

DEPARTMENT OF PHYSICS  
UNIVERSITY OF COLOMBO

PH 3039- Data Acquisition Laboratory  
2022

**Construction Simple Weather Station  
using ESP32 Device**

Name: D.P. Sasindu Dilhara

Index: s14811

Date: 15/09/2022

## ABSTRACT

Weather monitoring deals with tracking, and possible prediction of the weather conditions of a geographical area. The system is built around a collection of different sensors and a microcontroller. The primary goal is to create a low cost, adaptable system for storing measured data, it is an advanced and efficient solution for connection to the internet. The ESP32 microcontroller is in the series of low power and low cost on chip microcontroller which comes up with already integrated dual mode Bluetooth and Wi-Fi. DHT11 sensor is used to get the temperature and humidity data, BMP 180 is used to get pressure and altitude data, and to get luminosity data, LDR is used. Then weather data will be recorded on google sheet and analyzes them with Grafana and creates the dashboard.

# TABLE OF CONTENTS

1	INTRODUCTION .....	1
2	THEORY .....	2
2.1	Esp32 Wi-Fi Bluetooth module .....	2
2.1.1	ESP32 power consumption .....	2
2.1.2	ESP32 standalone .....	3
2.2	DHT11 Sensor .....	3
2.3	BME180 Pressure sensor .....	4
2.4	LDR sensor .....	5
3	METHODOLOGY.....	6
3.1	PCB design and making.....	6
3.2	Installing the ESP32 in Arduino IDE .....	8
3.3	Creating Google Sheet and Data Logger .....	10
3.4	Data Visualization using ThingSpeak .....	13
4	RESULTS AND ANALYSIS .....	15
4.1	LDR sensor calibration .....	15
4.2	Data logger and analyze .....	16
5	DISCUSSION .....	22
6	CONCLUSION .....	23
7	References.....	24

# TABLE OF FIGURES

Figure 1 ESP32 Wi-Fi Module .....	2
Figure 2: Comparison of light sleep mode and deep sleep mode of ESP32 .....	3
Figure 3: DHT11 sensor .....	3
Figure 4: BMP180 sensor .....	4
Figure 5: How to connect BMP180 with ESP32.....	4
Figure 6: LDR sensor .....	5
Figure 7: PCB design using EasyEDA software .....	6
Figure 8: Structure of the final PCB design .....	6
Figure 9: Real PCB marking .....	7
Figure 10: Adding the components to the PCB.....	7
Figure 11: Final look of the weather station .....	8
Figure 12: open the Arduino IDE and go to file preferences .....	8
Figure 13: Enter the additional Board Manager URLs .....	9
Figure 14: Selected the Boards Manager .....	9
Figure 15: Installed the ESP32 Systems.....	10
Figure 16: IFTTT web service interface.....	10
Figure 17: If This Then That applets .....	11
Figure 18: Search the Webhooks service and receive a web request .....	11
Figure 19: Create a new trigger in Webhooks .....	11
Figure 20: Choose the Add row to spreadsheet option .....	12
Figure 21: Google Sheet create in the google drive within event name .....	12
Figure 22: "ThingSpeak" channel setting .....	13
Figure 23: Given Latitude and Longitude of station location.....	13
Figure 24: API Keys and channel ID of ThingSpeak .....	14
Figure 25: Graph of Lux value vs Resistor value of LDR in different light intensity .....	15
Figure 26: ThingSpeak get Latitude and longitude of the weather station location .....	17
Figure 27: Temperature graph of ThingSpeak .....	17
Figure 28: Average temperature in Data collecting period .....	18
Figure 29: Humidity variation dashboard in ThingSpeak.....	18
Figure 30: Average humidity value in data collecting period .....	19
Figure 31: Pressure variation in ThingSpeak.....	19
Figure 32: Average Pressure in data collecting period .....	20
Figure 33: Station 3 & 4 Temperature and Humidity graphs.....	20
Figure 34: Station 5 & 6 Temperature and Humidity graphs.....	21
Figure 35: Station 7 & 8 Temperature and Humidity graphs.....	21
Figure 36: Station 9 & 10 Temperature and Humidity graphs.....	21

## LIST OF TABLES

Table 1: Lux values with ADC values(resistance) for LDR calibration.....	15
Table 2: Enter the data using WIFI to the Google Sheet.....	16

# 1 INTRODUCTION

Weather stations have become quite popular nowadays, as they can forecast everything regarding weather more accurately, unlike older devices. It is a low cost, adaptable device that collects data in real time, related to the weather and environment using different sensors. Commonly measured environmental variables include temperature, light, relative humidity, rain and wind. It instantly provides alerts regarding abnormal changes in the weather and can share these weather data on various devices by connecting the weather station to a WIFI network.

DHT11 and BMP180 sensors are used to collect the data from their surrounding environment and then communicate the data to the ESP32 microcontroller. DHT11 sensor is used to measure humidity and temperature and the BMP180 sensor is used to measure the barometric pressure. They are connected via a network and then data can be retrieved for analysis and processing. Weather data read by BME180 will be recorded on google sheet. The system is a development of weather reporting with an ESP 32 which serves as the central microcontroller. The sensors are linked to it with digital and analog pins. Esp32 receives readings from sensors in the environment and process them before displaying their results on the ESP serial monitor.

“ThingSpeak” web service is an open-source data platform for the IoT applications which allows us to send data to the cloud. The data collected by the sensors is published to the ThingSpeak web server.

## 2 THEORY

A weather station is a building, on land or at sea, that has tools and apparatus for measuring atmospheric conditions, which it uses to study the weather and climate and to give data for weather forecasts. ESP32 can measure the usual weather parameters; like temperature, humidity, air pressure, wind direction and speed. For this, the sensor should be connected externally.

### 2.1 Esp32 Wi-Fi Bluetooth module

The chip was created by Espressif Systems and is known as ESP32. This gives embedded devices connectivity for Wi-Fi and, in some models, dual-mode Bluetooth. The manufacturer frequently refers to modules and development boards that contain this chip as "ESP32" even though it is only the chip in theory.



*Figure 1 ESP32 Wi-Fi Module*

(Source:[https://www.google.com/search?q=esp32+module&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiMgMyqz5v6AhWnx3MBHfnfAcwQ\\_AUIBigB&biw=995&bih=422](https://www.google.com/search?q=esp32+module&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiMgMyqz5v6AhWnx3MBHfnfAcwQ_AUIBigB&biw=995&bih=422))

In ESP32 used "Tensilica Xtensa LX6" microprocessor. There are filters, power amplifiers, low-noise receive amplifiers. The ESP32 as a standalone chip or as a full featured development board.

The ESP32 is more powerful and faster than Arduino. It has a 32-bit microcontroller and 10 internal capacitive touch sensors. Specific static pins are designated for the ADC (analog to digital converter) and DAC (digital to analog converter) functionalities. The pins that will be used for UART, I2C, SPI, PWM, etc. can be chosen; all you need to do is designate them in the code. The multiplexing function of the ESP32 chip makes this possible.

#### 2.1.1 ESP32 power consumption

When not in use, the ESP32 can go into a power-saving mode called "sleep mode," which saves all data to RAM. Any unneeded peripherals are now turned off, and the RAM is given enough

power to maintain its contents. ESP32 has light sleep mode and deep sleep mode. ESP32 when the deep sleep mode it has a more power consumption than light sleep mode.

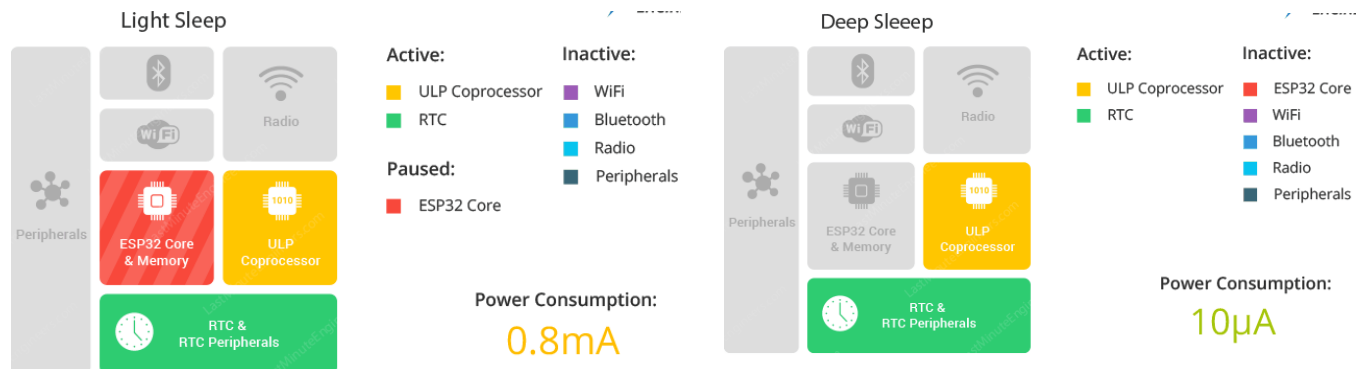


Figure 2: Comparison of light sleep mode and deep sleep mode of ESP32

(Source: [ESP32 Sleep Modes & Their Power Consumption lastminuteengineers.com](https://lastminuteengineers.com/esp32-sleep-modes-their-power-consumption/))

## 2.1.2 ESP32 standalone

ESP32 can function as a full standalone system or as a slave device to a host MCU, which lessens the burden on the primary application CPU caused by communication stack overhead.

