



**Project Supervisor: Dr. Rafia Mumtaz (Associate Professor)**

**HoD (Internet of Things)**

**Office No. A-203, Faculity Block SEECS, NUST, Islamabad**

**IoT Based Weather Station**



National University of Sciences & Technology

School of Electrical Engineering and Computer Science Department of Computing

**PROJECT DOCUMENTATION**



**Group Members**

Maasooma

429876

MS CS 2022

SEECS, NUST H12

maasooma.mscs22seecs@seecs.edu.pk

Ayesha Sana

402433

MS CS 2022

SEECS, NUST H12

ayeshasana.mscs22seecs@seecs.edu.pk

Tabinda Ghazal

402420

MS CS 2022

SEECS, NUST H12

tghazal.mscs22seecs@seecs.edu.pk

Saad Mehmood Mirza

400432

MS CS 2022

SEECS, NUST H12

smirza.mscs22seecs@seecs.edu.pk

**Table of Contents:**

1. **Chapter 1: Introduction** 6
   1. Background 7
   2. Objectives 7
   3. Scope 7
2. **Chapter 2: Application Requirement 8**
   1. Hardware Used 8
   2. **Hardware Specifications 9 2.2.1** NodeMCU ESP8266 Board 9

**2.2.2** BreadBoard 11

**2.2.2** BMP 80 Pressure Sensor 12

**2.2.4** DHT11 Sensor 13

**2.2.5** Rain Sensor 15

**2.2.6** Jumper Wires 17

**2.2.7** Resister 17

1. **Chapter 3: System Architecture 19**
   1. Overview of System Architecture 19
   2. Circuit Diagram 20
   3. Software Requirements 20
   4. Application Interface 24
   5. Communication Protocol 25
   6. Application Code 26
2. **Chapter 4: Testing & Visualization 33** 
   1. Testing 33
   2. Visualization 34
   3. Analysis 35
3. **Chapter 5: Conclusion and References 36**
   1. Conclusion 36
   2. References 36
4. **Github Link for Our Project 36**

**List of Figures**

**Figure 1 : Three Is of IBM**

**Figure 2 - NodeMCU Pinout**

**Figure 3: Bread Board with labels**

**Figure 4 BMP 80 Pressure Sensor**

**Figure 5 BMP 180 Sensor Pinout**

**Figure 6 DHT 11 Pinout**

**Figure 7 Rain Sensor**

**Figure 8 Rain Sensor Pinout**

**Figure 9 Rain Sensor Output Voltage**

**Figure 10 Jumper Wires**

**Figure 11 Architecture overview**

**Figure 12 Circuit Diagram**

**Figure 13: Arduino IDE Interface [15]**

**Figure 14 ThingSpeak Logi**

**Figure 15 ThingSpeak Channel Creation**

**Figure 16 ThingSpeak Field Adjustment for Sensors**

**Figure 17 Field Adjustments in thingSpeak**

**Figure 18 Initial Dashboard**

**Figure 19 Serial Monitor Output**

**Figure 14 Application Interface**

**Figure 15 ThingSpeak Dashboard**

**Figure 16 Sensor wise Dashboard**

**List of Tables**

**Table 1 - Details of Hardware Used**

**Table 2 - NodeMCU Specifications**

**Table 3 BMP 180 Sensor Pinout**

**Table 4 BMP 180 Sensor Specifications**

**Table 5 Rain Sensor Pinout**

**Table 6 Rain Sensor Specifications**

**Table 7 Technical Specifications of Jumper Wires**

**CHAPTER 1**

**INTRODUCTION**

1. **Background**

With the advent of high speed Internet, more and more humans around the globe are interconnected. Internet of Things (IoT) takes this a step further, and connects not only humans but electronic devices which can speak amongst themselves. The main concept behind the Internet of Things(IoT) is to connect various electronic devices through a network and then retrieve the data from these devices (sensors) which can be distributed in any fashion, upload them to any cloud service where one can analyze and process the gathered information.

According to the IBM concept, smarter planet is built on the following set of three pillars called the Three Is which is illustrated in fig:1. The first I stands for Instrumented which means through the use of remote sensor information is captured wherever it exists. The second I stand for Interconnected that means collected information is moved from one point to the other where it is useful. The third I stands for Intelligent which means information is processed, analysed, and acted upon it to get the knowledge. [1]

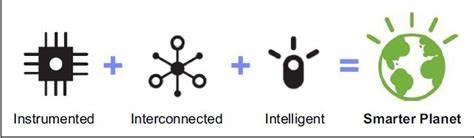


Fig 1 : Three Is of IBM

A weather station can be described as an instrument or device, which provides us with the information of the weather in our neighbouring environment. For example it can provide us with details about the surrounding temperature, barometric pressure, humidity, etc. Hence, this device basically senses the temperature, pressure, humidity, etc. It can be used to monitor the temperature or humidity of a particular room/place. [2]

The system uses NodeMCU 8266, which is the heart of the device. It provides the platform for IOT. It’s a wifi module having esp8266 firmware within. All the other sensors are connected to this micro-controler. They send the measuered values to it and it uploads all the values to the cloud where the values are analyzed. The developer of this board is ESP8266 Opensource Community. It has an operating system called XTOS. The CPU is ESP8266(LX106). It has an in-built memory of 128 KBytes and a storage capacity of 4 Mbytes.[3]

The system also includes BMP 180(Pressure Sensor) which senses the barometric pressure from the surrounding. BMP180 is an I2C standard device. Its a 4-pin device, viz, SDA, SCL, VIN, GND. Vin and GND are connected to 3.3V and GND respectively. SDA is connected to D2 pin of nodemcu and SCL is connected to D3 pin of nodemcu.[3]

For measuring of temperature, DHT-11 has been used. It senses the temperature of the surrounding. Its a 4-pin device. We should connect a 10k resistor between pin 1 and pin 2. Pin 1 is connected to the 3.3V. Pin 4 is connected to GND. Pin 2 is the output pin which gives input to the nodemcu pin D4. Pin 3 is left empty.[3]

1. **Objectives**

An IoT-based weather system aims to provide real-time monitoring formation about various weather parameters such as temperature, humidity, air pressure, wind speed, and precipitation. This data can be collected from a network of sensors deployed in different locations.By collecting and analyzing data from multiple sensors, an IoT-based weather system can help in weather forecasting and prediction. IoT-based weather systems can contribute to disaster management efforts by providing early warnings for severe weather conditions such as hurricanes, floods, or heatwaves.

1. **Scope**

The scope of this project is limited to the development of a local weather station using Usb link for communication between the user's device and the Node MCU presenting a real time weather status of the surrounding areas. The main objective of this paper is to view weather conditions of any location and allows to access the current data of any station.

