**INTRODUCTION**

The DHT-11 is a widely used digital temperature and humidity sensor that is commonly employed in various electronic projects and applications. The DHT-11 sensor is known for its simplicity, affordability, and ease of use.

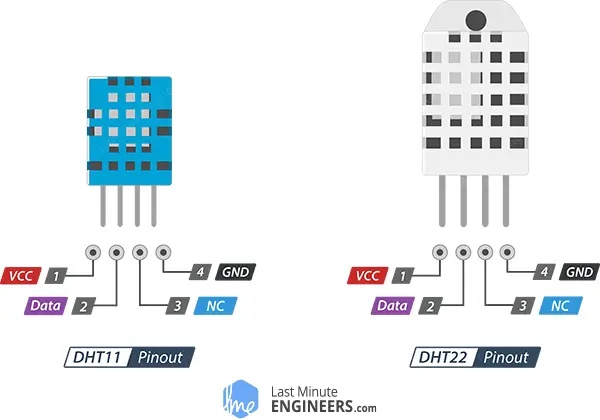
Typically, using the DHT-11 involves connecting it to a microcontroller, reading the digital signal, and then converting the obtained data into temperature and humidity values. This sensor is commonly utilized in weather stations, environmental monitoring systems, and various DIY projects where monitoring temperature and humidity is essential.

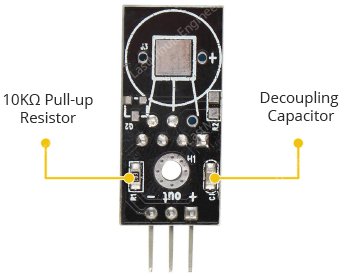
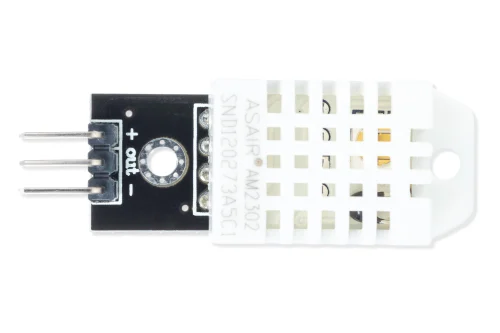
Specification :-

* Operating Voltage : 3 - 5V
* Max Operating Current : 3 - 2.5mA Max
* Temperature Range : 0-50°C / ±2°C
* Humidity Range : 20-80% / 5%
* Sampling Rate : 1 Hz (making every second)
* Advantage : Low cost
* Resolution : Humidity: 1% , Temperature: 1%

**DHT-11 Pinout**

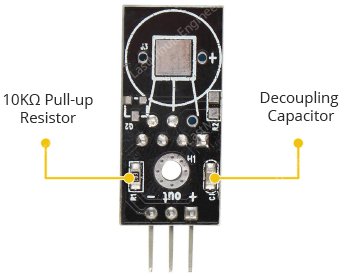
 A diagram of a sensor

Description automatically generated

* **VCC -** Pin provides power to the sensor. The supply voltage of the module ranges from 3.3V to 5.5V, a 5V supply is recommended. With a 5V power supply, the sensor can be placed up to 20 meters away. With 3.3V supply voltage, the sensor can be placed just 1 meter away.
* **DATA/SIG** - Pin is used for communication between the sensor and the microcontroller.
* **GND** – Ground pin