## 2.2 DHT11 Sensor

The DHT11 sensor is a basic low budget temperature and humidity sensor. It can measure the humidity in the air using a thermistor and a capacitive humidity sensor and it outputs a digital signal on the data pin, there is no analog input pins needed. Although reasonably easy to operate, data collection requires precise timing.

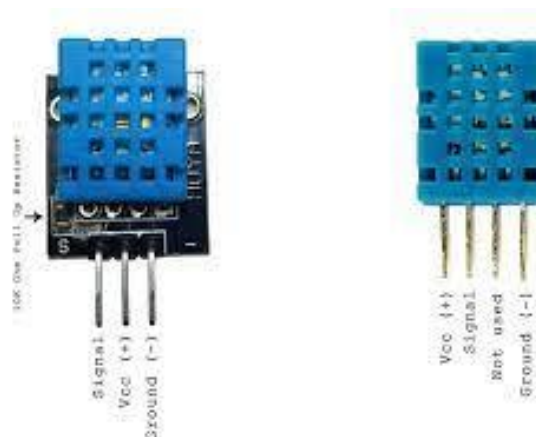


Figure 3: DHT11 sensor

(Source: [https://www.google.com/search?q=dht11+sensor&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiW4\\_jfnZ76AhVb63MBHTAHDCMQ\\_AUIBigB&biw=1349&bih=659](https://www.google.com/search?q=dht11+sensor&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiW4_jfnZ76AhVb63MBHTAHDCMQ_AUIBigB&biw=1349&bih=659))



The DHT11 uses a 3-5.5V DC power supply. To get over the unstable status after the sensor receives power, don't send it any instructions for one second. DHT11 can measure 0 to 50 Celsius temperature range and 20% to 90% humidity range. There are  $\pm 1$  °C and  $\pm 1\%$  accuracy.

## 2.3 BME180 Pressure sensor



Figure 4: BMP180 sensor

(source: [bmp180 - Google Search](#))

A pressure and temperature measurement sensor is the BMP180. It can monitor temperatures between -40°C and 85°C and pressures between 300 and 1100 hPa. It is a cheap sensor that runs on DC power between 3.3 and 5 volts.

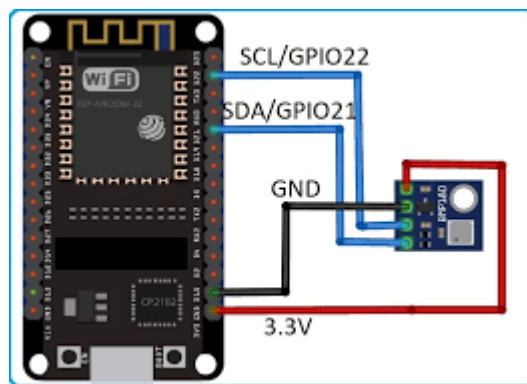


Figure 5: How to connect BMP180 with ESP32

## 2.4 LDR sensor

LDR means Light Dependent Resistor. LDR is made from a piece of exposed semiconductor material. This sensor is a light-sensitive device that is often used to indicate the presence or absence of light or to measure the intensity of light.



*Figure 6: LDR sensor*

(Source: [ldr sensor - Google Search](#))

This resistor can vary from about 100 Ohm in the sun light, to over 10 Mega Ohm in absolute darkness. Alarm clocks, street lights, light intensity meters are some applications of LDR sensor.

## 3 METHODOLOGY

### Individual Project

#### 3.1 PCB design and making

First PCB was designed using EasyEDA software. Additionally, it is processed using sensors.

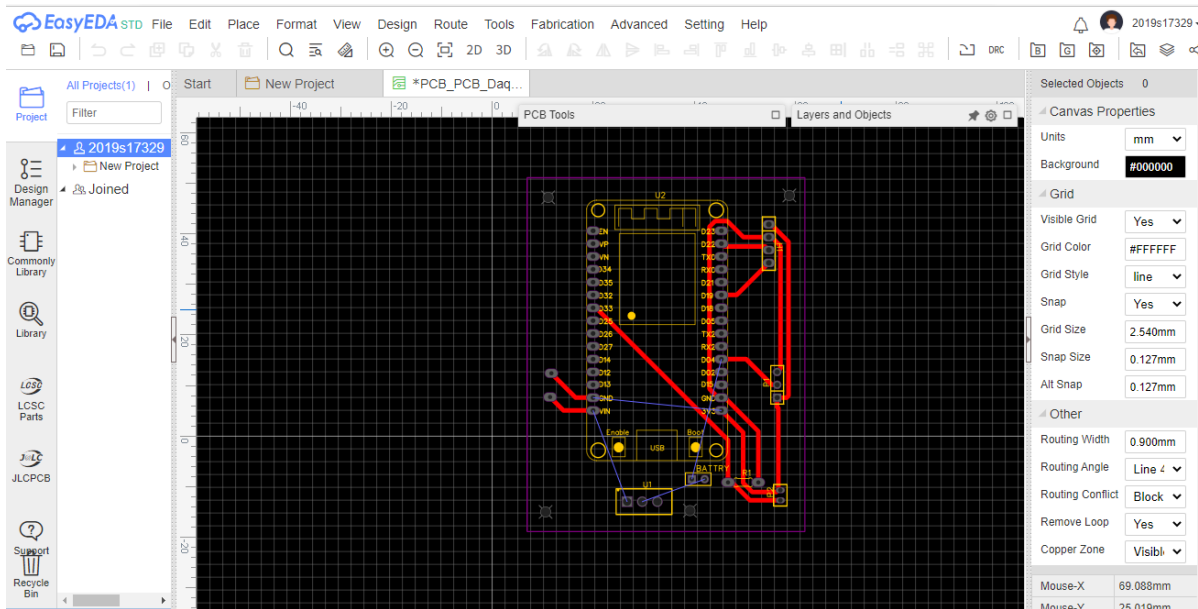


Figure 7: PCB design using EasyEDA software

Routing the whole path of the PCB and download pdf of circuit. Then PCB was marked according to the relevant process.

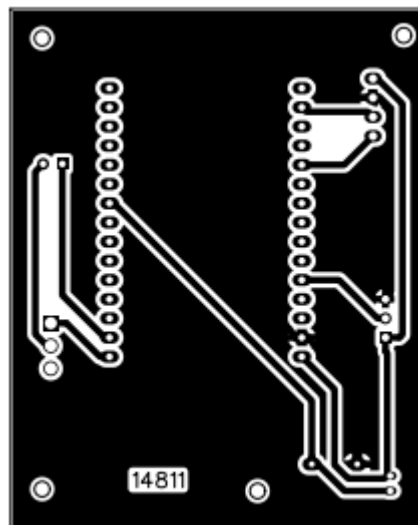


Figure 8: Structure of the final PCB design



Figure 9: Real PCB marking

DHT11 sensor, BMP180 sensor and LDR sensor were attached to the PCB. Also attached is an 18650 (3.7 V) battery. The switch was set to control on/off of the battery.