**CHAPTER 2**

**APPLICATION REQUIREMENT**

**2.1 Hardware Components**

Details of hardware used for this project is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Hardware** | **Details** | **Hardware** | **Details** |
| NodeMCU ESP8266 Board |  | BMP180 Sensor |  |
| Rain Sensor FC-37 |  | DHT11 Sensor |  |
| Resister 4.7k |  | Connecting Wires |  |
| Breadboard |  |  |

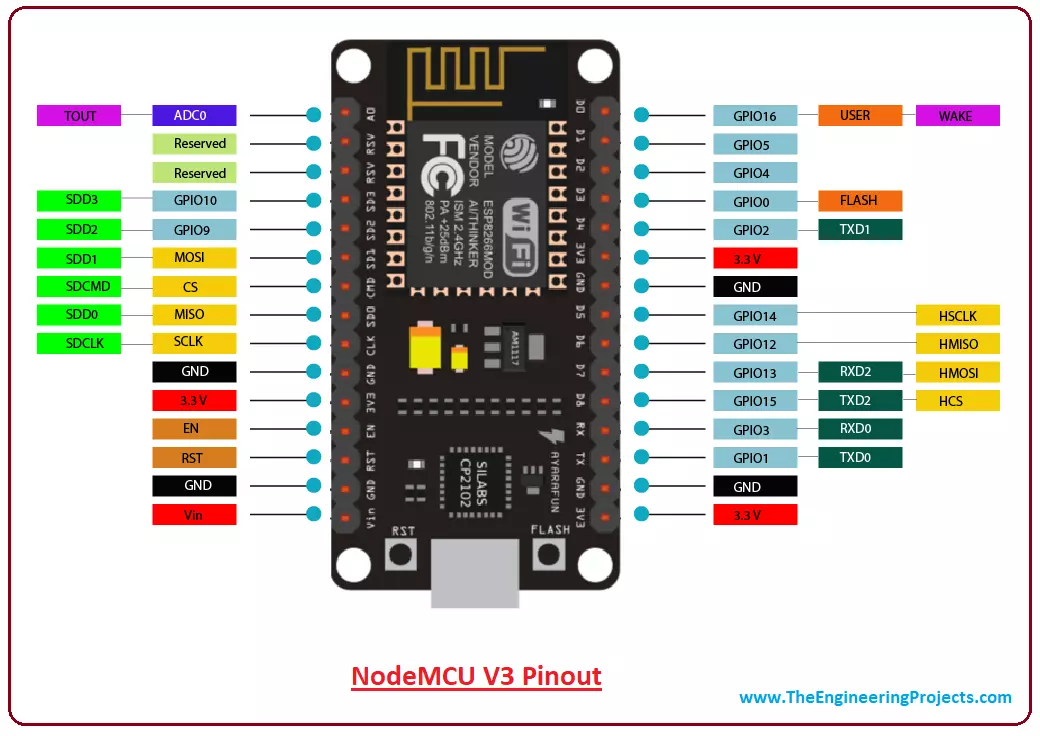
**Table 1 - Details of Hardware Used**

**2.2 Componets Used in the Project**

**2.2.1 NodeMCU ESP8266 Board**

The NodeMCU (**N**ode **M**icro**C**ontroller **U**nit) is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi). That makes it an excellent choice for Internet of Things (IoT) projects of all kinds. You have to program it in low-level machine instructions that can be interpreted by the chip hardware. The NodeMCU is available in various package styles.

For this project we are using Official Amica NodeMCU. NodeMCU V3 comes with a number of GPIO Pins. Following figure shows the Pinout of the board



**Fig 2 - NodeMCU Pinout**

Amica NodeMCU measures 49mm x 26mm with a standard pin space of 0.1″ between pins and 0.9″ between rows. The Amica NodeMCU is approximately 25% smaller in size than a closely compatible LoLin style NodeMCU. Technical specifications are as follows:

|  |  |  |
| --- | --- | --- |
| **S.No** | **Specification** | **Detail** |
|  | **Microcontroller** | ESP-8266 32-bit |
|  | **NodeMCU Model** | Amica |
|  | **NodeMCU Size** | 49mm x 26mm |
|  | **Pin Spacing** | **0.9" (22.86mm)** |
|  | **Clock Speed** | 80 MHz |
|  | **USB to Serial** | CP2102 |
|  | **USB Connector** | Micro USB |
|  | **Operating Voltage** | 3.3V |
|  | **Input Voltage** | 4.5V-10V |
|  | **Flash Memory/SRAM** | 4 MB / 64 KB |
|  | **Digital I/O Pins** | 11 |
|  | **Analog In Pins** | 1 |
|  | **ADC Range** | 0-3.3V |
|  | **UART/SPI/I2C** | 1 / 1 / 1 |
|  | **WiFi Built-In** | 802.11 b/g/n |
|  | **Temperature Range** | -40C - 125C |

**Table 2 - NodeMCU Specifications**

**2.2.2 Breadboard**

A breadboard (also known as a prototyping board) is a tool used for creating temporary electronic circuits by inserting and connecting components without the need for soldering. Here are some technical details about breadboards:

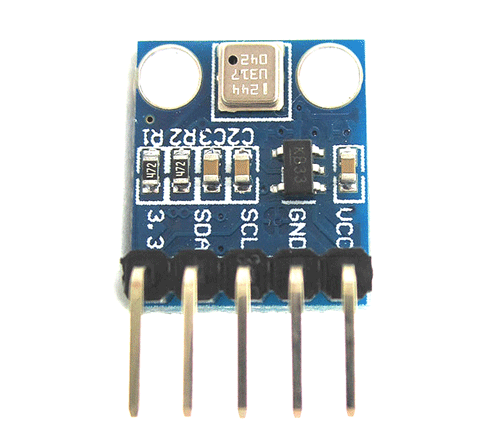
* Rows and columns: A typical breadboard has two sets of rows and columns. The rows are labeled A to J, and the columns are labeled 1 to 30. Each row and column is electrically connected vertically.
* Tie points: The holes on a breadboard are called tie points, and they are used to insert and connect components such as resistors, capacitors, LEDs, and integrated circuits (ICs).
* Power rails: The two outer columns on each side of the breadboard are called the power rails. They are used to provide power to the circuit and are typically connected to a power source such as a battery or a DC power supply.
* Distribution strips: The distribution strips run horizontally across the middle of the breadboard and are used to connect components in the same row.
* Breadboard size: Breadboards come in various sizes, with the most common size being 830 tie points (30 rows x 10 columns).
* Breadboard types: There are two main types of breadboards: solderless breadboards and solderable breadboards. Solderless breadboards are designed for temporary prototyping, while solderable breadboards are designed for more permanent circuit designs that require soldering.



**Figure 3: Bread Board with labels**

**2.2.3 BMP180 - Pressure Sensor (Barometer)**

The BMP180 is the function compatible successor of the BMP085, a high precision digital pressure sensors for consumer applications. The ultra -low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment.



