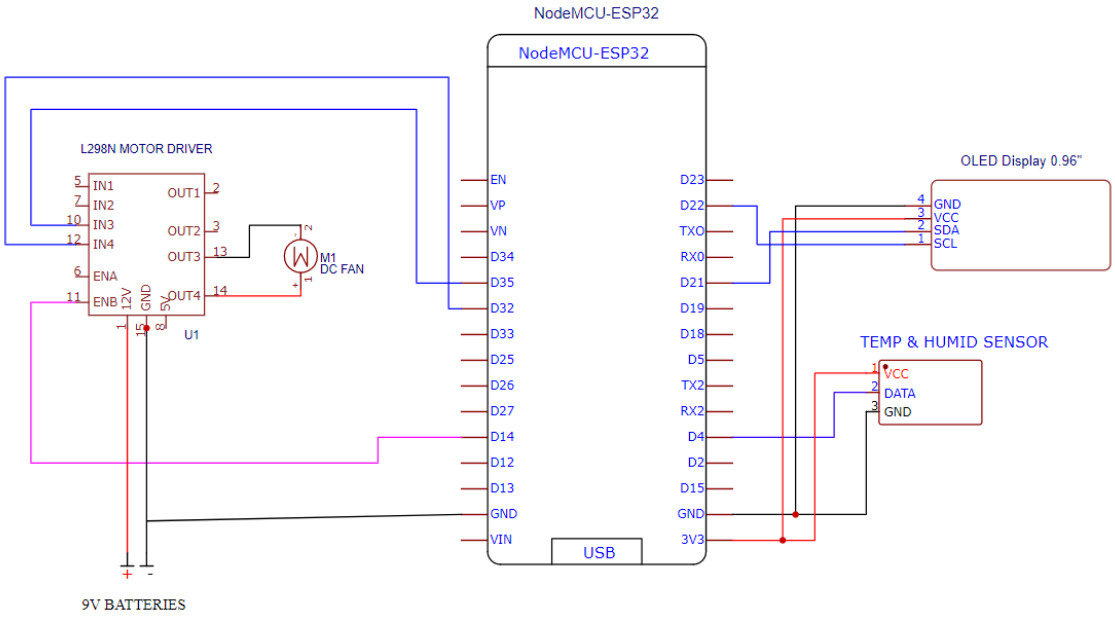
The main components used in our project are:

1. NodeMCU ESP32 Micro Controller
2. DHT11 – A temperature and humidity sensor
3. L298N Motor Driver
4. 12V DC Fan

Other components used are:

1. 0.96 inches I2C/IIC OLED 128 x 96 screen
2. Breadboard
3. 9V Batteries
4. Jumper wires
5. USB cable

Pin-out Diagram:



DHT11 Sensor:

- GND → GND pin of ESP32
- VCC → 3.3V
- DATA → D4

OLED Display:

- GND → GND pin of ESP32
- VCC → 3.3V
- SDA → D21
- SCL → D22

L298N Motor Driver:

- GND → GND pin of ESP32
- GND → Negative terminal of Battery
- 12V → Positive terminal of Battery
- ENB → D14
- In3 → D35
- In4 → D32
- out3 → Negative terminal of Fan
- out4 → Positive terminal of Fan

DC Fan:

- Positive terminal → out4 of L298N
- Negative terminal → out3 of L298N

Working:

Use of the embedded technology plays a vital role in making this control system **efficient** and **reliable**.

NodeMCU ESP32 micro controller is the main part of the circuit system to which every other component is connected.

DHT11 humidity and temperature sensor helps in sensing the temperature from the surroundings and convert it into electrical signals to the micro controller.

0.96-inch OLED display is used to display the values of parameters like temperature, fan speed, humidity.

The temperature sensed from the **DHT11** sensor is sent to the **NodeMcu micro controller** (in the form of electrical signals) which performs some computations and control the speed of the **DC fan** though **L298N Motor Driver** according to the temperature. Then the temperature and fan speed are displayed on the **OLED display.**

L298N motor driver acts as a bridge between the **DC fan** and the **NodeMcu controller.** It acts as a power supply to the **DC fan** and also helps in controlling the speed of the DC **fan.**

We used the **PWM** technique to control the speed of the fan by sending the varying PWM signals. Yes. There is a dedicated hardware block for PWM in the silicon of **ESP32**. Pulse Width Modulation or PWM in short is an established and widely used techniques for power delivery. The **PWM Controller** in ESP32 consists of two main sub-modules: LEDC Peripheral and MCPWM Peripheral. We'll be using the **LEDC peripheral** for our project due to its ease of use. The LEDC Peripheral of ESP32 consists of 16 PWM Channels capable of generating independent waveforms.

There are a couple of interesting points about LED PWM Controller in ESP32 that you should be aware of.