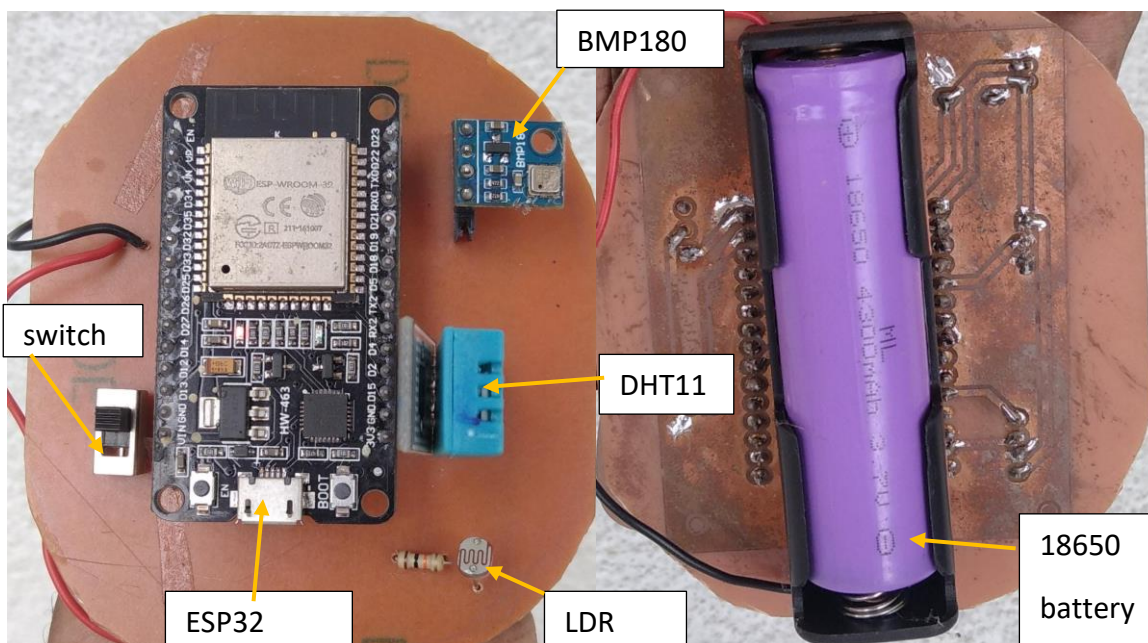


Figure 10: Adding the components to the PCB

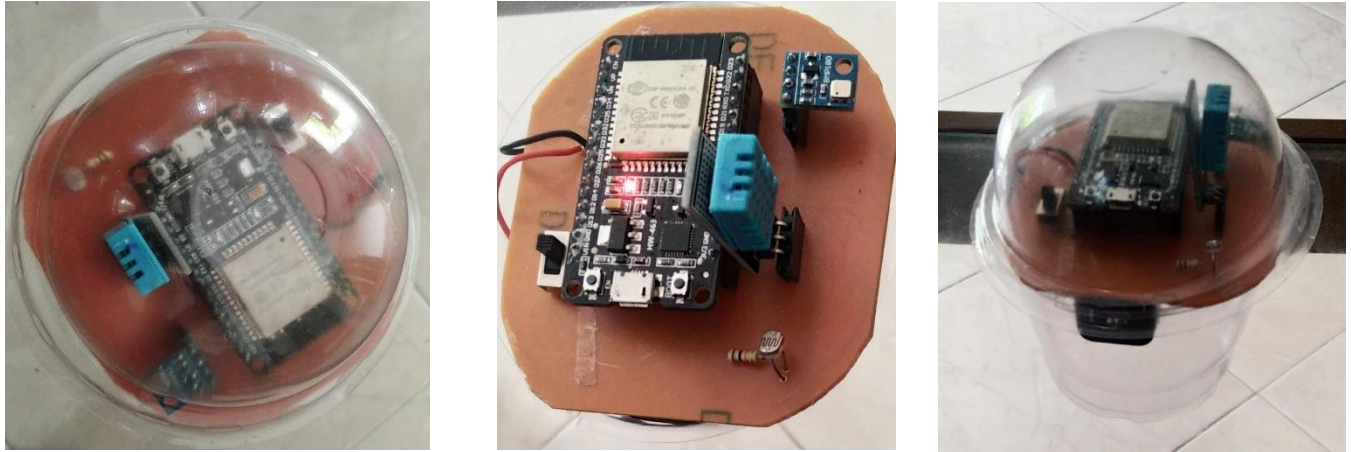


Figure 11: Final look of the weather station

### 3.2 Installing the ESP32 in Arduino IDE

In this experiment was used ESP32 microcontroller. The ESP32 can be programmed with the Arduino IDE and its programming language using an add-on for that software. Before installing ESP32 wants to update to the latest version of the Arduino IDE. Then open the Arduino IDE and go to File, Preferences.

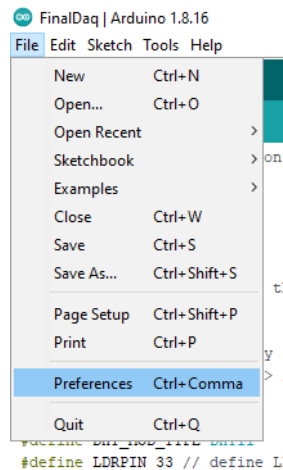


Figure 12: open the Arduino IDE and go to file preferences

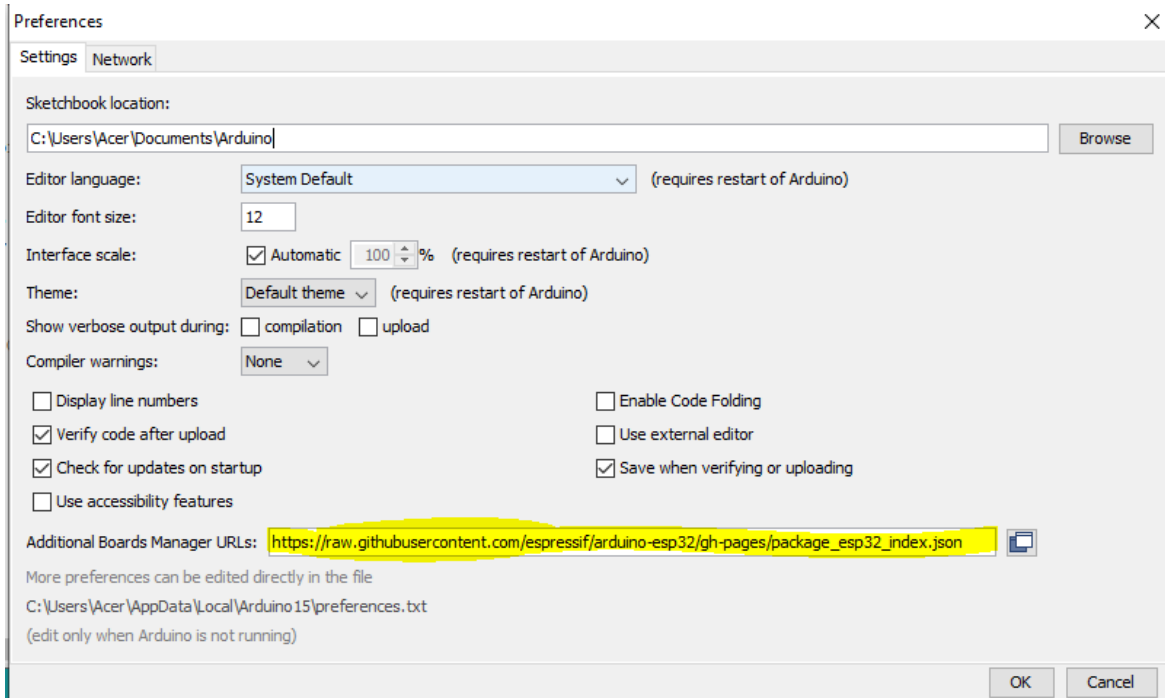


Figure 13: Enter the additional Board Manager URLs

In Figure 6 highlight part was shown the json URL. Then click the tools in action bar was selected the Boards Manager. And also selected the board as DOIT ESP 32 DEVKIT V1.

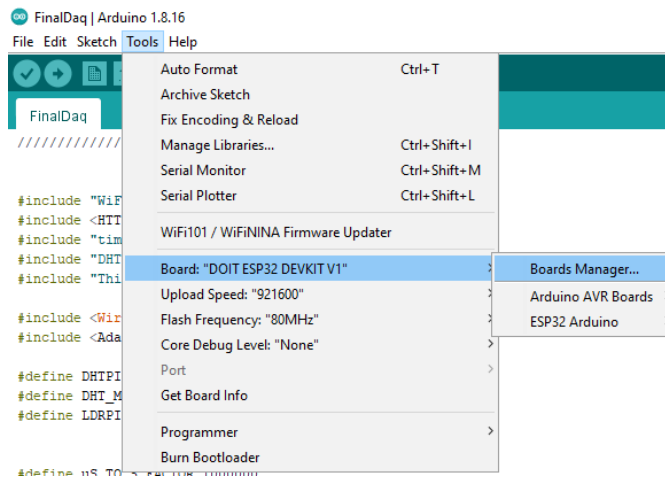


Figure 14: Selected the Boards Manager

Select the Boards manager and search ESP32 in the search box. Then installed the ESP32 by Espressif Systems.

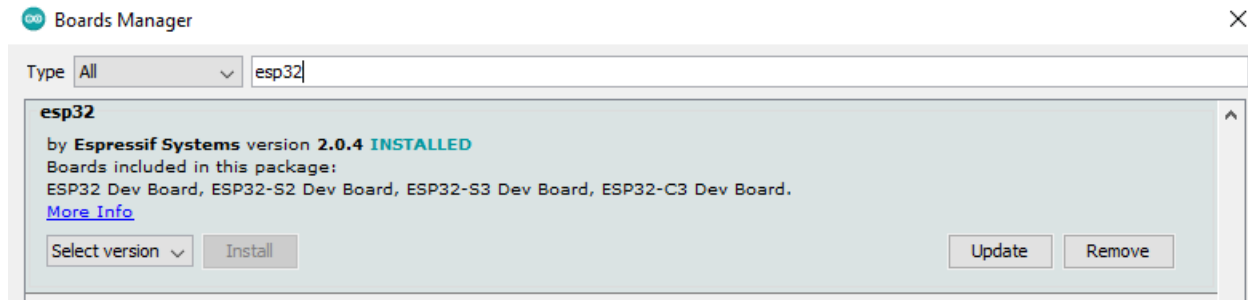


Figure 15: Installed the ESP32 Systems

### 3.3 Creating Google Sheet and Data Logger

In here, was used IFTTT platform. IFTTT is the web base service. IFTTT means If This, Then, That service. Its purpose is to connect disparate services and systems. Users can automate web-based processes and increase productivity with the use of this free web application. IFTTT links diverse developers' hardware, software, and applications to produce "applets" that carry out automations.