**Figure 4 BMP 80 Pressure Sensor**

With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I2C interface allows for easy system integration with a micro controller. The BMP180 is based on piezo -resistive technology for EMC robustness, high accuracy and linearity as well as long term stability. Robert Bosch is the world market leader for pressure sensors in automotive applications



**Figure 5 BMP 180 Sensor Pinout**

|  |  |
| --- | --- |
| **Pin Name** | **Description** |
| VCC | Connected to +5V |
| GND | Connected to ground. |
| SDA | Serial Data pin (I2C interface) |
| SCL | Serial Clock pin (I2C interface) |
| 3.3V | If +5V is not present. Can power module by connecting +3.3V to this pin. |

**Table 3 BMP 180 Sensor Pinout**

**BMP 180 Module Features**

* Can measure temperature and altitude.
* Pressure range: 300 to 1100hPa
* High relative accuracy of ±0.12hPa
* Can work on low voltages
* 3.4Mhz I2C interface
* Low power consumption (3uA)
* Pressure conversion time: 5msec
* Potable size

**BMP 180 Module Specifications**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Component** | **Specs** |
|  | Operating  voltage of | 1.3V – 3.6V |
|  | Input voltage | :3.3V to 5.5V |
|  | Peak current | 1000uA |
|  | Consumes | 0.1uA standby |
|  | Maximum voltage at SDA , SCL : VCC + | 0.3V |
|  | Operating temperature | -40ºC to +80ºC |

**Table 4 BMP 180 Sensor Specifications**

**2.2.4 DHT11 Sensor ( Hygrometer)**

The DHT11 is a commonly used temperature and humidity sensor. The **DHT11 Temperature and Humidity Sensor** has a **temperature and humidity sensor complex**with a calibrated digital signal output. It offers high dependability and outstanding long-term stability by employing an innovative digital-signal-acquisition technique as well as temperature and humidity sensing technologies. This sensor consists of**a resistive-type humidity measurement component** and an**NTC temperature measurement component**

**DHT 11 Sensor Module Features**

* Voltage range: **3.5V** to **5.5V**
* **0.3mA**(measuring) 60uA operating current (standby)
* Serial data is output.
* **0°C**to **50°C** temperature range
* Humidity levels range from**20%** to**90%.**
* Temperature and humidity are both 16-bit resolutions.
* 1°C and 1 per cent accuracy

**DHT11 Pinout**

It consists of 4 Pins in total, listed below from left to right

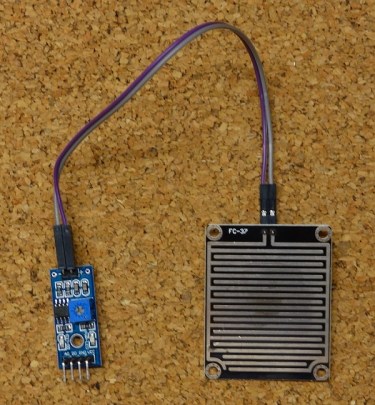
* Vcc: Need to provide +5V at this pinout.
* Data: It's the digital output pin, gives either 0V or 5V.
* NC: Not Connected. (It's left open for future design)
* GND: Need to provide Ground at this pinout



**Figure 6 DHT 11 Pinout**

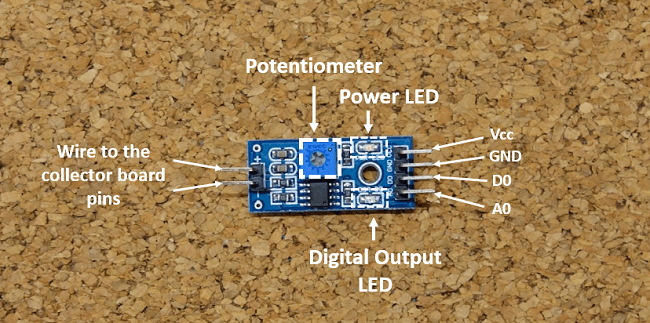
**2.2.5 Rain Sensor FC-37**

The FC-37 rain sensor is set up by two pieces: the electronic board (at the left) and the collector board (at the right) that collects the water drops, as you can see in the following figure:



**Figure 7 Rain Sensor**

The rain sensor has a built-in potentiometer for sensitivity adjustment of the digital output (D0). It also has a power LED that lights up when the sensor is turned on and a digital output LED.



**Figure 8 Rain Sensor Pinout**

|  |  |
| --- | --- |
| **SENSOR PIN** | **CONNECTION** |
| A0 | Any Analog Pin |
| D0 | Digital Pins |
| GND | Ground of board |
| VCC | 5V (Regulated Type) |

**Table 5 Rain Sensor Pinout**

The resistance of the collector board varies accordingly to the amount of water on its surface. When the board is:

* **Wet**: the resistance increases, and the output voltage decreases
* **Dry:** the resistance is lower, and the output voltage is higher



**Figure 9 Rain Sensor Output Voltage**

Technical specifications of the sensor used in this project are as follows: -

|  |  |  |
| --- | --- | --- |
| **S No** | **Component** | **Specs** |
|  | Operating Voltage | 5V |
|  | Output | Digital and Analog |
|  | Sensitivity | Adjustable |
|  | Output LED Indication |  |
|  | Bolt Holes | For easy installation |
|  | TTL Compatible |  |

**Table 6 Rain Sensor Specifications**

**2.2.6 Connecting Wires**

Jumper wires are a type of wire used in electronics and prototyping to connect components on a breadboard or other circuit board. They are typically made from flexible stranded wire with a plastic insulation and have pre-stripped ends that can be easily inserted into breadboard or connector pins. Here are some details about jumper wires:

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Specifications** | **Details** |
| **1** | Wire Gauge | 22 AWG |
| **2** | Conductor Material | Solid or stranded copper |
| **3** | Insulation Material | PVC or silicone |
| **4** | Insulation Colors | Assorted (e.g., red, black, yellow, green, blue, white) |
| **5** | Length | Varies (e.g., 3 inches, 6 inches, 12 inches) |
| **6** | Connector Type | Male to male, male to female, or female to female |
| **7** | Quantity | Typically sold in sets of 40, 65, or 120 |
| **8** | Package Type | Reusable plastic case or plastic bag |
| **9** | Operating Temperature Range | -40°C to +80°C |
| **10** | Voltage Rating | 300V |
| **11** | Current Rating | 3A |
| **12** | Features | Flexible, easy to bend, reusable |