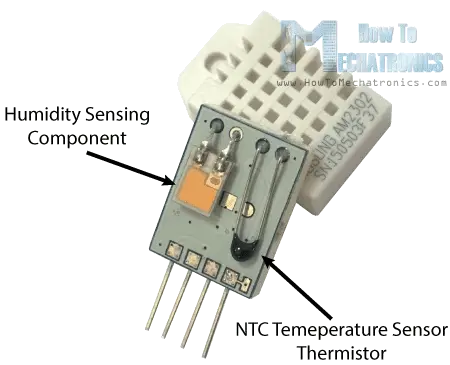
**DHT-11 Hardware Overview**



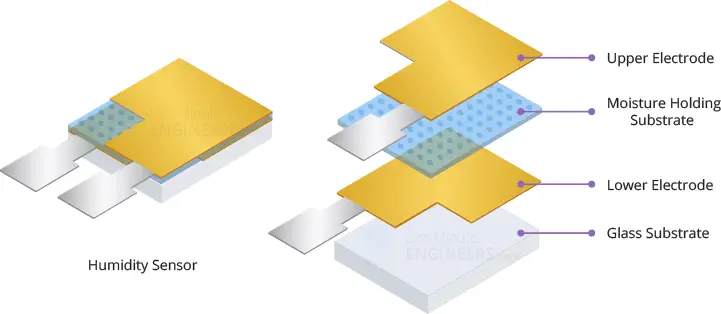
The DHT11 sensors typically necessitate an external 10K pull-up resistor on the output pin to facilitate effective communication with an Arduino. However, it's noteworthy that the module itself is equipped with an integrated pull-up resistor, obviating the need for an additional one.

Additionally, the module comes with a decoupling capacitor designed to filter power supply noise, enhancing the overall stability and reliability of the sensor's performance.

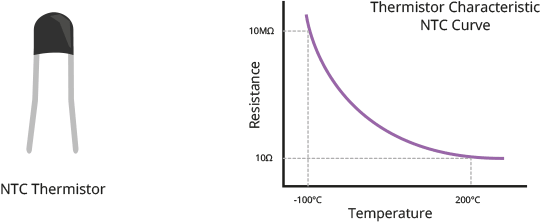
**Inside DHT-11 Sensor**



The humidity sensing component within the DHT11 sensor comprises two electrodes separated by a moisture-holding substrate, typically composed of salt or conductive plastic polymer. As humidity levels increase, the substrate absorbs water vapor, leading to the release of ions and a subsequent reduction in resistance between the two electrodes. This alteration in resistance is directly proportional to the humidity, providing a measurable parameter for estimating relative humidity.