- 16 independent PWM Channels, divided into group of two with 8 channels per group.
- Programmable resolution between 1-bit and 16-bits.
- Frequency of the PWM wave depends on the resolution of PWM.
- Automatically increases / decreases duty cycle without processor intervention.

Do you remember **'analogWrite()'** function in Arduino programming? It is the function responsible for generating PWM in Arduino UNO (and other 'Arduino' boards). Since, pretty much every thing in LED PWM of ESP32 is **user configurable (channel, resolution and frequency)**, instead of using 'analogWrite()' function, we used a different (and dedicated) set of functions to configure PWM in ESP32. The functions we used are :

- ***ledcSetup(channel, frequency, resolution_bits);***
- ***ledcAttachPin(pin, channel);***
- ***ledcWrite(channel, dutycycle);***

Depending on the frequency and resolution bits of the PWM signal, we were able to calculate the range of the duty cycle we wanted to use for our project. We took the frequency as 5000 Hz and resolution bits as 10. We calculate the duty cycle using the formula:

Duty cycle = (2^resolution bits) -1

Duty cycle = (2^10) - 1 = 1023

The range of the duty cycle at **5 KHz frequency** using **10 resolution bits** is **0 – 1023.** We start with a duty cycle of 400 (you can set a duty cycle value from 0 to 1023). For the frequency we're using, when you apply duty cycles smaller than 400, the fan won't move and will make a weird buzz sound. So, that's why we set a duty cycle of 400 at the start.

The duty cycle is directly proportional to the speed of the fan. The more the duty cycle, the more the fan speed will be. So, we decided to control the fan speed according to the temperature by setting the duty cycle to a suitable value. For example, we've set the duty cycle value to 520 at a temperature ranging from 27 to 30 °C. Duty cycle of value 520 corresponds to 20 % of the fan speed percent if consider the duty cycle range of 400 – 1023.

Similarly, we set the duty cycle values according to the change in temperature. Thus, we control the fan speed according to the temperature.

Duty Cycle values according to various temperature values:

Fan Speed Level	Duty Cycle	Fan Speed Percent	Temperature Range
0	0	0 %	<27°C
1	520	20 %	27°C - 30°C
2	640	40 %	30°C - 33°C
3	760	60 %	33°C - 36°C
4	880	80 %	36°C - 39°C
5	1023	100 %	39°C

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak. With the ability to execute MATLAB code in ThingSpeak you can perform online analysis and processing of the data as it comes in. ThingSpeak is often used for prototyping and proof of concept IoT systems that require analytics. We send the data to ThingSpeak, we visualize and analyze the data in the form of graphs in a private channel so that our data cannot be seen or manipulated by anyone except us. We can say that our data is safe and secure with the ThingSpeak's private channel feature.

Blynk is a Platform with IOS and Android apps to control Arduino, NodeMCU and the likes over the Internet. It's a digital dashboard where you can build a graphic interface for your project by simply dragging and dropping widgets. We can receive notifications from the hardware to our mobile devices using Blynk app. We facilitated the idea of receiving the temperature and corresponding fan speed data on our mobile devices in the form of notifications. We can also monitor the temperature and fan speed data in the form of a graphically rich gauge widget in the Blynk app.