**Table 7 Specification of Jumper Wires**



**Figure 10 Jumper Wires**

**CHAPTER 3**

**SYSTEM ARCHITECTURE**

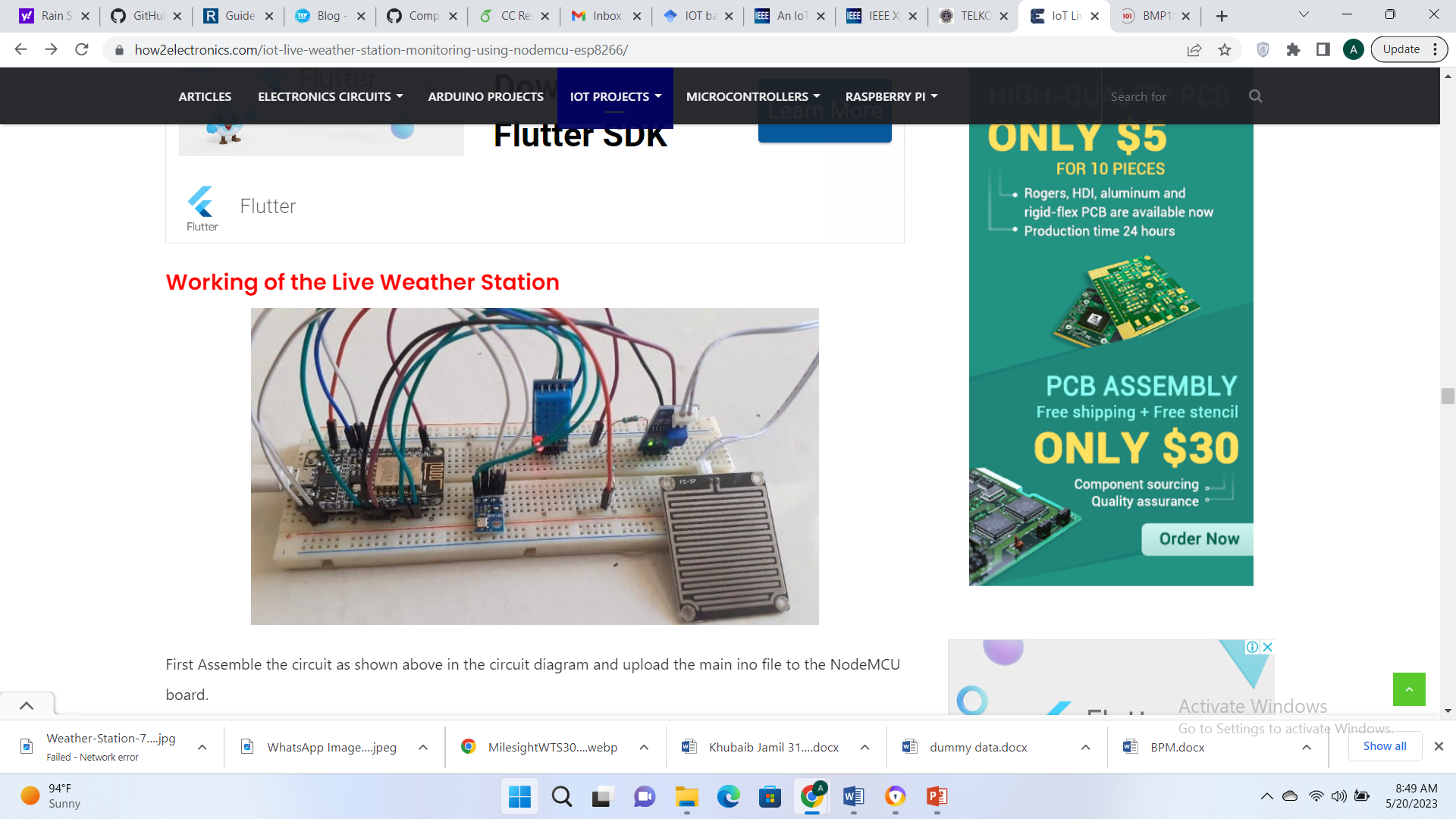
**3.1 Overview**

The system typically consists of various components as discussed above and it works in layers, together to collect, transmit, process, and analyze weather data. The sensors are deployed to capture real time date of weather. The interface is web based dashboard, which shows the reading in real time. System is connected to ThinkSpeak server for analysis of the gathered data, which is shown in graphical format.

The NodeMCU ESP8266 act as the brains of the system, providing the processing power and connectivity necessary to control the temperature, pressure and rain sensor. It is responsible for controlling the connection to the application, and connecting to the cloud based ThinkSpeak server.

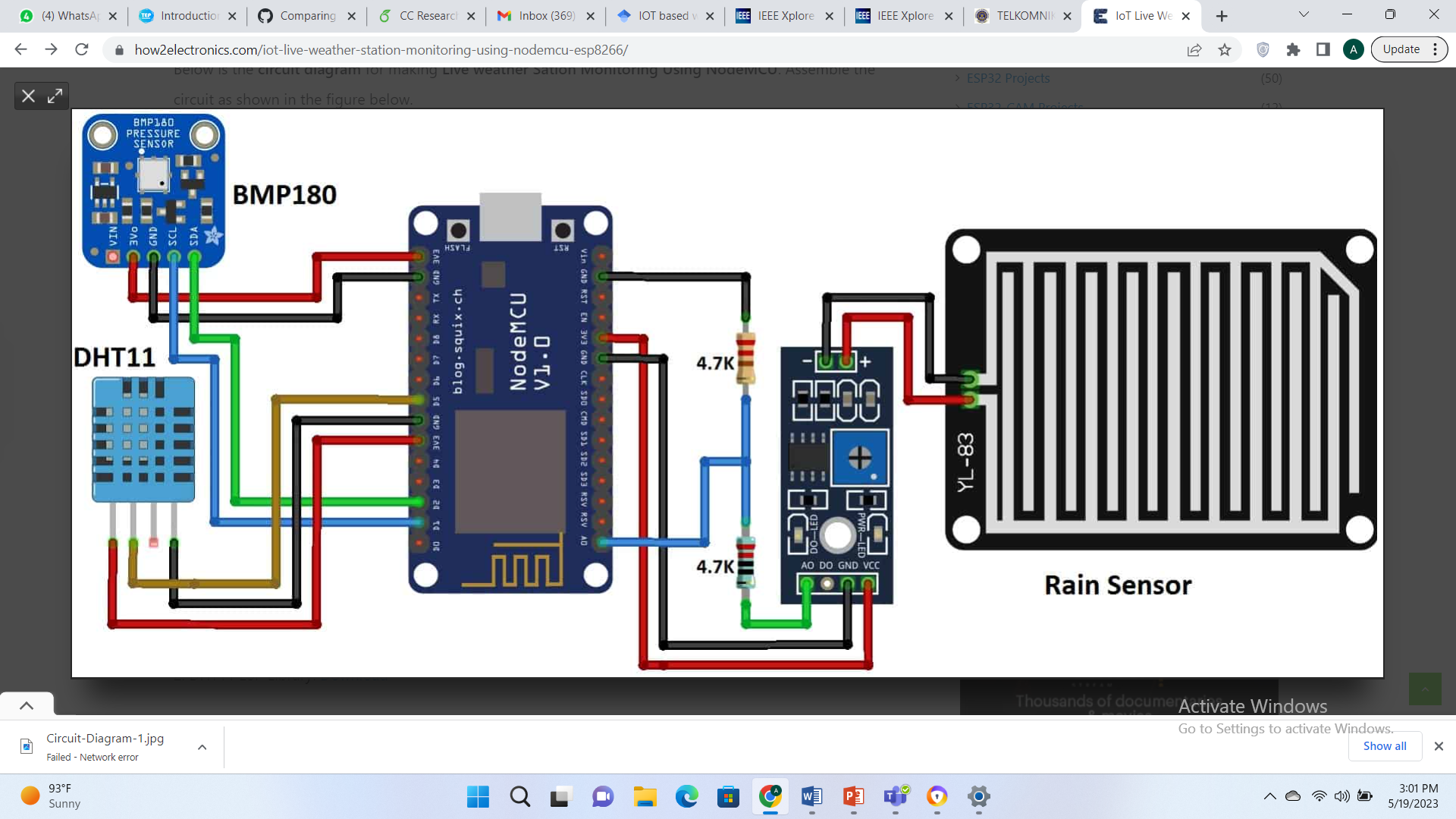
ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts.

After Interfacing the sensors with nodemcu both dht11 and BMP 180 are connected directly without using any external resistance but for rain sensor we are using 4.7K ohm resistors. The Wi-Fi connection in the above project is established using the ESP8266 Wi-Fi module. The ESP8266 module is connected to the Arduino Uno board via hardware serial communication, allowing the board to connect to a wireless network and transmit data over the internet.