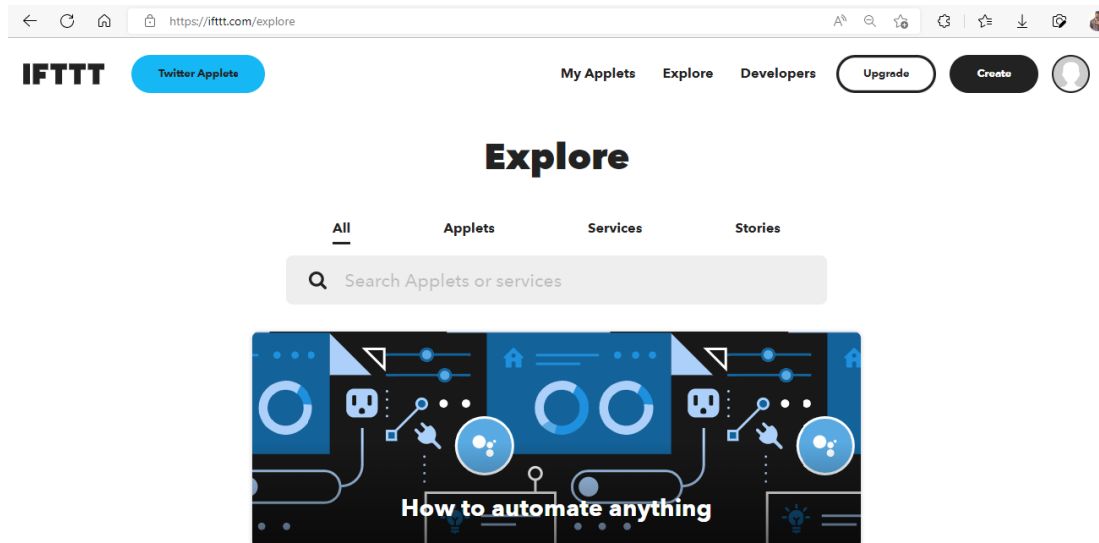


Figure 16: IFTTT web service interface

Press the create button and proceed. Then the applet can be seen as 'if this then that'. They should be filled one by one.

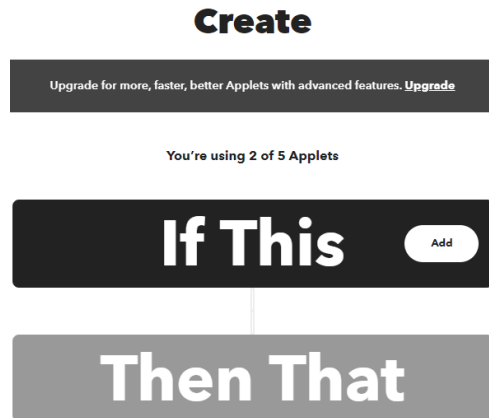


Figure 17: If This Then That applets

Then click the add button can be seen choose a service. Webhooks was searched the search bar and clicked it.

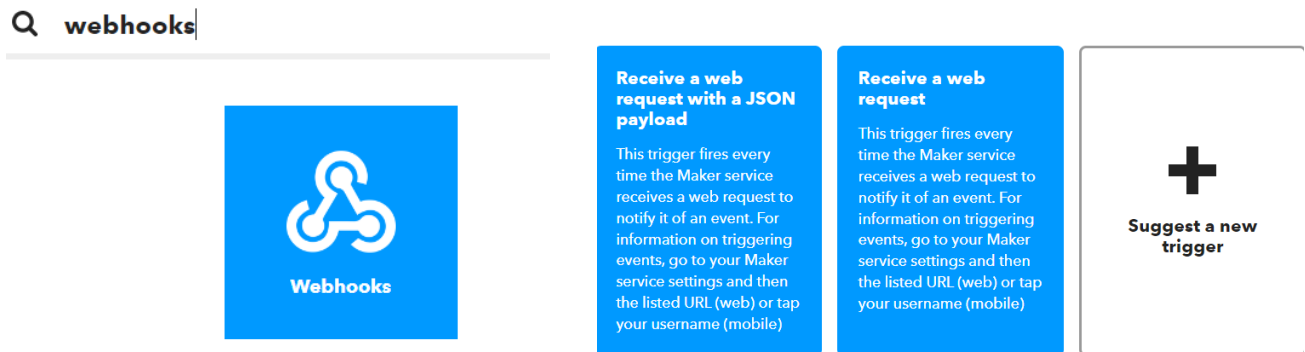


Figure 18: Search the Webhooks service and receive a web request

After the entering the Webhooks choose the receive a web request. Event Name was given in next page and create a new trigger.

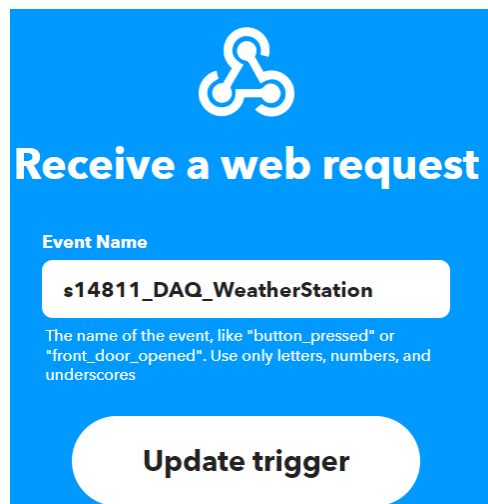


Figure 19: Create a new trigger in Webhooks



After that choose the google sheet service and select the 'add row to spreadsheet option'. This action will add a single row to the bottom of the first worksheet of a spreadsheet.

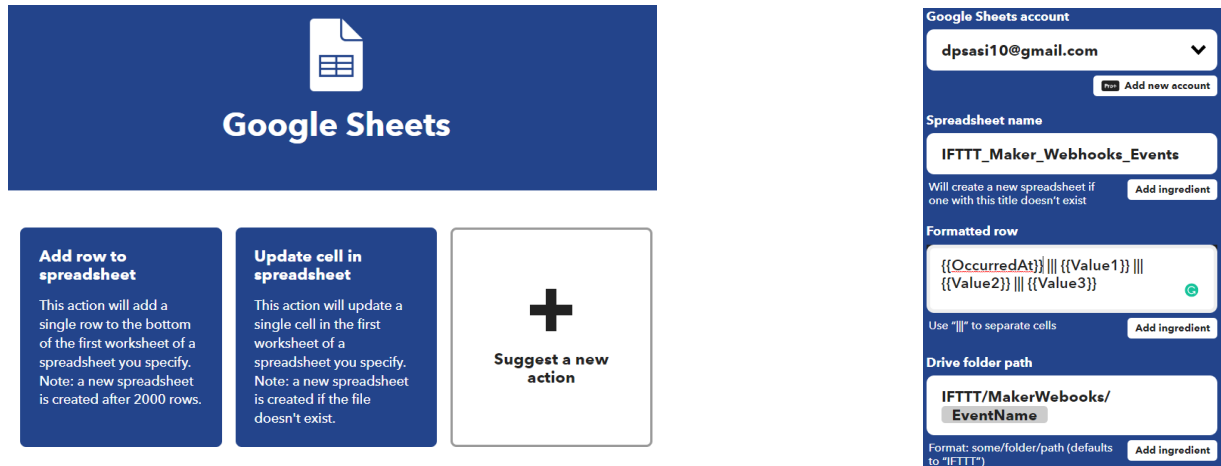


Figure 20: Choose the Add row to spreadsheet option

More details can be seen after selecting Add row option. Here you can see the details of the Account, spreadsheet name, formatted rows and Drive Folder path, where the google sheet is created. They can also be changed if needed.

When logged into the relevant google drive, a google sheet with the event name has been created.