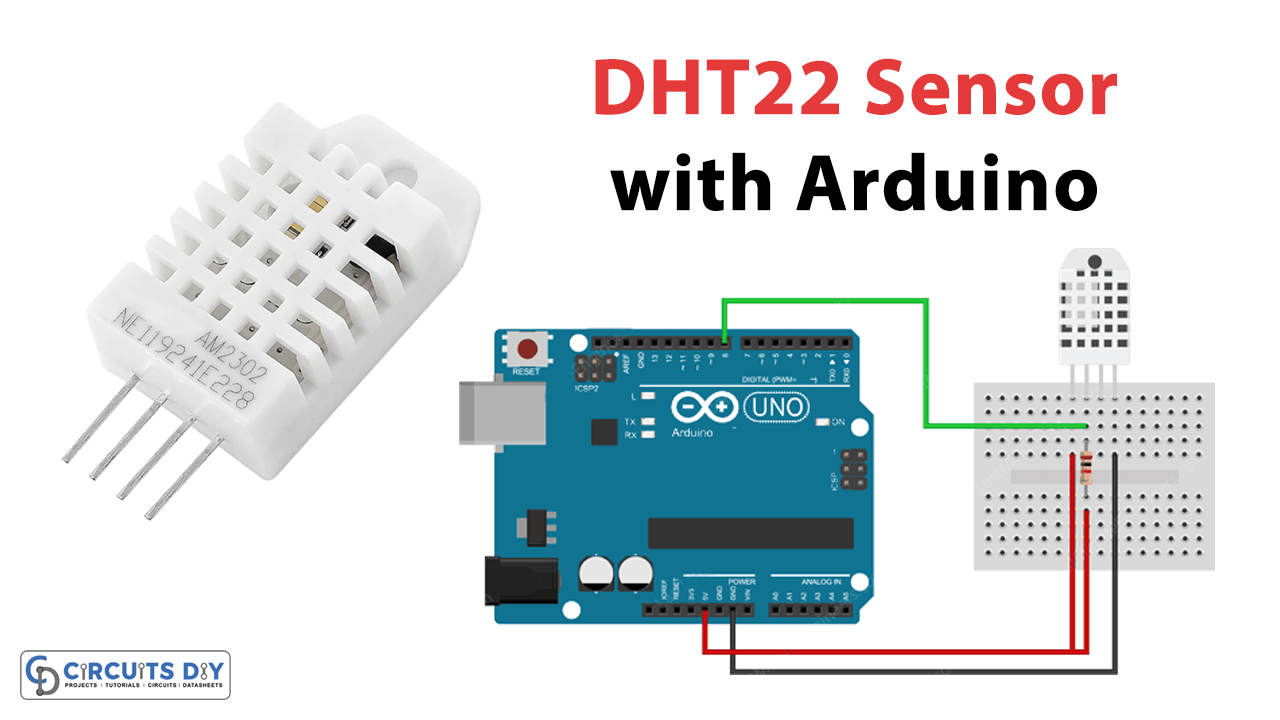


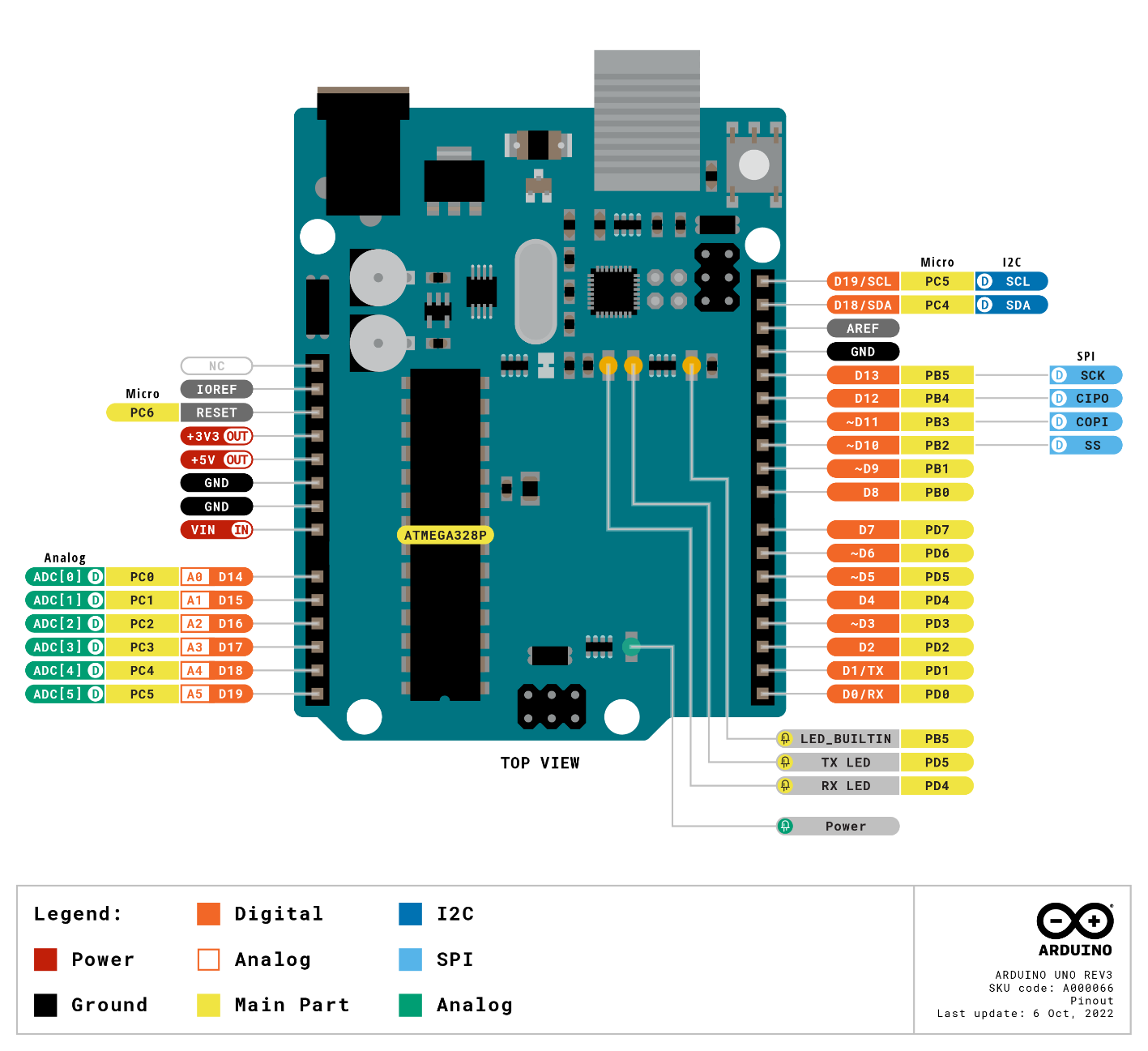
In addition to humidity measurement, the DHT11 sensor integrates a Negative Temperature Coefficient (NTC) thermistor for temperature sensing. A thermistor is a specialized resistor whose resistance varies significantly with temperature. While all resistors exhibit slight temperature-dependent resistance changes, thermistors are designed to undergo substantial resistance variations, typically exceeding 100 ohms per degree. The "NTC" designation indicates a Negative Temperature Coefficient, implying that the resistance of the thermistor decreases as the temperature rises.