**Fig 11 Architecture overview**

**3.2 Circuit Diagram**



**Fig 12 Circuit Diagram**

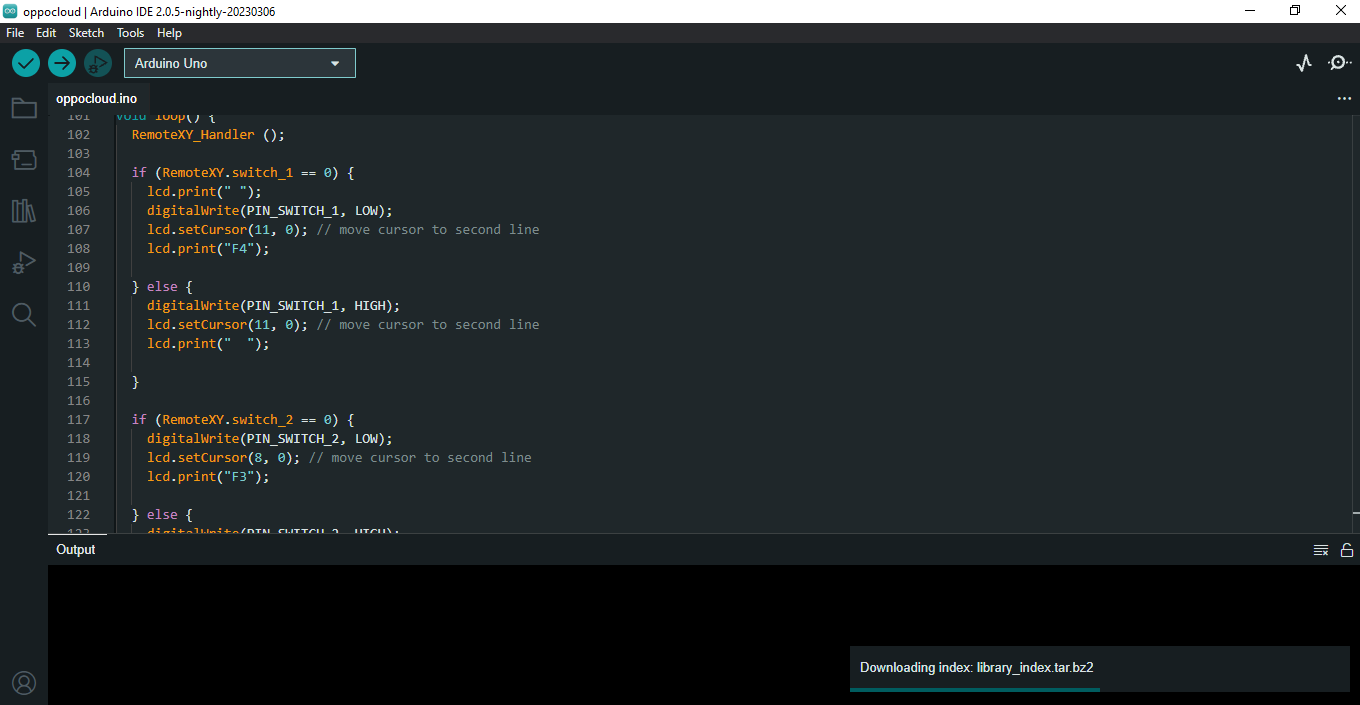
**3.3 Software Requirements**

* **Arduino IDE**

The Arduino Integrated Development Environment (IDE) is an open-source software platform that provides a comprehensive environment for programming Arduino boards. It is a software tool used to write, compile, and upload code to the Arduino microcontroller. The IDE is available for Windows, Mac OS X, and Linux, and it is free to download and use. [15]

The Arduino IDE includes a code editor with syntax highlighting, a serial monitor for debugging and communication with the microcontroller, and a compiler and linker for generating the binary code that runs on the Arduino board. The IDE also includes a library manager that makes it easy to install and manage third-party libraries.

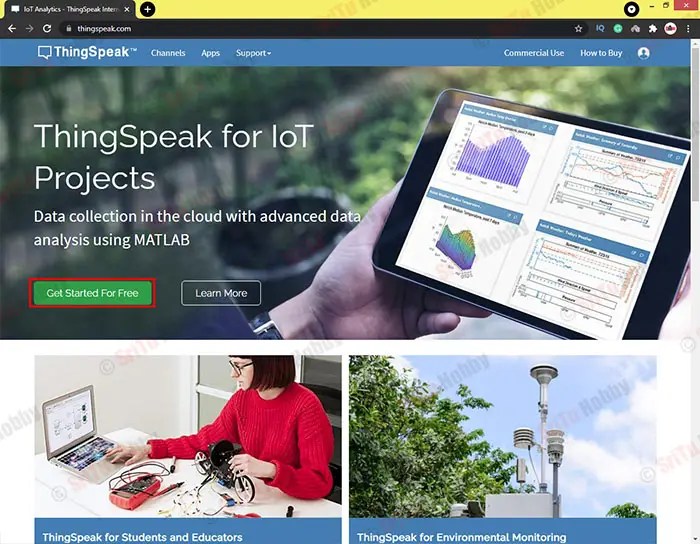
The Arduino IDE uses a simplified version of the C++ programming language, with a set of pre-defined functions and libraries that make it easier for beginners to get started with microcontroller programming. The code is written in a sketch file (.ino) and uploaded to the Arduino board via a USB cable or wireless communication.



**Figure 13: Arduino IDE Interface [15]**

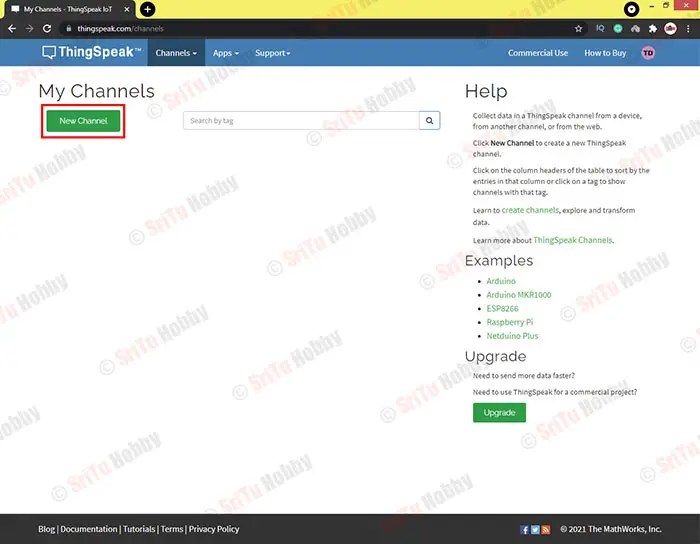
* **ThingSpeak**

After circuit diagram is complete and IDE code is active, we need to make account on ThingSpeak. First, go to the Thingspeak website and create a new account using your email address. Then, click the “New channel” button.