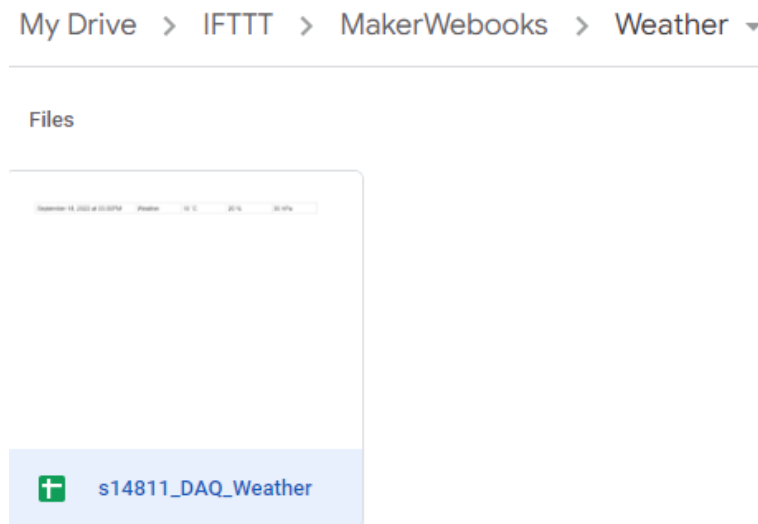


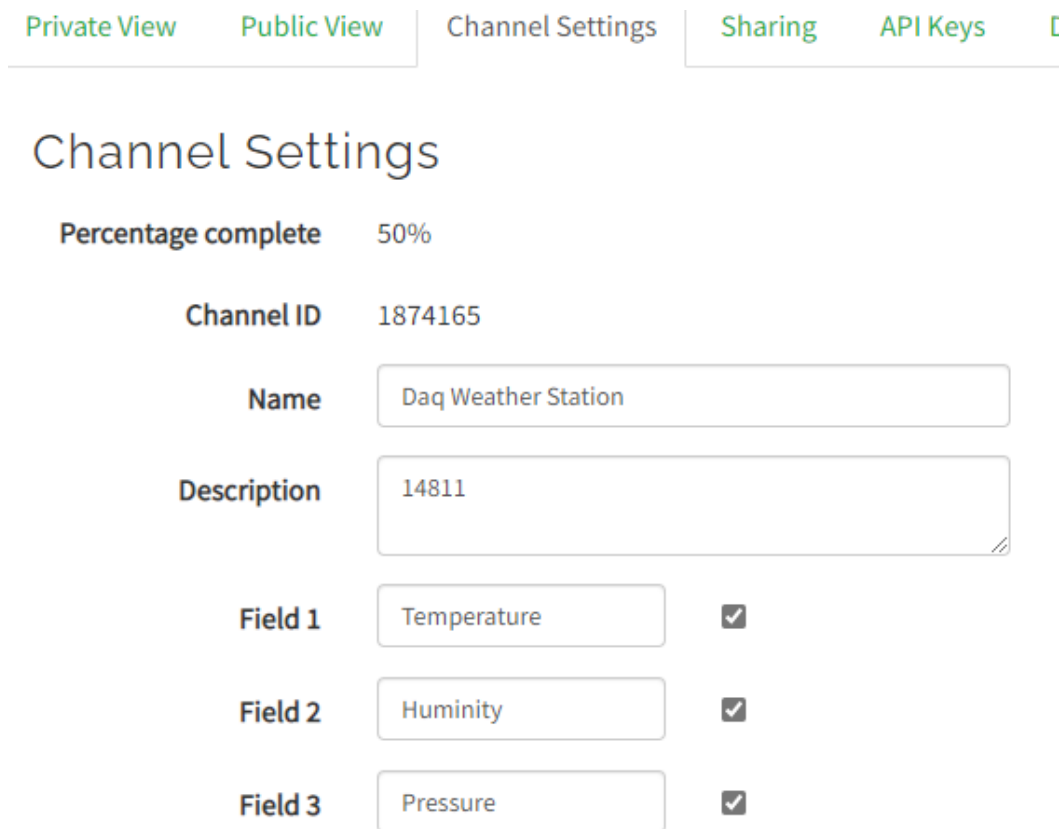
Figure 21: Google Sheet create in the google drive within event name

A test run can be done to see if the data goes to the Google Sheet.

### 3.4 Data Visualization using ThingSpeak

Arduino IDE serial monitor was used to data visualization. The best observation can be obtained from the 'Thinkspeak' online, even if it is possible to represent the serial monitor. The web browser was opened and searched "Thingspeak" channels.

First haven't any ThingSpeak account create account using Gmail. Then create new channel and the "Name" column was then filled with "DAQ Weather Station." Then "Temperature" was entered in "Field 1." Then the other field for "Humidity," "Pressure," and "Light Intensity" was filled in in the same manner. The "Save Channel" button was clicked after the user-required form had been filled out. The new channel was then developed.



Private View   Public View   Channel Settings   Sharing   API Keys   [

## Channel Settings

Percentage complete   50%

Channel ID   1874165

Name   Daq Weather Station

Description   14811

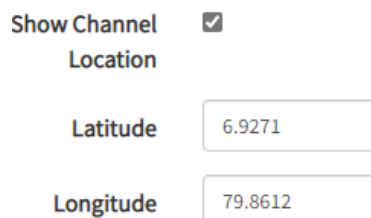
Field 1   Temperature   ☒

Field 2   Humidity   ☒

Field 3   Pressure   ☒

Figure 22: "ThingSpeak" channel setting

For show channel location we can give Latitude and Longitude in ThingSpeak.



Show Channel Location   ☒

Latitude   6.9271

Longitude   79.8612

Figure 23: Given Latitude and Longitude of station location

# Daq Weather Station

Channel ID: 1874165

14811

Author: mwa0000027684023

Access: Private

[Private View](#)

[Public View](#)

[Channel Settings](#)

[Sharing](#)

[API Keys](#)

## Write API Key

Key

9YS1LMH6JYV6NYGC

*Figure 24: API Keys and channel ID of ThingSpeak*

ThingSpeak website and Arduino IDE can be connected using "API key" and "channel ID". Finally, we can come to some conclusions after studying the dashboards of ThingSpeak.

### Group Project

The similar process was used for the group project. Data from every member was uploaded to a single Google Sheet that included a number for each weather station. Weather information was gathered nationwide while everyone was at their homes. It was decided to utilize just DHT11 sensor for each weather station and push only temperature and humidity to the google sheet because the sensors used varied from member to member.

Then, Grafana Dashboard was used to integrate all of the group members' data, making data analysis simple. With the help of charts and graphs on a single dashboard or across numerous dashboards, users of the user-friendly data visualization platform Grafana may examine data.

## 4 RESULTS AND ANALYSIS

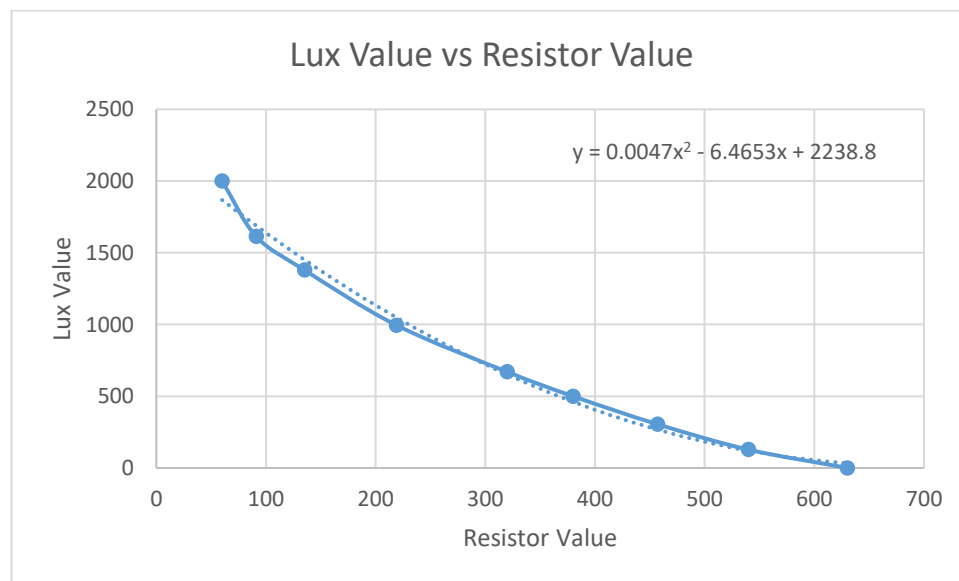
### Individual Project

#### 4.1 LDR sensor calibration

Lux meter was used for get corresponding resistance value of LDR sensor in different light intensity. Then Excel was used to obtained the graph of lux value and resistor value.

*Table 1: Lux values with ADC values(resistance) for LDR calibration*

Lux Value	Resistance Value
0.6	630
128	540
305	457
500	380
670	320
995	219
1380	135
1615	91
2000	60



*Figure 25: Graph of Lux value vs Resistor value of LDR in different light intensity*

According to the above graph when low light intensity resistor value was increase and high light intensity resistor value was become decrease. The graph was given the equation for the calibrate LDR sensor.

$$y = 0.0047x^2 - 6.4653x + 2238.8 \text{ --- } \rightarrow \text{ (Equation 01)}$$

## 4.2 Data logger and analyze

In this experiment was used ESP32 microcontroller. Weather station update every 60 seconds in the google sheet. ESP32 goes into deep sleep mode to avoid power loss between two modes because it records the weather only once every 60 seconds. Power consumption is very important because the weather station receives power from the battery. By saving power, you can easily get a day's worth of data from a fully charged battery.

Wi-Fi was enabled, the weather station was powered on, and data transmission was permitted. Data on temperature, humidity, light intensity and atmospheric pressure has been sent to a Google Sheet.