The DHT11 sensor further incorporates an 8-bit Small Outline Integrated Circuit (SOIC-14) packaged IC. This IC performs crucial functions, including the measurement and processing of analog signals using stored calibration coefficients. It converts the analog signal to a digital format and outputs a digital signal containing precise temperature and humidity data. This comprehensive internal structure enables the DHT11 sensor to provide accurate and reliable environmental data through its digital output.

**WIRING DHT-11 MODULE WITH ARDUINO**





|  |  |
| --- | --- |
| **Arduino Uno/Arduino Nano (Pin Number)** | **DHT-11 (Pins Number)** |
| 8 | DATA/OUT/SIG |
| 5v | VCC |
| GND | GND |

## **Installing DHT library**

The DHT sensors have their own proprietary single-wire data transfer protocol. This protocol requires precise timing. We don’t have to worry too much about this, though, because we’ll be using the [DHTlib library](https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib" \t "_blank), which handles almost everything.

To install the library, navigate to Sketch > Include Library > Manage Libraries… Wait for the Library Manager to download the libraries index and update the list of installed libraries.

A screenshot of a computer

Description automatically generated

Filter your search by entering **‘DHTlib’**. There should only be a single entry. Click on that and then choose Install.

A screenshot of a computer

Description automatically generated

**Displaying Reading on Serial Monitor**

**Code:**

#include <dht.h> // Include library

#define outPin 8 // Defines pin number to which the sensor is connected

dht DHT; // Creates a DHT object

void setup () {

Serial.begin(9600);

}

void loop () {

int readData = DHT.read22(outPin);

float t = DHT.temperature; // Read temperature

float h = DHT.humidity; // Read humidity

Serial.print("Temperature = ");

Serial.print(t);

Serial.print("°C | ");

Serial.print((t\*9.0)/5.0+32.0); // Convert celsius to fahrenheit

Serial.println("°F ");

Serial.print("Humidity = ");

Serial.print(h);

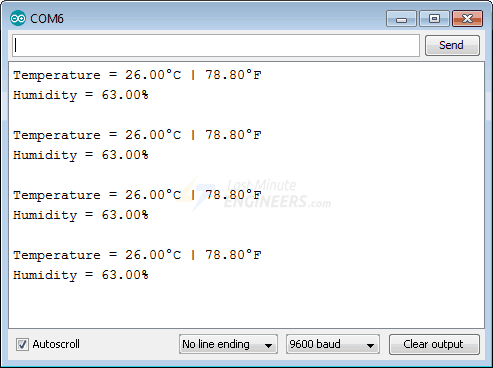
Serial.println("% ");

Serial.println("");

delay(2000); // wait two seconds

}

After uploading the sketch, you should see the following output on the serial monitor.



### **Code Explanation:**

The sketch begins by including the DHT library. Following that, we specify the Arduino pin number to which our sensor’s Data pin is connected and create a **DHT** object.