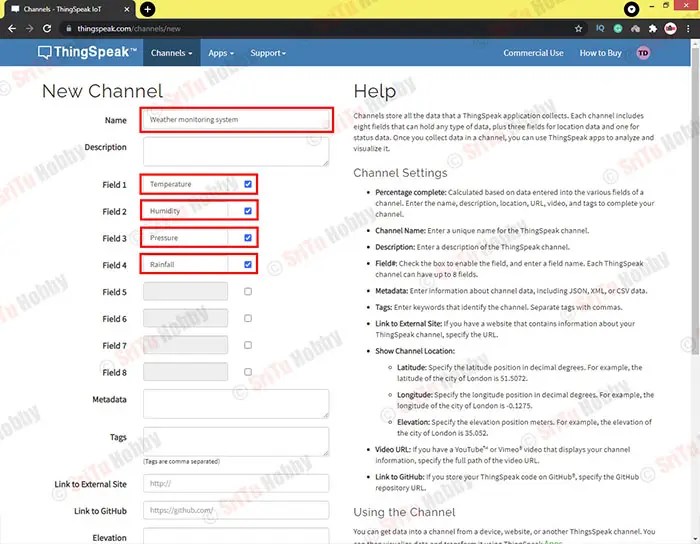


**Figure 14 ThingSpeak Login**

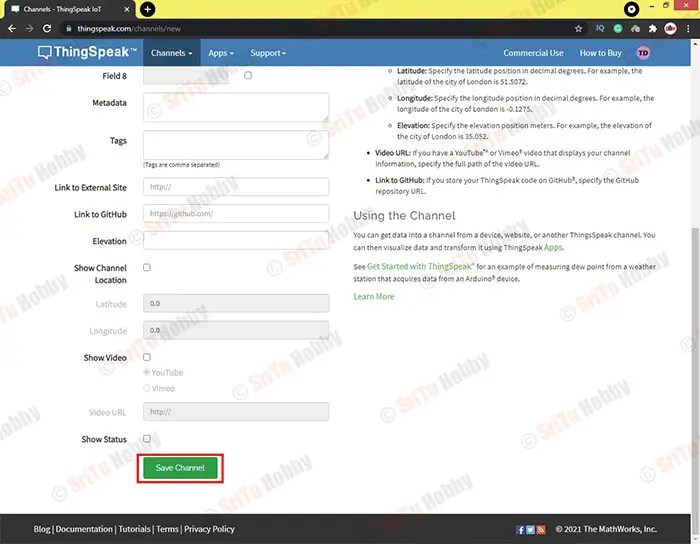
Create a New Channel. Next, enter your project name as you like. Then, activate the fields and name them Temperature, Humidity, Pressure, and Rainfall respectively. After, click the “Save channel” button.



**Figure 15 ThingSpeak Channel Creation**

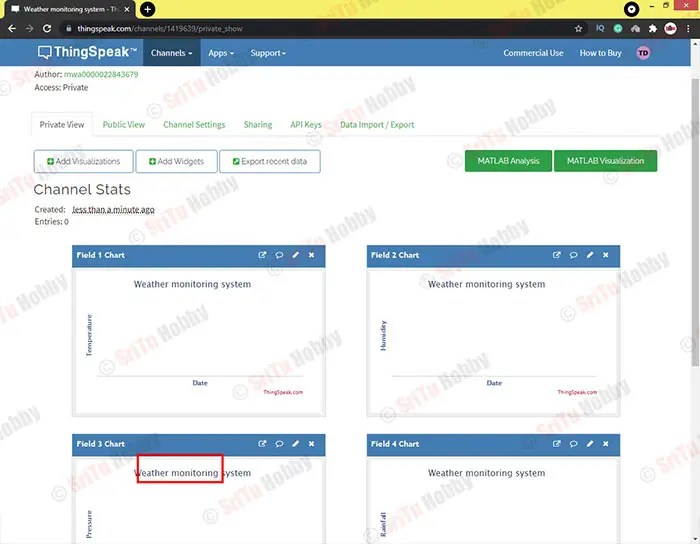


**Figure 16 ThingSpeak Field Adjustment for Sensors**



**Figure 17 Fields Adjustments in ThingSpeak**

After the fields are adjusted, we are required to save the channel. Then the main dashboard appears. Currently , it is showing empty graphs, for real time data we are required to connect our station to the application. Figure showing the real time point data is shown in the result section

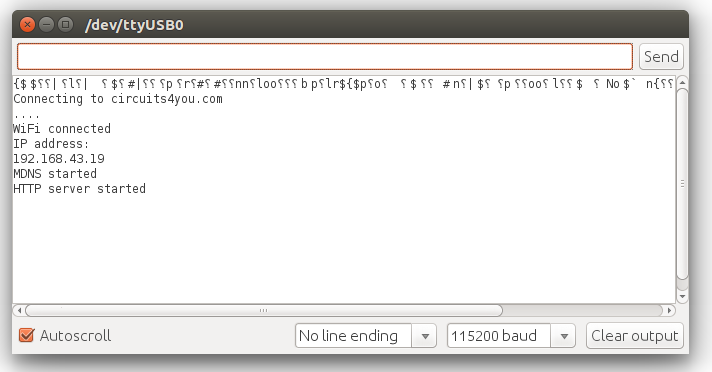


**Figure 18 Initial Dashboard**

* **HTML based Application**

Program is divided in two parts. First part contains ESP8266 WiFi and Hardware related functions i.e. our **main.ino** file. Second part is HTML and user interface GUI. it is **index.h**file. Before uploading program to ESP, we are required to update our SSID and Password in code and ensure that it is on same network.

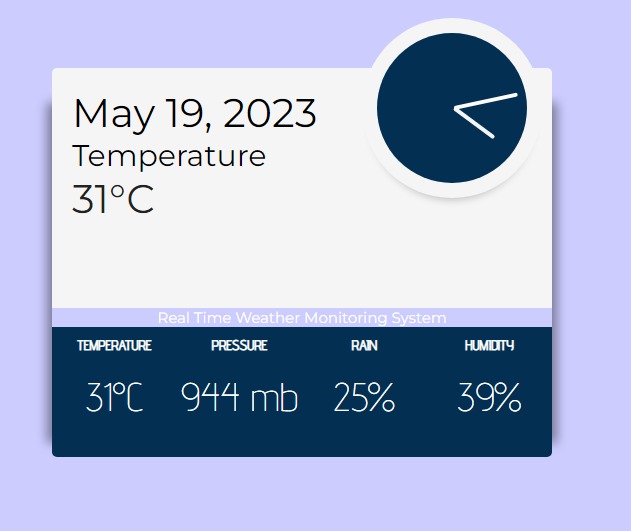
Once the code is complete, we need to open serial monitor, which will give us the IP Address



**Figure 19 Serial Monitor Output**

Open IP address in Web Browser.  After opening it in web browser, you will see that on board blue LED will Blink. this led toggles when it receives http request from web browser.

**3.4 Application Interface**

****

**Figure 20 Application Interface**

After all connections are done and the code has been uploaded to NodeMCU, Serial monitor shows the IP address which has been assigned to the device. Copying the IP and pasting it in web browser will display the live weather station dashboard as shown in figure 14

**3.5 Software Components**

Software consists of following parts.

* Connecting to WiFi
* Server Creation
* Getting Data from Sensors
* Sending GUI to user
* Updating Sensors values on web page

**3.6 Communication Protocols**

* At physical layer, The NodeMCU ESP8266 board has built-in Wi-Fi capabilities, making it a suitable choice for wireless connectivity
* At the internet layer, the Internet Protocol (IP) is commonly used. The weather station communicates with other devices or servers over the same netwrok using IP-based protocols.
* Hypertext Transfer Protocol (HTTP) is used for web based communication. Once the IP is generated, the same can be put in the browser to get the interface of the application.