*Table 2: Enter the data using WIFI to the Google Sheet*

Date	Time	Temperature(*C)	Humidity (%)	Light Intensity	Pressure (Pa)
9/27/2022	10.51.34	32.7	70	585.13	100825
9/27/2022	10.52.20	32.7	70	590.3	100827
9/27/2022	10.53.07	32.7	70	590.3	100829
9/27/2022	10.54.00	32.7	69	590.3	100824
9/27/2022	10.54.46	32.6	69	509.58	100823
9/27/2022	10.55.19	32.7	75	464.58	100819
9/27/2022	10.56.35	32.3	69	581.84	100814
9/27/2022	10.58.18	32.8	68	590.3	100806
9/27/2022	11.03.05	32.7	69	590.3	100811
9/27/2022	11.03.59	32.7	69	590.3	100809
9/27/2022	11.04.45	32.7	69	590.3	100814
9/27/2022	11.05.30	32.7	68	590.3	100806
9/27/2022	11.06.14	32.7	69	590.3	100809
9/27/2022	11.07.00	32.1	69	590.3	100804
9/27/2022	11.07.58	32.2	69	590.3	100811
9/27/2022	11.41.56	32.2	77	590.3	100762
9/27/2022	11.42.39	32.7	71	590.3	100766
9/27/2022	11.43.26	32.7	69	590.3	100767
9/27/2022	11.44.10	32.7	69	590.3	100763

The data obtained by the weather station was taken to "ThingSpeak" and data visualization was done from it.

ThingSpeak retrieves the latitude and longitude of the location where the data is received.

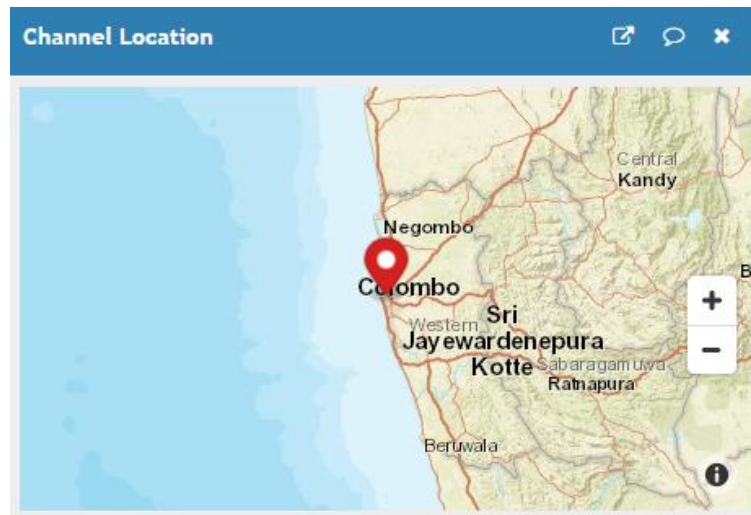


Figure 26: ThingSpeak get Latitude and longitude of the weather station location

Data were taken when weather station connected to the WIFI. Temperature was displayed in ThingSpeak dashboard as below.

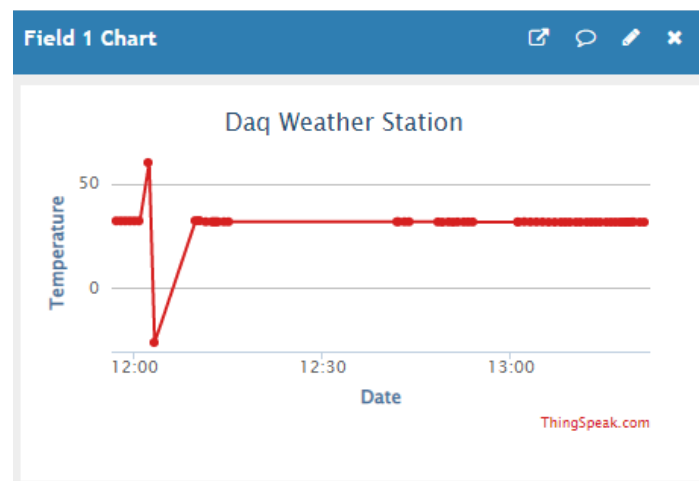


Figure 27: Temperature graph of ThingSpeak

There are some unusal points. It may be happened because poor WIFI connection. Almost all the time it has not much variation.

Widgets can be added in ThingSpeak and below gauge shown the average temperature in data collecting period. Its value is 31.6 Celsius.

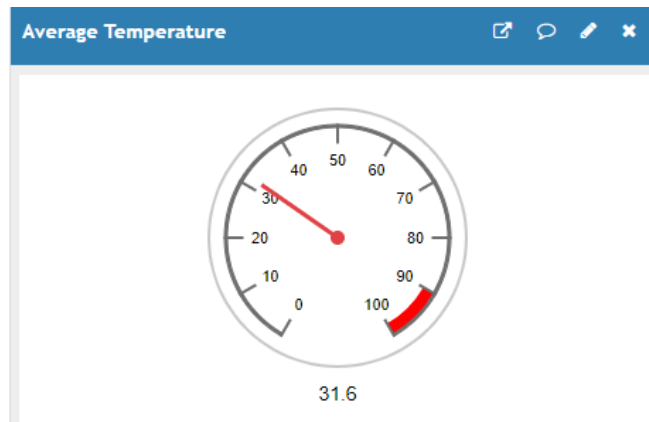


Figure 28: Average temperature in Data collecting period

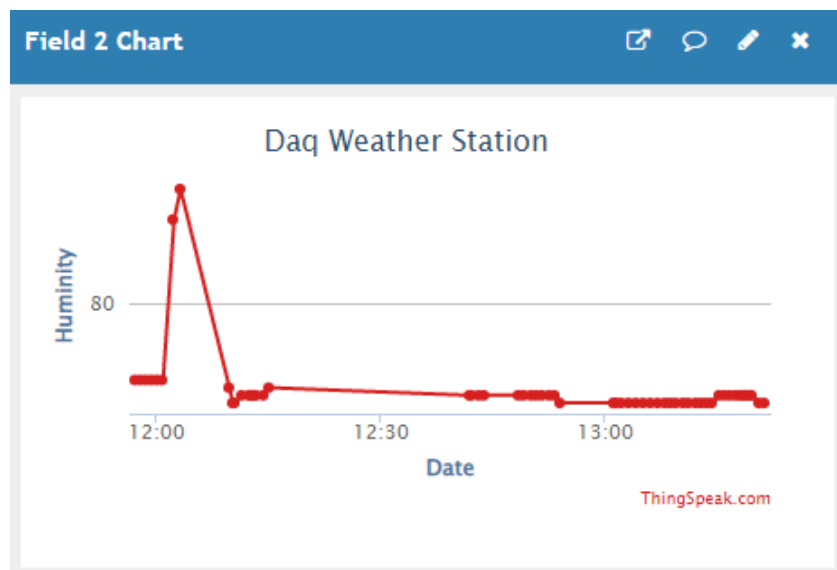


Figure 29: Humidity variation dashboard in ThingSpeak

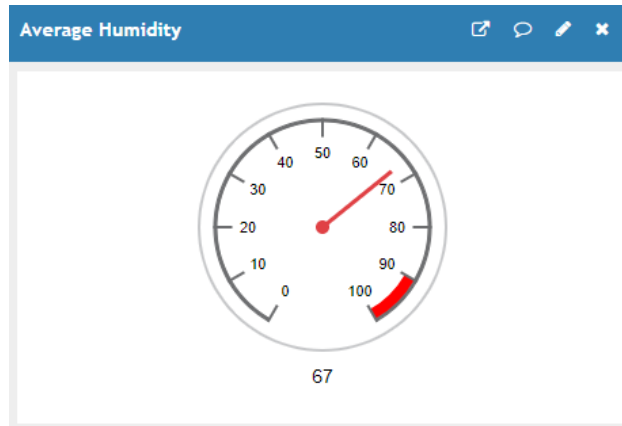


Figure 30: Average humidity value in data collecting period

The pressure that BMP180 measured is shown below. There are a few oddities. As a result, the average pressure may be calculated incorrectly. The average pressure is 100600 Pa after the odd data is removed from the average. That figure is quite precise.