#include <dht.h>

#define outPin 8

dht DHT;

In the setup, we initialize the serial communication.

void setup() {

Serial.begin(9600);

}

In the loop, we use the **read22()** function to read the DHT22 module. This function takes as a parameter the sensor’s Data pin number.

int readData = DHT.read22(outPin);

We can now retrieve the humidity and temperature values by accessing the DHT object’s properties using dots **.**notation.

float t = DHT.temperature; // Read temperature

float h = DHT.humidity; // Read humidity

The DHT object returns the temperature in degrees Celsius (°C). It is easy to convert to Fahrenheit (°F) using the following formula:

T(°F) = T(°C) × 9/5 + 32

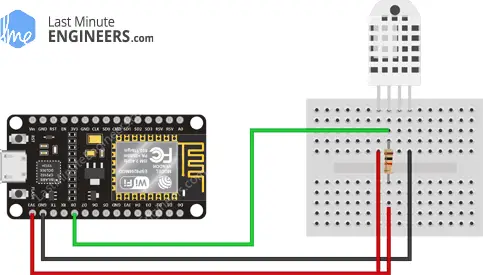
Serial.print((t \* 9.0) / 5.0 + 32.0);

**Wiring DHT-22 Module with ESP8266**

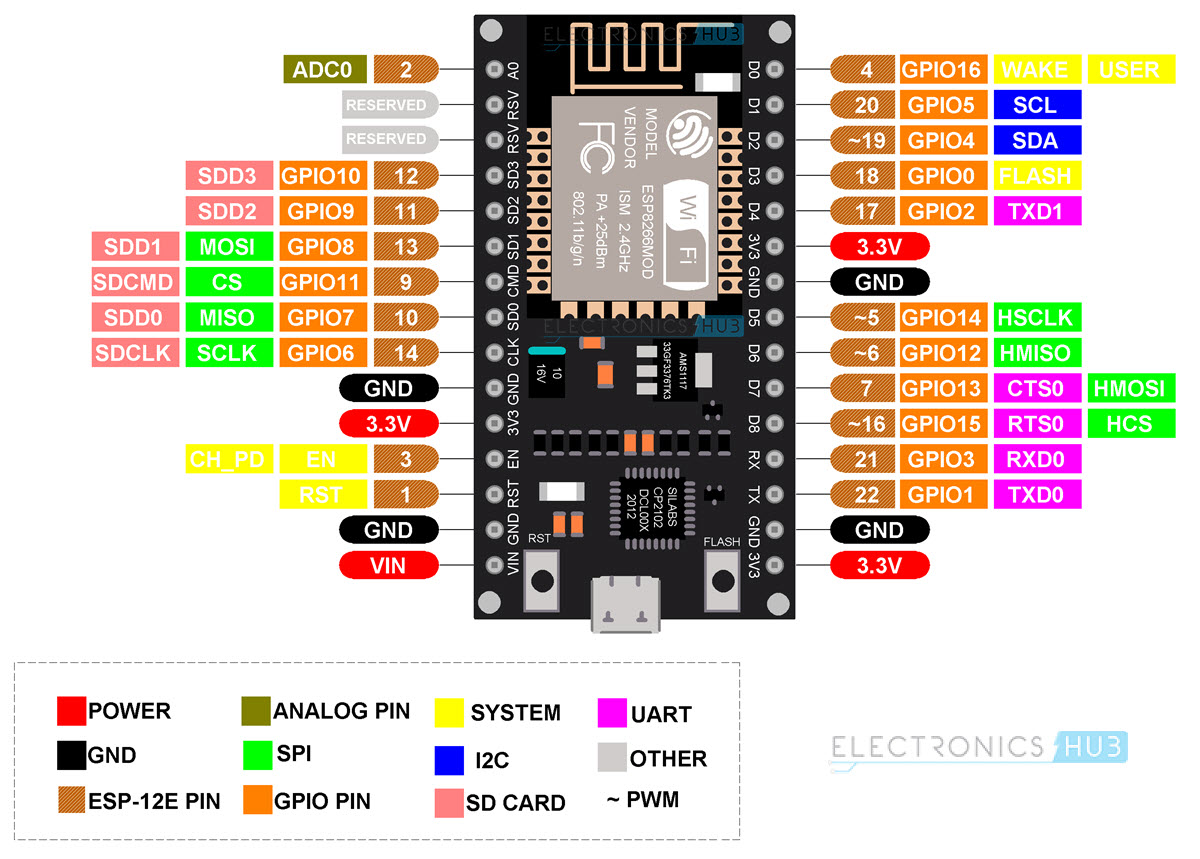
The ESP8266 NodeMCU as the control device, which connects to an existing Wi-Fi network and creates a Web Server. When a device connects to this web server, the ESP8266 will read the temperature and relative humidity from the DHT22 sensor and send it to the device’s web browser with a nice interface.