Program for our project:

```
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <HTTPClient.h>
#include "DHT.h"
#include <Wire.h>
#include <SPI.h>
#include <ACROBOTIC_SSD1306.h>

#define DHTPIN 4     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);
BlynkTimer timer;

int buttonState = 0;
int motor1Pin3 = 35;
int motor1Pin4 = 32;
int enable1Pin = 14;
const int freq = 5000;
const int pwmChannel = 0;
const int resolution = 10;
int dutyCycle = 0;
int fanSpeedPercent = 0;

// You should get Auth Token in the Blynk App.
```

```
// Go to the Project Settings (nut icon).
char auth[] = "kTB9RXVBW72rwkjdY6fSow5iZATejcru";

const char* ssid = "**********";
const char* password = " ********** ";

String serverName = "https://api.thingspeak.com/update?api_key=9YNTT4KW3AIL4615";

unsigned long lastTime = 0;
unsigned long timerDelay = 60000;

float x = 0; //Temporary Variable

void setup() {

  // put your setup code here, to run once:
  pinMode(motor1Pin3, OUTPUT);
  pinMode(motor1Pin4, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  // configure PWM functionalities
  ledcSetup(pwmChannel, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, pwmChannel);

  Serial.begin(9600);
  Serial.println(F("DHTxx test!"));
```

```
  Wire.begin();
  oled.init();                    // Initialze SSD1306 OLED display
  oled.clearDisplay();

  Blynk.begin(auth, ssid, password);
  dht.begin();
  timer.setInterval(25000L, sendSensor);

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  Blynk.run();
  timer.run();
}

void

void sendSensor() {
```

```
float hmdt = dht.readHumidity();
// Read temperature as Celsius (the default)
float temp = dht.readTemperature();
// Check if any reads failed and exit early (to try again).
if (isnan(hmdt) || isnan(temp) ) {
  Serial.println(F("Failed to read from DHT sensor!"));
  return;
}

Serial.print(F("Humidity: "));
Serial.print(hmdt);

Serial.print(F("%  Temperature: "));
Serial.print(temp);

oled.setTextXY(1,0);// Clear screen
oled.putString("Humidity:");
oled.setTextXY(1,10);
oled.putNumber(hmdt);

oled.setTextXY(3,0);// Clear screen
oled.putString("Temperature:");
oled.setTextXY(3,13);
oled.putNumber(temp);

digitalWrite(enable1Pin, HIGH);

//Move DC motor forward with increasing speed
```

```
  digitalWrite(motor1Pin3, LOW);
  digitalWrite(motor1Pin4, HIGH);

  if(temp<27) {
    digitalWrite(enable1Pin, LOW);
    delay(1000);
    oled.setTextXY(5,0);// Clear screen
    oled.putString("Fan Speed:");
    oled.setTextXY(5,11);
    oled.putNumber(fanSpeedPercent);
  }

  if(temp>=27 && temp<30) {
    dutyCycle = 520;
    fanSpeedPercent = 20;

    ledcWrite(pwmChannel, dutyCycle);
    Serial.println(" The fan is rotating at " + String(fanSpeedPercent) + "%");  // I have a doubt regarding this

    delay(1000);
    oled.setTextXY(5,0);// Clear screen
    oled.putString("Fan Speed:");
    oled.setTextXY(5,11);
    oled.putNumber(fanSpeedPercent);

  }

  if(temp>=30 && temp<33) {
```

```
  dutyCycle = 640;
  fanSpeedPercent = 40;
  ledcWrite(pwmChannel, dutyCycle);
  Serial.println(" The fan is rotating at " + String(fanSpeedPercent) + "%");

  delay(1000);
  oled.setTextXY(5,0);// Clear screen
  oled.putString("Fan Speed:");
  oled.setTextXY(5,11);
  oled.putNumber(fanSpeedPercent);

 }

 if(temp>=33 && temp<36) {
  dutyCycle = 760;
  fanSpeedPercent = 60;
  ledcWrite(pwmChannel, dutyCycle);
  Serial.println(" The fan is rotating at " + String(fanSpeedPercent) + "%");

  delay(1000);
  oled.setTextXY(5,0);// Clear screen
  oled.putString("Fan Speed:");
  oled.setTextXY(5,11);
  oled.putNumber(fanSpeedPercent);

 }

 if(temp>=36 && temp<39) {
```

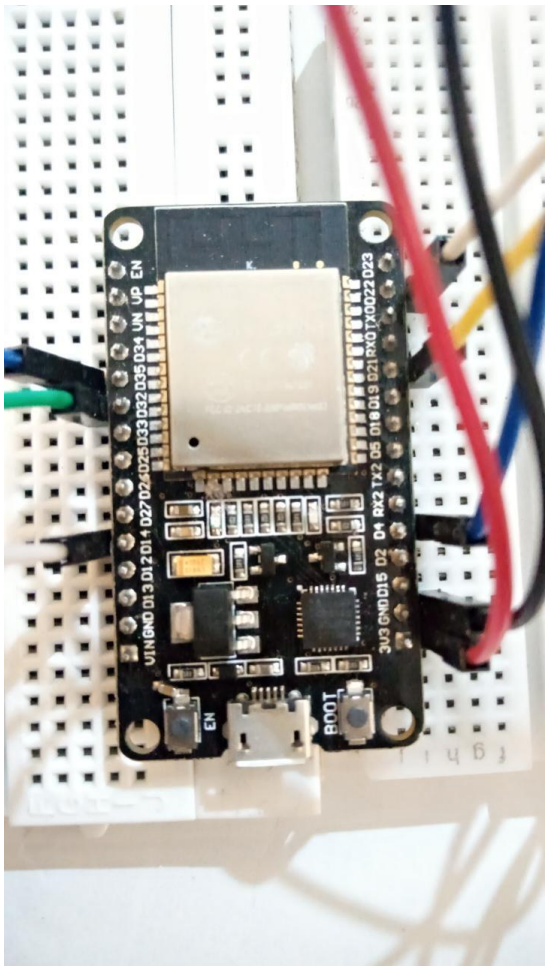
```
  dutyCycle = 880;
  fanSpeedPercent = 80;
  ledcWrite(pwmChannel, dutyCycle);
  Serial.println(" The fan is rotating at " + String(fanSpeedPercent) + "%");

  delay(1000);
  oled.setTextXY(5,0);// Clear screen
  oled.putString("Fan Speed:");
  oled.setTextXY(5,11);
  oled.putNumber(fanSpeedPercent);

 }

 if(temp>=39) {
  dutyCycle = 1023;
  fanSpeedPercent = 100;
  ledcWrite(pwmChannel, dutyCycle);
  Serial.println(" The fan is rotating at " + String(fanSpeedPercent) + "%");

  delay(1000);
  oled.setTextXY(5,0);// Clear screen
  oled.putString("Fan Speed:");
  oled.setTextXY(5,11);
  oled.putNumber(fanSpeedPercent);

 }

 Blynk.virtualWrite(V6, temp);
```