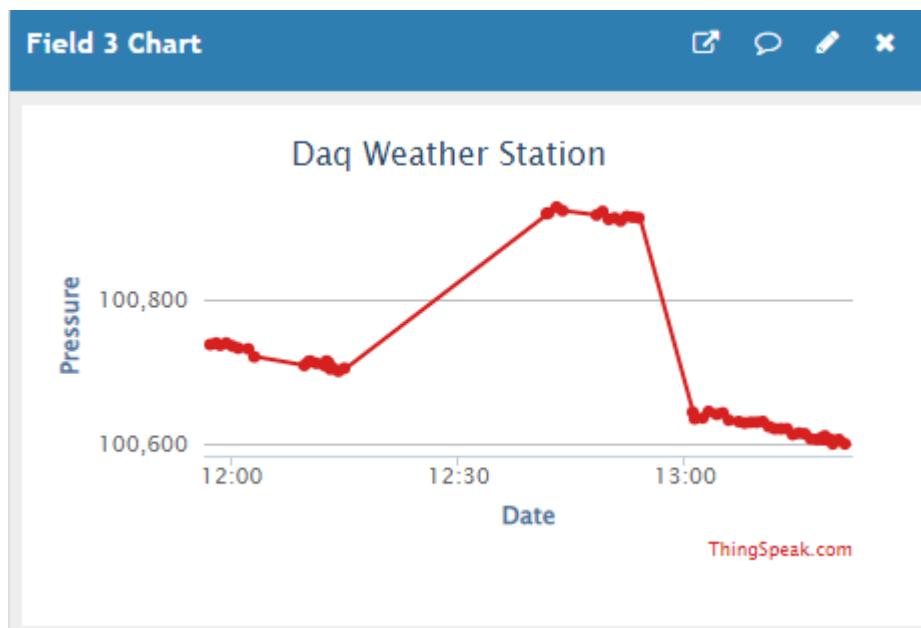


Figure 31: Pressure variation in ThingSpeak



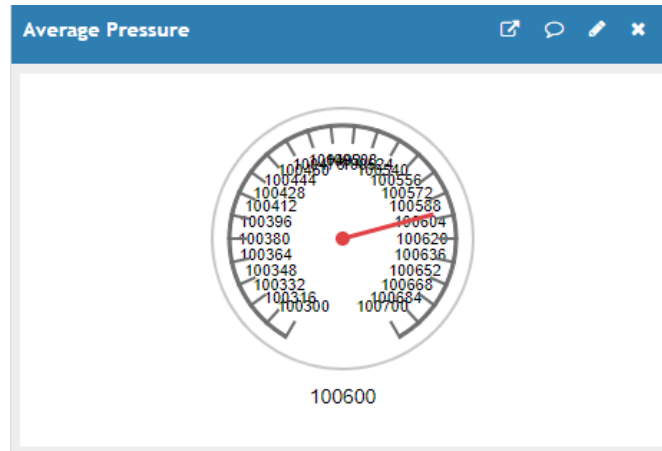


Figure 32: Average Pressure in data collecting period

## Group Project

Temperature and humidity were collected by all the group members in various places in the Sri Lanka. Therefore, used only DHT11 sensor. Then all the data analyzed using Grafana software.

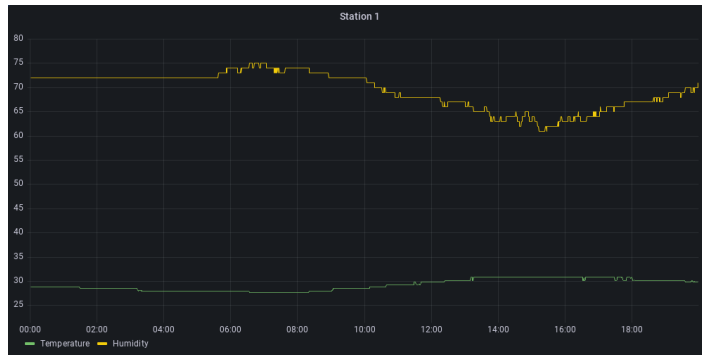


Figure 31: Station 1 & 2 Temperature and Humidity graphs



Figure 332: Station 3 & 4 Temperature and Humidity graphs

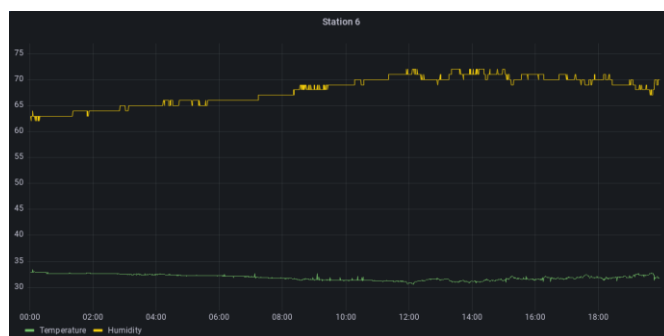


Figure 34: Station 5 & 6 Temperature and Humidity graphs

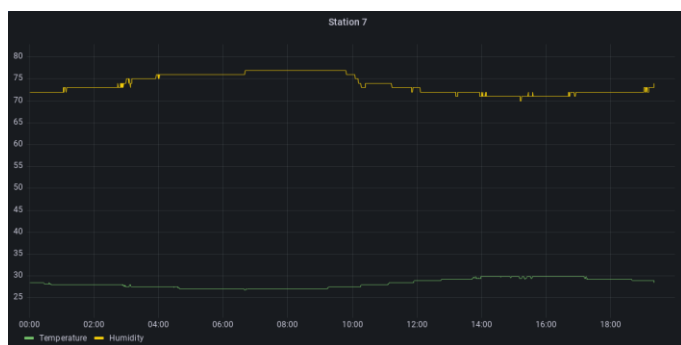


Figure 35: Station 7 & 8 Temperature and Humidity graphs

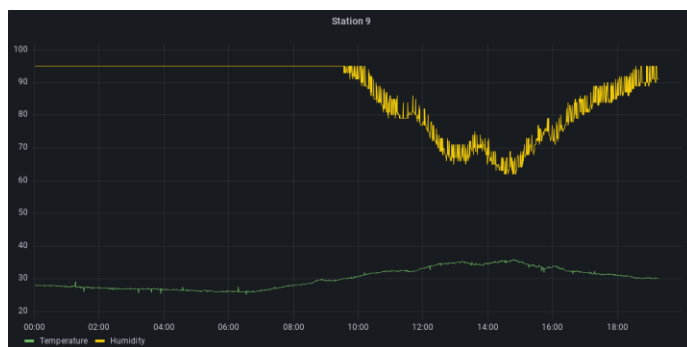


Figure 36: Station 9 & 10 Temperature and Humidity graphs

## 5 DISCUSSION

For this weather station project circuit and control have been made with ESP32 microcontroller and control all weather parameter sensors BMP180, DHT11 for pressure and temperature, humidity respectively. Arduino IDE was used to write the code and upload to the ESP32 dev kit board. ESP32 connect with the Wi-Fi and applied to the system. So, then web server can be created display the data that obtained from sensor and logged them to Google sheet. Thingspeak website also has been used for log data and Thingspeak stored data and display that data created channel on that website. Data has collected from ESP32 MCU. The analysis of the data has been made easy with Thingspeak website due to it interface shows with all the time. Considering power saving ESP works with deep-sleep mode and once they get data sleep for 30 seconds and with the analysis seems to be there is no such variation of the data so that if set sleep ESP32 more than 30 seconds can be saved power. By increasing the time of deep-sleep not only save the power but also can be saved the memory of ESP32 and it useful when connection is lost because frequency of data will be increased that can store in the ESP32. Another problem was faced, ESP32 was slept but all the sensor consumed the power. To solve that problem, transistor can be attached to the circuit work as a switch when ESP sleeping all sensor's input pin move to low using that transistor.

## 6 CONCLUSION

The ESP32 board is a versatile and very affordable development platform that could be easily programmed to create cloud base data acquisition setups and using most of the functions the power consumption of the device is reduced. The weather station system is built around the steps of direct environmental sensing, data measurement and storage, and allowing users to access the results through the cloud. This is built around a collection of embedded sensors, and a microcontroller serving as the system's core and server, and wireless internet communication protocol. It is very easy to use as it can be kept in a small space. The 4 sleep modes of this microcontroller help to improve the mobility of the setup and to reduce the power consumption. The observed real time data is stored on the ThingSpeak server which can be accessed globally. The different values of each environmental factor at different intervals in time are also observed and the observed result clearly shows the changes in the weather conditions for a full day cycle. Most of the time the sensors have given reliable data but sometimes it can be unusual due to poor connection.

## 7 References

Anon., 2022. *randomnerdtutorials.com*. [Online]

Available at: <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

[Accessed 01 October 2022].

Anon., n.d. *iotdesignpro.com*. [Online]

Available at: <https://iotdesignpro.com/articles/esp32-data-logging-to-google-sheets-with-google-scripts>

[Accessed 22 April 2022].