Connecting a DHT22 sensor to an ESP8266 is easy. Begin by placing the ESP8266 on your breadboard, making sure that each side of the board is on a different side of the breadboard.

Place the sensor on your breadboard next to the ESP8266. Finally, add a 10KΩ pull-up resistor between VCC and the data line to keep it HIGH for proper communication between the sensor and the ESP8266.









|  |  |
| --- | --- |
| **ESP8266 (Pin Number)** | **DHT-11 (Pin Number)** |
| D8 | DATA/SIG/OUT |
| 3.3V | VCC |
| GND | GND |

## **Installing DHT Sensor Library**

To begin reading sensor data, you will need to install the [DHT sensor library](https://github.com/adafruit/DHT-sensor-library). It is available from the Arduino library manager.

To install the library, navigate to **Sketch** > **Include Library** > **Manage Libraries…** Wait for the Library Manager to download the libraries index and update the list of installed libraries.

A screenshot of a computer

Description automatically generated

Filter your search by entering ‘**DHT sensor**’. Look for the **DHT sensor library** by **Adafruit**. Click on that entry and then choose Install.

A screenshot of a computer

Description automatically generated

The DHT sensor library makes use of the [Adafruit Sensor support backend](https://github.com/adafruit/Adafruit_Sensor). So, look for **Adafruit Unified Sensor** in the library manager and install it as well.

A screenshot of a computer

Description automatically generated

## **Creating an ESP8266 Web Server using WiFi Station (STA) mode**

Before you begin uploading the sketch, you must replace the following two variables with your network credentials so that the ESP8266 can connect to an existing network.

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

**Code:**

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include "DHT.h"

#define DHTTYPE DHT22 // DHT 22

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);

// DHT Sensor

uint8\_t DHTPin = D8;

// Initialize DHT sensor.

DHT dht(DHTPin, DHTTYPE);

float Temperature;

float Humidity;

void setup() {

Serial.begin(115200);

delay(100);

pinMode(DHTPin, INPUT);

dht.begin();

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

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

delay(1000);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle\_OnConnect);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

}

void handle\_OnConnect() {

Temperature = dht.readTemperature(); // Gets the values of the temperature

Humidity = dht.readHumidity(); // Gets the values of the humidity

server.send(200, "text/html", SendHTML(Temperature,Humidity));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(float Temperaturestat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 NodeMCU Weather Report</h1>\n";

ptr +="<p>Temperature: ";

ptr +=(int)Temperaturestat;

ptr +="°C</p>";

ptr +="<p>Humidity: ";

ptr +=(int)Humiditystat;

ptr +="%</p>";

ptr +="</div>\n";

ptr +="</body>\n";

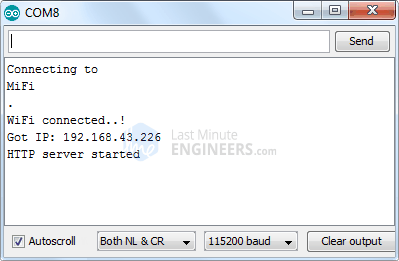
ptr +="</html>\n";

return ptr;

}

## **Accessing the Web Server**

After uploading the sketch, open the Serial Monitor at 115200 baud and press the RESET button on the NodeMCU.