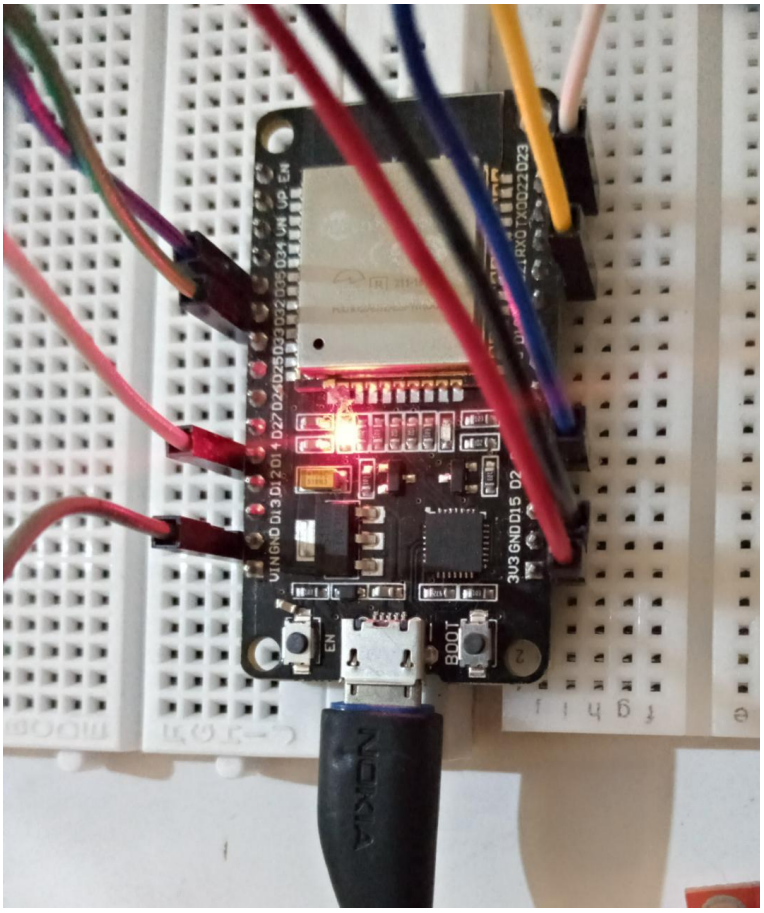
```
  Blynk.virtualWrite(V7, fanSpeedPercent);


 Serial.println("ESP32 Alert - Temperature = " + String(temp) + " °C," + " Fan Speed = " + String(fanSpeedPercent)+ "%");

w

 if(int(x) != int(temp)) {

     Blynk.notify("ESP32 Alert - Temperature = " + String(temp) + " °C," + " Fan Speed = " + String(fanSpeedPercent)+ "%" );    // I have a doubt regarding this

  }

  x = temp;

  Serial.println(x);


  if ((millis() - lastTime) > timerDelay) {

    if(WiFi.status()== WL_CONNECTED){

    // Read data from our dht object and pass it to our sendData function

        sendData(temp, fanSpeedPercent, hmdt);

    }

    else {

      Serial.println("WiFi Disconnected");

    }


   lastTime = millis();

  }

}


void sendData(double temp, int fanSpeedPercent, double hmdt){

    HTTPClient http;

   String url = serverName + "&field1=" + temp + "&field2=" + fanSpeedPercent + "&field3=" + hmdt ;
```

```
  http.begin(url.c_str());

  int httpResponseCode = http.GET();

  if (httpResponseCode > 0){
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
  }else{
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  http.end();
}
```

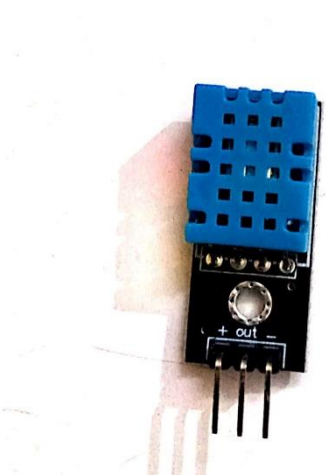
Picture of Each Module with Circuit Connections:

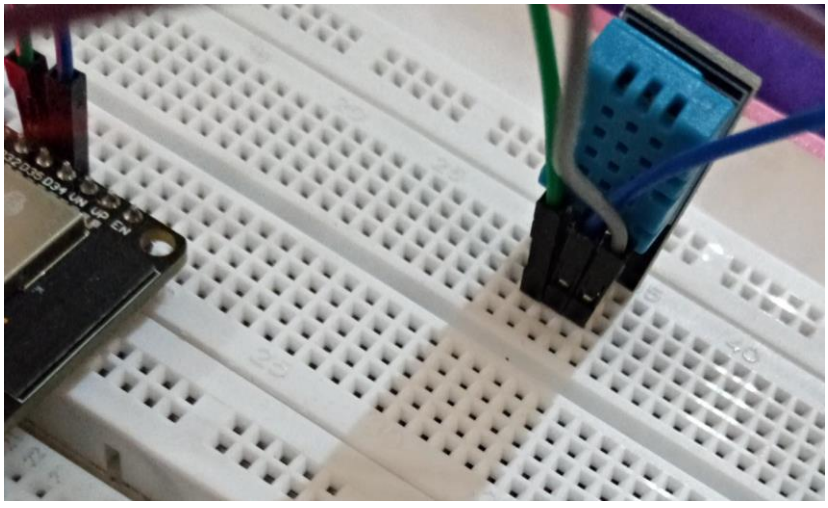
NodeMCU ESP32:



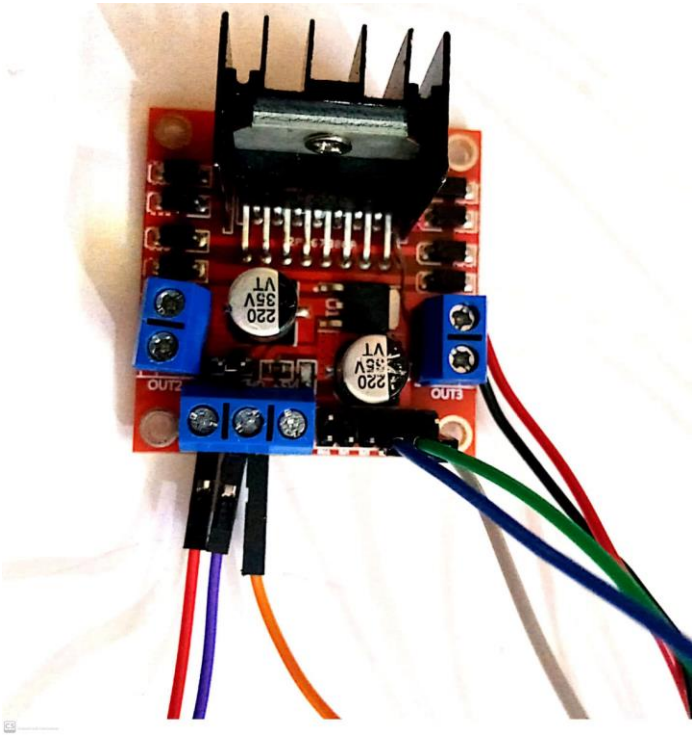


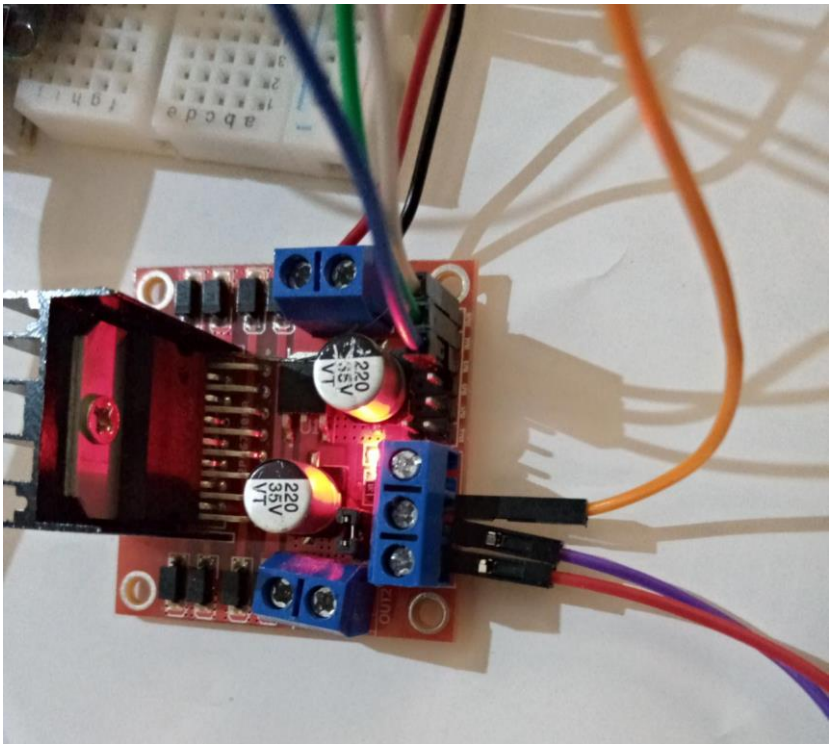
DHT11:



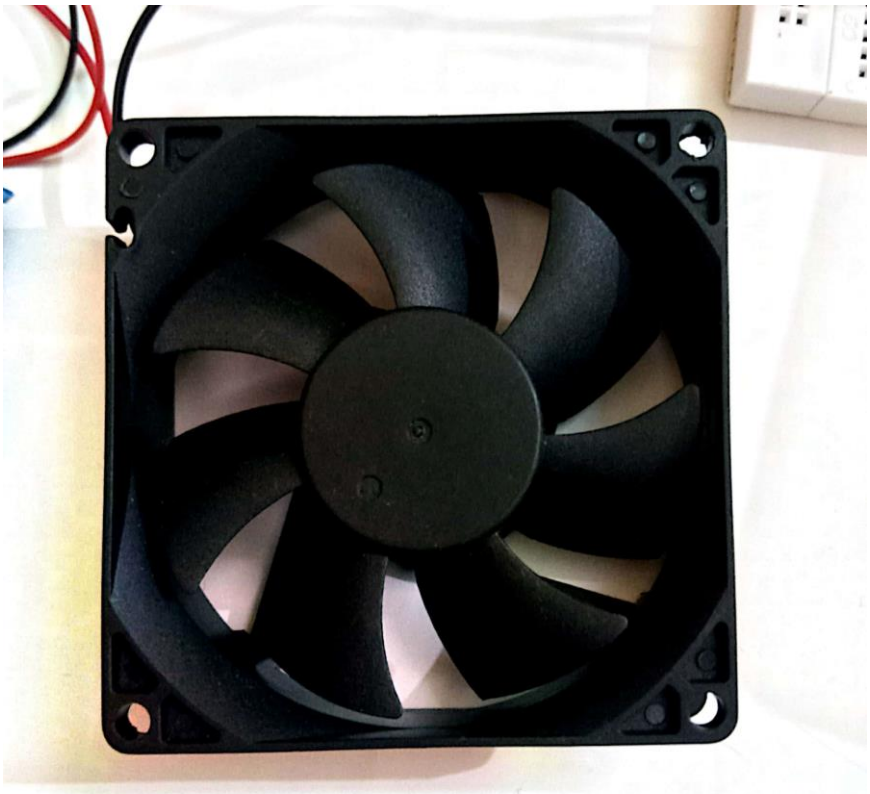


L298N Motor Driver:



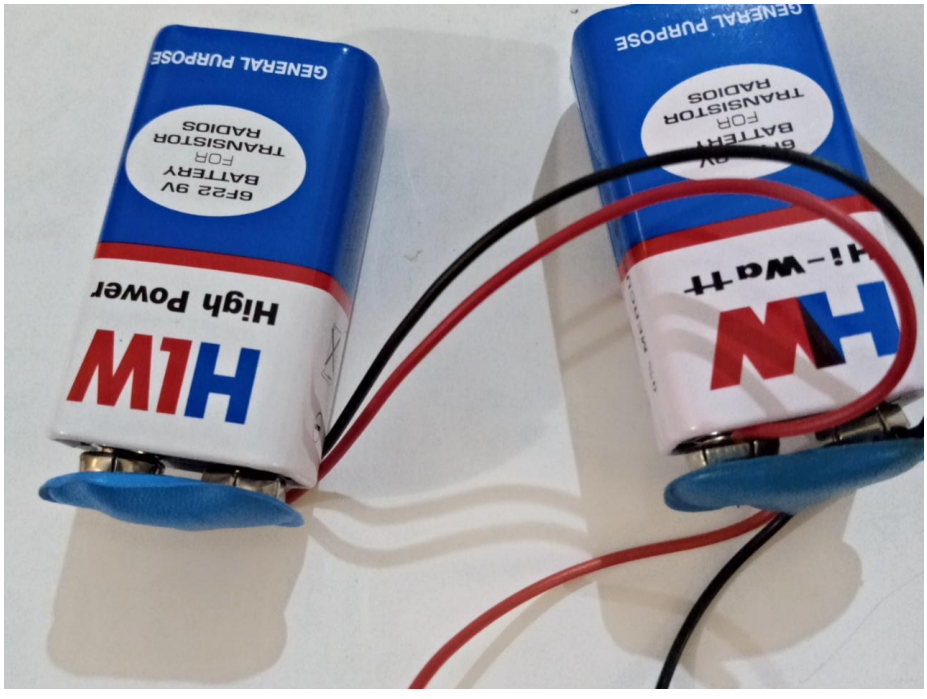


DC Fan:

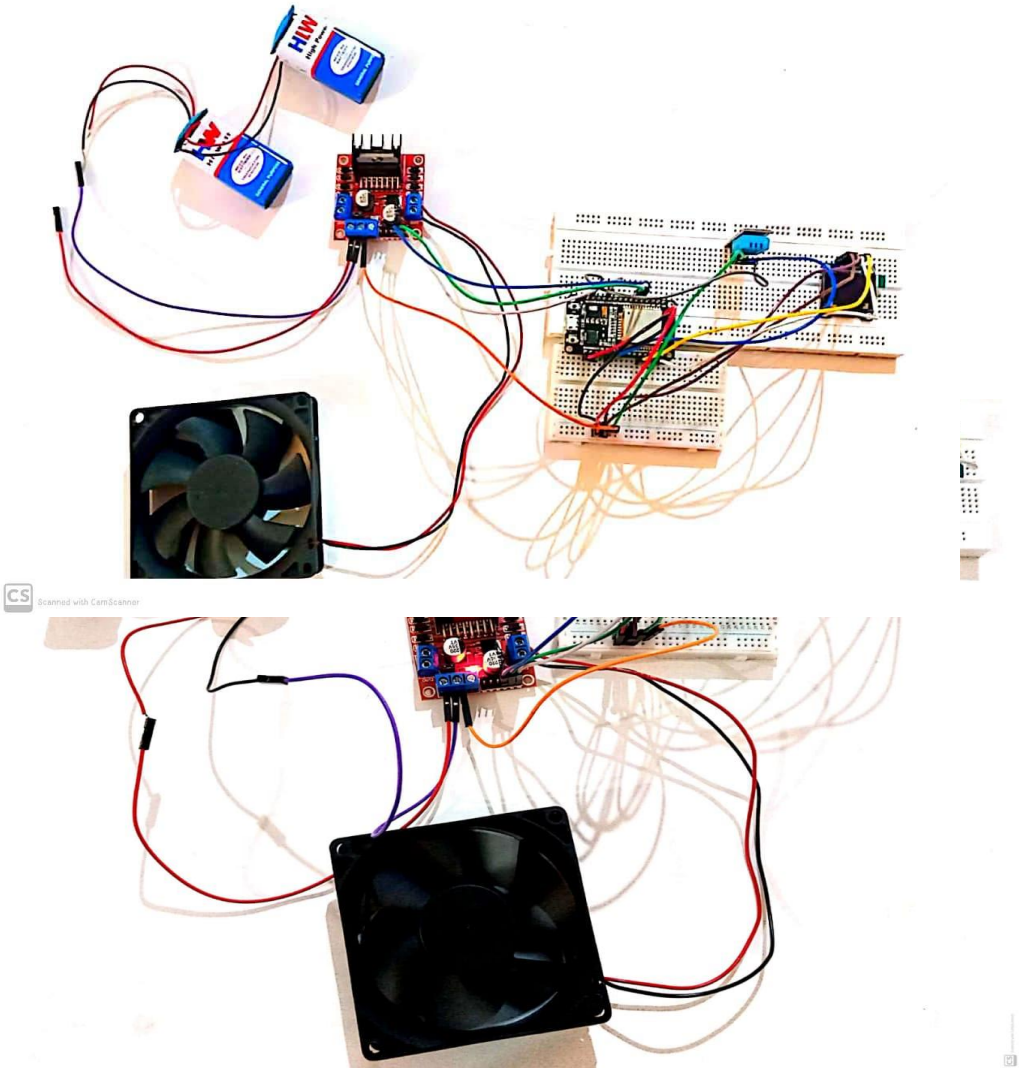


OLED Display:

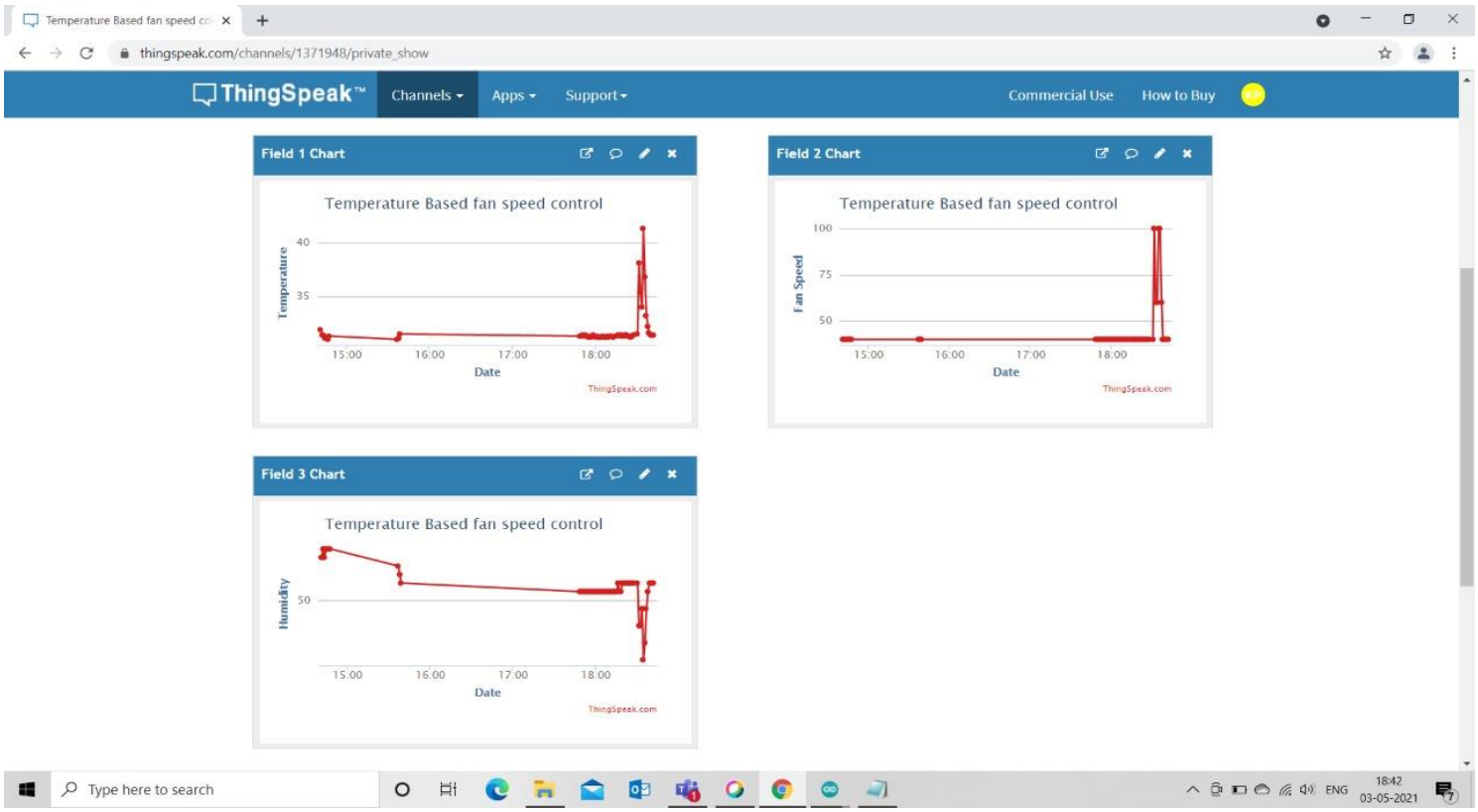
9V Batteries:



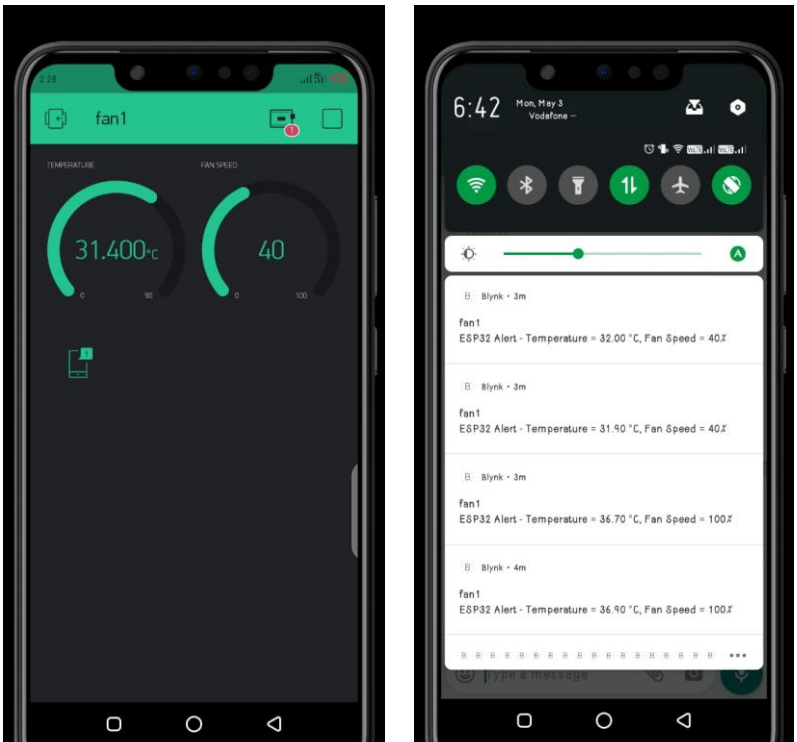
The Complete picture of our Project:



Data Visualization using ThingSpeak:



Data Monitoring and Notifications on Mobile Phones using Blynk App:



Conclusion:

With the help of our project, we are able to facilitate the idea of controlling the speed of the fan automatically based on the surrounding temperature without any human intervention. We are also able to monitor the temperature and fan speed on the LED panel.

We are able to send the data to an online cloud platform and visualize the data. We are also able to send the notifications to our mobile phones and monitor the data using Blynk app.