**3.7 Code**

**3.7.1 Connection to Wifi**. Connection to wifi is made from this code. You need to give your desktop name and wifi password at this location.

https://circuits4you.com/wp-content/uploads/2019/03/ssid-password.png

|  |
| --- |
| String apiKey = "CP0W5CEKEKWIAM8E"; |
| const char \*ssid = "DESKTOP-HP08"; |
| const char \*pass = "12345678"; |
| const char\* server = "api.thingspeak.com"; |
| **3.7.2 .ino File Code**  #include <ESP8266WiFi.h>  #include <WiFiClient.h>  #include <ESP8266WebServer.h>  #include <SFE\_BMP180.h>  #include <Wire.h>    #include "index.h" //Our HTML webpage contents with javascripts  #include "DHTesp.h" //DHT11 Library for ESP    #define LED 2 //On board LED  #define DHTpin 0 //D3 of NodeMCU is GPIO0    SFE\_BMP180 pressure;    #define ALTITUDE 1655.0 // Altitude in meters    DHTesp dht;    //SSID and Password of your WiFi router  const char\* ssid = "DESKTOP-HP08";  const char\* password = "12345678";    ESP8266WebServer server(80); //Server on port 80    void handleRoot() {  String s = MAIN\_page; //Read HTML contents  server.send(200, "text/html", s); //Send web page  }    float humidity, temperature;    void handleADC() {  char status;  double T,P,p0,a;  double Tdeg, Tfar, phg, pmb;    status = pressure.startTemperature();  if (status != 0)  {  // Wait for the measurement to complete:  delay(status);  status = pressure.getTemperature(T);  if (status != 0)  {  // Print out the measurement:  //Serial.begin(74880);  Serial.print("temperature: ");  Serial.print(T,2);  Tdeg = T;  Serial.print(" deg C, ");  Tfar = (9.0/5.0)\*T+32.0;  Serial.print((9.0/5.0)\*T+32.0,2);  Serial.println(" deg F");    status = pressure.startPressure(3);  if (status != 0)  {  // Wait for the measurement to complete:  delay(status);  status = pressure.getPressure(P,T);  if (status != 0)  {  // Print out the measurement:  Serial.print("absolute pressure: ");  Serial.print(P,2);  pmb = P;  Serial.print(" mb, ");  phg = P\*0.0295333727;  Serial.print(P\*0.0295333727,2);  Serial.println(" inHg");    p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters (Boulder, CO)  Serial.print("relative (sea-level) pressure: ");  Serial.print(p0,2);  Serial.print(" mb, ");  Serial.print(p0\*0.0295333727,2);  Serial.println(" inHg");    a = pressure.altitude(P,p0);  Serial.print("computed altitude: ");  Serial.print(a,0);  Serial.print(" meters, ");  Serial.print(a\*3.28084,0);  Serial.println(" feet");  }  else Serial.println("error retrieving pressure measurement\n");  }  else Serial.println("error starting pressure measurement\n");  }  else Serial.println("error retrieving temperature measurement\n");  }  else Serial.println("error starting temperature measurement\n");      int rain = analogRead(A0);  rain = map(rain, 0, 1024, 0, 100);    //Create JSON data  String data = "{\"Rain\":\""+String(rain)+"\",\"Pressuremb\":\""+String(pmb)+"\",\"Pressurehg\":\""+String(phg)+"\", \"Temperature\":\""+ String(temperature) +"\", \"Humidity\":\""+ String(humidity) +"\"}";    digitalWrite(LED,!digitalRead(LED)); //Toggle LED on data request ajax  server.send(200, "text/plane", data); //Send ADC value, temperature and humidity JSON to client ajax request    delay(dht.getMinimumSamplingPeriod());    humidity = dht.getHumidity();  temperature = dht.getTemperature();    Serial.print("H:");  Serial.println(humidity);  Serial.print("T:");  Serial.println(temperature); //dht.toFahrenheit(temperature));  Serial.print("R:");  Serial.println(rain);  }    void setup()  {  Serial.begin(115200);  Serial.println();    // dht11 Sensor    dht.setup(DHTpin, DHTesp::DHT11); //for DHT11 Connect DHT sensor to GPIO 17  pinMode(LED,OUTPUT);    //BMP180 Sensor  if (pressure.begin())  Serial.println("BMP180 init success");  else  {  Serial.println("BMP180 init fail\n\n");  while(1); // Pause forever.  }    WiFi.begin(ssid, password); //Connect to your WiFi routers  Serial.println("");    // Wait for connection  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }    //If connection successful show IP address in serial monitor  Serial.println("");  Serial.print("Connected to ");  Serial.println(ssid);  Serial.print("IP address: ");  Serial.println(WiFi.localIP()); //IP address assigned to your ESP    server.on("/", handleRoot); //Which routine to handle at root location. This is display page  server.on("/readADC", handleADC); //This page is called by java Script AJAX    server.begin(); //Start server  Serial.println("HTTP server started");  }    void loop()  {  server.handleClient(); //Handle client requests  }  **3.7.3 Code with ThingSpeak Integration**  \*/ |
| #include <SFE\_BMP180.h> |
| #include <Wire.h> |
| #include <ESP8266WiFi.h> |
| #include "DHT.h" |
|  |
| DHT dht(D3, DHT11); |
| SFE\_BMP180 bmp; |
| double T, P; |
| char status; |
| WiFiClient client; |
|  |
| String apiKey = "CP0W5CEKEKWIAM8E"; |
| const char \*ssid = "DESKTOP-HP08"; |
| const char \*pass = "12345678"; |
| const char\* server = "api.thingspeak.com"; |
|  |
|  |
| void setup() { |
| Serial.begin(115200); |
| delay(10); |
| bmp.begin(); |
| Wire.begin(); |
| dht.begin(); |
| WiFi.begin(ssid, pass); |
|  |
|  |
| while (WiFi.status() != WL\_CONNECTED) { |
| delay(500); |
| Serial.print("."); |
| } |
| Serial.println(""); |
| Serial.println("WiFi connected"); |
| } |
|  |
| void loop() { |
| **//BMP180 sensor** |
| status = bmp.startTemperature(); |
| if (status != 0) { |
| delay(status); |
| status = bmp.getTemperature(T); |
|  |
| status = bmp.startPressure(3);// 0 to 3 |
| if (status != 0) { |
| delay(status); |
| status = bmp.getPressure(P, T); |
| if (status != 0) { |
|  |
| } |
| } |
| } |
|  |
| **//DHT11 sensor** |
| float h = dht.readHumidity(); |
| float t = dht.readTemperature(); |
|  |
| if (isnan(h) || isnan(t)) { |
| Serial.println("Failed to read from DHT sensor!"); |
| return; |
| } |
|  |
| **//Rain sensor** |
| int r = analogRead(A0); |
| r = map(r, 0, 1024, 0, 100); |
|  |
|  |
| if (client.connect(server, 80)) { |
| String postStr = apiKey; |
| postStr += "&field1="; |
| postStr += String(t); |
| postStr += "&field2="; |
| postStr += String(h); |
| postStr += "&field3="; |
| postStr += String(P, 2); |
| postStr += "&field4="; |
| postStr += String(r); |
| postStr += "\r\n\r\n\r\n\r\n"; |
|  |
| client.print("POST /update HTTP/1.1\n"); |
| client.print("Host: api.thingspeak.com\n"); |
| client.print("Connection: close\n"); |
| client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n"); |
| client.print("Content-Type: application/x-www-form-urlencoded\n"); |
| client.print("Content-Length: "); |
| client.print(postStr.length()); |
| client.print("\n\n\n\n"); |
| client.print(postStr); |
|  |
| Serial.print("Temperature: "); |
| Serial.println(t); |
| Serial.print("Humidity: "); |
| Serial.println(h); |
| Serial.print("absolute pressure: "); |
| Serial.print(P, 2); |
| Serial.println("mb"); |
| Serial.print("Rain"); |
| Serial.println(r); |
|  |
| } |
| client.stop(); |
| delay(1000); |
| } |