Fahad, E., 2021. [Online]

Available at: <https://www.electronicclinic.com/deep-sleep-modes-in-esp32-and-how-to-use-them/>

## APPENDIX

```
//////////Weather Station//////////
////////// s14811 //////////

#include "WiFi.h"           //Library add for WiFi connection
#include <HTTPClient.h>
#include "time.h"
#include "DHT.h"           //Library add for DHT11 sensor
#include "ThingSpeak.h"    //Library add for ThingSpeak service
#include "SPIFFS.h"
#include <vector>

#include <Wire.h>           //Library add for BMP180 Sensor
#include <Adafruit_BMP085.h>

#define PIN_DHT 4
#define PIN_MODE_DHT DHT11
#define PIN_LDR 33        // define LDR pin is GPIO33

//Deep Sleep ESP32 for 60 seconds
#define uS_TO_S_FACTOR 1000000
#define TIME_TO_SLEEP 60

RTC_DATA_ATTR int bootCount = 0;

const char* ntpServer = "pool.ntp.org";
const long  gmtoffset_sec = 19800;
const int   daylightOffset_sec = 0;

//WIFI details ssid & password
const char * ssid = "Dialog 4G 640";
const char * password = "1d90cB6c";

//IFTTT server connection
String server = "http://maker.ifttt.com";
String eventName = "s14811_DAQ_WeatherStation";
String IFTTT_Key = "H-RtTi-Zyflg7UL-Nqp4f";
String
IFTTUrl="https://maker.ifttt.com/trigger/{sensor_data}/json/with/key/H-
RtTi-Zyflg7UL-Nqp4f";

//ThingSpeak channel ID and API key
unsigned long chanelId = 1874165;
const char *rightAPIkey = "9YS1LMH6JYV6NYGC";
WiFiClient client;
```

```

// Google script ID and required
String GOOGLE_SCRIPT_ID =
"AKfycbwNEDXXE8xbOujzaZEj9nJjcAPhKUj3ywCQswzN7QvMzBRRNNQ5p1VLlUyNxKo99UtA";
// change Gscript ID
using namespace std; // "SPIFFS.h library
int count = 0;

DHT dht11(PIN_DHT, PIN_MODE_DHT);

Adafruit_BMP085 bmp;

void print_wakeup_reason() {
    esp_sleep_wakeup_cause_t wakeup_reason;

    wakeup_reason = esp_sleep_get_wakeup_cause();

    switch(wakeup_reason)
    {
        case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused -->
RTC_IO"); break;
        case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused -->
RTC_CNTL"); break;
        case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused --> timer");
break;
        case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused -->
touchpad"); break;
        case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused --> ULP
program"); break;
        default : Serial.printf("Wakeup was not caused by deep sleep:
%d\n", wakeup_reason); break;
    }
}

void listAllFiles() {
    File root = SPIFFS.open("/");
    File file = root.openNextFile();
    while (file) {
        Serial.print("FILE: ");
        Serial.println(file.name());
        file = root.openNextFile();
    }
}

void setup() {
    delay(1000);
    Serial.begin(115200);
    delay(1000);
    SPIFFS.begin(true);
    File tempfile;
    File humidityfile;

```

```

char firstVariable = 0;
char secondVariable = 0;
char thirdVariable = 0;
char timeVariable=0;
char StartupEmptypoint;
char number = 80;
char WifiConnectWaitingTime=20;
String temparray[number];
String humidarray[number];

delay(1000);
dht11.begin();
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

if(!bmp.begin()){
    while(1){}
}

Serial.println();
Serial.print("Connecting to wifi: ");
Serial.println(ssid);
Serial.flush();
WiFi.begin(ssid, password);

    float dhtHumidity = dht11.readHumidity();
    float dhtTemperature = dht11.readTemperature();

while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
    thirdVariable++;
    if (thirdVariable > WifiConnectWaitingTime) {
        Serial.println("\ndata recording Offline...");
        if(!(SPIFFS.exists("/tempc.txt")) &&
!(SPIFFS.exists("/humidity.txt"))){
            File tempfile = SPIFFS.open("/tempc.txt", FILE_WRITE);
            File humidityfile = SPIFFS.open("/humidity.txt", FILE_WRITE);

            Serial.println("file write in write mode");
            tempfile.println(String(dhtTemperature));
            humidityfile.println(String(dhtHumidity));

            tempfile.close();
            humidityfile.close();
        }else{
            File tempfilea = SPIFFS.open("/tempc.txt", "a" );
            File humidityfilea = SPIFFS.open("/humidity.txt", "a");
            tempfilea.println(String(dhtTemperature)); // log temperature to
tempfile
            humidityfilea.println(String(dhtHumidity)); //log humidity to
humidityfile
            Serial.println("file write in append mode");
            tempfilea.close();

```



```

        humidityfilea.close();
    }
    thirdVariable = 0;
    break;
}
}
++bootCount;
Serial.println("Boot number: " + String(bootCount));

//ESP32 wakeup reason
print_wakeup_reason();

esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
" Seconds");

ThingSpeak.begin(client);
}

void loop() {
    delay(2000);
    float dhtHumidity = dht11.readHumidity();
    float dhtTemperature = dht11.readTemperature();
    float bmpPressure = bmp.readPressure();
    char firstVariable = 0;
    char secondVariable = 0;
    char thirdVariable = 0;
    char timevariable=0;
    char StartupEmptypoint;
    char number = 100;
    char WifiConnectWaitingTime=25;
    String temparray[number];
    String humidarray[number];

    //check the any reads fail to read, then try again
    if(isnan(dhtHumidity) || isnan(dhtTemperature)){
        Serial.println(F("Failed read DHT11 sensor!!!"));
        return;
    }

    Serial.print(F("Humidity: "));
    Serial.println(dhtHumidity);
    Serial.print(F("Temperature: "));
    Serial.print(dhtTemperature);
    Serial.println(F("°C "));

    Serial.print("Pressure = ");
    Serial.print(bmpPressure);
    Serial.println(" Pa");

```

```

//LDR sensor data
int analogValueLDR = analogRead(PIN_LDR);
Serial.print("Analog value of LDR = ");
Serial.println(analogValueLDR);

//calibrated equation for analog to lux converter for LDR. EQ; y' =
0.0047x^2 - 6.4653x + 2238.8;
float luxvalue = (0.0047*(analogValueLDR*analogValueLDR)) -
(6.4653*analogValueLDR)+2238.8;
Serial.println(luxvalue);

ThingSpeak.setField(1,dhtTemperature);
ThingSpeak.setField(2,dhtHumidity);
ThingSpeak.setField(3,bmpPressure);

int respons = ThingSpeak.writeFields(chanelId,rightAPIkey);

if (respons == 200) {
    Serial.println("Data upload sucessful");
} else {
    Serial.println("Data upload unsucessful");
}

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

    if ((SPIFFS.exists("/tempc.txt")) &&
(SPIFFS.exists("/humidity.txt"))) {

        File tempfile1 = SPIFFS.open("/tempc.txt");
        File humidityfile1 = SPIFFS.open("/humidity.txt");
        vector<String> v1;
        vector<String> v2;

        while (tempfile1.available()) {
            v1.push_back(tempfile1.readStringUntil('\n'));
        }

        while (humidityfile1.available()) {
            v2.push_back(humidityfile1.readStringUntil('\n'));
        }

        tempfile1.close();
        humidityfile1.close();

        for (String s1 : v1) {

            temparray[firstVariable] = s1;
            firstVariable++;
        }
    }
}

```

```

    for (String s2 : v2) {

        humidarray[secondVariable] = s2;
        secondVariable++;
    }
    while (timeVariable <= number) {
        if(temparray[timeVariable]==0 && humidarray[timeVariable]==0){
            Serial.println("\nArrays are empty..");
            StartupEmptypoint=timeVariable;
            break;
        }else{
            if (WiFi.status() == WL_CONNECTED) {
                HTTPClient http;
                Serial.println("offline data uploading... ");
                Serial.print("temp :");
                Serial.print(temparray[timeVariable]);
                Serial.print("\t");
                Serial.print("humid :");
                Serial.print(humidarray[timeVariable]);
                Serial.print("\n");
                String url = "https://script.google.com/macros/s/" +
GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(temparray[t]) +
"&Humidity=" + String(humidarray[t]);

                Serial.println("Making a request");
                http.begin(url.c_str());
                http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
                int httpCode = http.GET();
                String payload;
                if (httpCode > 0) {
                    payload = http.getString();
                    Serial.println(httpCode);
                    Serial.println(payload);
                }
                else {
                    Serial.println("Error on HTTP request");
                }
                http.end();
            }

            timeVariable++;

        }
    }
    if (timeVariable == number || timeVariable==StartupEmptypoint) {

        listAllFiles();
        SPIFFS.remove("/tempc.txt");
        SPIFFS.remove("/humidity.txt");

        listAllFiles();
    }
}

```

```

        firstVariable = 0;
        secondVariable = 0;
    }
}
else{
    static bool flag = false;
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Failed to obtain time");
        return;
    }
    char timeStringBuff[50];
    strftime(timeStringBuff, sizeof(timeStringBuff), "%A, %B %d %Y
%H:%M:%S", &timeinfo);
    String asString(timeStringBuff);
    asString.replace(" ", "-");
    Serial.print("Time:");
    Serial.println(asString);
    String urlFinal = "https://script.google.com/macros/s/"+
GOOGLE_SCRIPT_ID +"/exec?"+"Temperature=" + String(dhtTemperature) +
"&Humidity=" + String(dhtHumidity) + "&LDR=" + String(luxvalue) +
"&Pressure=" + String(bmpPressure);
    Serial.print("POST data to spreadsheet:");
    Serial.println(urlFinal);
    HTTPClient http;
    http.begin(urlFinal.c_str());
    http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
    int httpCode = http.GET();
    Serial.print("HTTP Status Code: ");
    Serial.println(httpCode);

    String payload;
    if (httpCode > 0) {
        payload = http.getString();
        Serial.println("Payload: "+payload);
    }

    http.end();
}
}
WiFi.disconnect();

Serial.println("Disconnectiong the WiFi....");
delay(1000);

Serial.println("Sleep now");
delay(1000);
Serial.flush();
esp_deep_sleep_start();
}

```