**CHAPTER 4**

**TESTING & VISUALIZATION**

**4.1 Testing**

We test our Project in real conditions with following methodology:

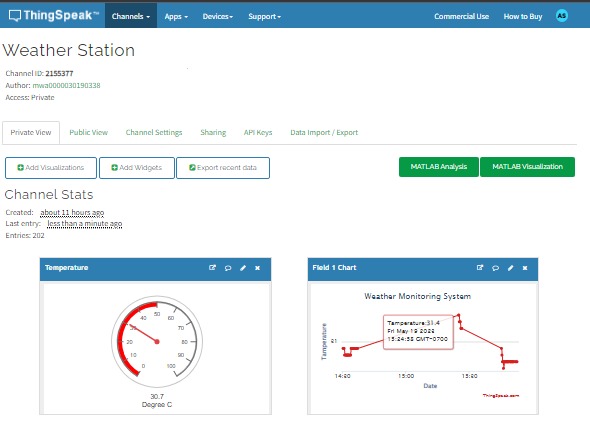
* Hardware Testing: The hardware components of the system, including the NodeMCU and various sensors and breadboard tested for functionality and proper connections. This includes verifying that they are powered on and connected to the appropriate modules and components.
* Reliability Testing: The system is tested over same time for a number of days to ensure that it is reliable and functioning. The system was tested in open air, closed quarters and at day and night time. It was ensured that with change in weather conditions the readings also change.
* Energy Efficiency Testing: The system is also tested for energy efficiency to ensure that it is saving energy and reducing electricity consumption. The idea behind is that energy efficiency is basically a design problem, and efficiency in usage of energy can be achieved by fine tuning the system scheduling and transferring filtered data
* Sometimes some parts of the system went unresponsive due to poor contact on the breadboard. If the ESP8266 becomes unresponsive out of a sudden, first thing we suggest doing would be to check the connections once again
* If you see a lot of garbage in the Serial Monitor console make sure the baud rate in the console is set accordingly with the ESP8266 settings.

**4.2 Visualization**

Results and analysis of the gathered data will be discussed in this section.

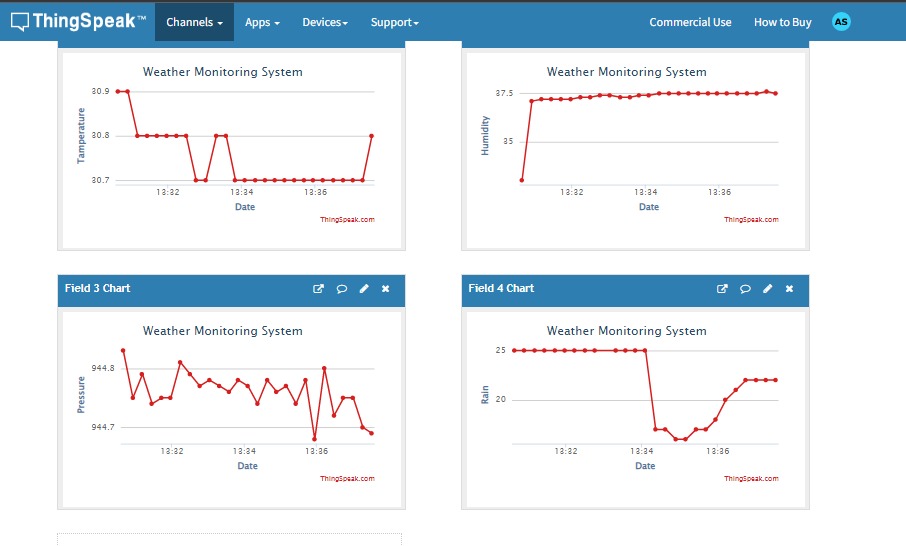
**4.2.1 ThingSpeak Dashboard**

After connection of the weather station with ThingSpeak following data was shown



**Figure 21 ThingSpeak Dashboard**

After the connection, data read by the sensors was fed into the ThingSpeak application to generate the analysis charts, as shows below.



**Figure 22 Sensor wise Dashboard**

**4.3 Analysis**

Data values were gathered from the proposed weather system. The results were obtained from the station placed in NUST university IoT lab for 3 days, SEECS parking and for evening in Hostel area. Temp, Humidity, Pressure and rain readings were taken at 0700 hrs, 1600 hrs and 1900 hrs for three consecutive days. The data was then compared with accuweather.com on those particular days.

**The difference in temperature is about 2 - 3 C, the difference in humidity is about 4 -5 %, no difference in rain status, and the difference in pressure is about 5 /10 MB**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Proposed Station | Accuweather | Proposed Station | Accuweather | Proposed Station | Accuweather |
| Sensor | **Time** | **Thurs** | | **Fri** | | **Sat** | |
| Temp | 0700 hrs | 30 C | 32 C | 30 C | 31 C | 31 C | 33 C |
|  | 1600hrs | 32 C | 28 C | 33 C | 35 C | 38 C | 40C |
| Pressure | 0700 hrs | 998 mb | 1013 mb | 989 mb | 1013 mb | 987 mb | 1001 mb |
|  | 1600 hrs | 985 mb | 1009 mb | 976 mb | 1009 Mb | 936 mb | 944 mb |
| Humidity | 0700 hrs | 39% | 33% | 43% | 47% | 27% | 30% |
|  | 1600hrs | 40% | 36% | 38% | 40% | 10% | 13% |
| Rain | 0700 hrs | 18% | 8% | 15% | 10% | 10% | 5% |
|  | 1600hrs | 20% | 15% | 16% | 13% | 6% | 2% |

**CHAPTER 5**

**CONCLUSION & REFRENCES**

**5.1 Conclusion**

To conclude, the IoT based weather station is an excellent example of how the Internet of Things (IoT) technology can be leveraged to improve energy efficiency, convenience, and overall user experience. By utilizing a network of interconnected sensors and devices, this project has achieved real-time data collection and analysis. The collected data provides valuable insights into the current weather conditions and serves as a foundation for accurate weather forecasting and prediction.

**5.2 References**

* 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies , IoT Based Weather Station, Ravi Kishore Kodali and Snehashish Mandal , National Institute of Technology, Warangal(ICCICCT)
* <http://telkomnika.uad.ac.id/index.php/TELKOMNIKA/article/viewFile/12784/7508>
* <https://how2electronics.com/iot-live-weather-station-monitoring-using-nodemcu-esp8266/>
* <https://components101.com/sensors/bmp180-atmospheric-pressure-sensor>
* <https://www.circuitschools.com/live-weather-station-using-nodemcu-esp8266-with-dashboard-iot/>
* <https://circuitdigest.com/microcontroller-projects/esp12-nodemcu-based-iot-weather-station>
* H. Saini, A. Thakur, S. Ahuja, N. Sabharwal, and N. Kumar, “Arduino based automatic wireless weather station with remote graphical application and alerts,” in 2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN), Feb 2016, pp. 605–609.