Navigate the IP address on the serial monitor.



**Detailed Code Explanation**

The sketch starts by importing the esp8266wifi.h library, which houses esp8266-specific methods essential for network connection. subsequently, the esp8266webserver.h library is included. this library provides methods that streamline server configuration and facilitate the handling of incoming http requests, abstracting away low-level implementation intricacies. lastly, the dht.h library is included to incorporate functionalities related to the dht sensor, simplifying the code for interacting with and retrieving data from the sensor.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include "DHT.h"

it will join the existing WiFi network. So, we need to specify the **SSID** and **password**.

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

After including the necessary libraries, the sketch proceeds to instantiate an object of the ESP8266WebServer library. This object creation enables access to the library's functions throughout the program. The constructor of this object takes a parameter, specifically the port number to which the server will be listening. As HTTP typically utilizes port 80 by default, the constructor is initialized with this value. This setup ensures that connections to the server can be established without explicitly specifying the port number in the URL, enhancing the simplicity of accessing the server.

// declare an object of WebServer library

ESP8266WebServer server(80);

we define the GPIO pin number on the ESP8266 NodeMCU to which our sensor’s Data pin is connected and create a DHT object. So, we can access DHT library-specific functions.

// DHT Sensor

uint8\_t DHTPin = D8;

// Initialize DHT sensor.

DHT dht(DHTPin, DHTTYPE);

Creating two float variables set up: Temperature and Humidity.

float Temperature;

float Humidity;

### **Inside Setup () Function**

we establish a serial connection for debugging purposes and configure the GPIO pin to behave as an INPUT. We also initialize the DHT object.

Serial.begin(115200);

delay(100);

pinMode(DHTPin, INPUT);

dht.begin();

In the setup function of the sketch, the **WiFi.begin()** function is employed to establish a connection to the WiFi network. This function requires two parameters:

1. **SSID (Network Name):** The Service Set Identifier, or SSID, is the unique name that identifies a Wi-Fi network. The SSID is a case-sensitive string that is usually assigned by the network administrator or set by the user when configuring their Wi-Fi router.
2. **Password:** The password associated with the selected Wi-Fi network. This password is used to authenticate and authorize the device attempting to connect to the network.

The **WiFi.begin()** function initiates the connection process, and the ESP8266 module attempts to connect to the specified Wi-Fi network using the provided SSID and password. This function is crucial for enabling network communication and allowing the ESP8266 device to interact with other devices or services over the Wi-Fi network.

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

While the ESP8266 attempts to connect to the network, we can use the **WiFi.status()** function to check the connectivity status.

//check wi-fi is connected to wi-fi network

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

delay(1000);

Serial.print(".");

}

Once connected to the network, the **WiFi.localIP()** function is used to print the IP address assigned to the ESP8266.

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

To handle incoming HTTP requests, we must specify which code should be executed when a specific URL is accessed. For this, we use the **.on()** method. This method accepts two parameters: a relative URL path and the name of the function to be executed when that URL is visited.

The code below indicates that when a server receives an HTTP request on the root (/) path, it will call the **handle\_OnConnect()** function. It is important to note that the URL specified is a relative path.

server.on("/", handle\_OnConnect);

We haven’t specified what the server should serve if the client requests a URL that isn’t specified with **server.on()** . It should give a 404 error (Page Not Found) as a response. To accomplish this, we use the **server.onNotFound()** method.

server.onNotFound(handle\_NotFound);

Now, to start the server, we call the server object’s **begin ()** method.

server.begin();

Serial.println("HTTP server started");

### **Inside Loop () Function**

Actual incoming HTTP requests are handled in the loop function. For this, we use the server object’s **handleClient()** method.

server.handleClient();

Now we must write the **handle\_OnConnect()**  function, which we previously attached to the root (/) URL with **server.on**. We begin this function by reading temperature and humidity values from the sensor.

We use the send method to respond to an HTTP request. Although the method can be called with several different arguments, the simplest form requires the HTTP response code, the content type, and the content.

The first parameter we pass to the send method is the code 200 (one of the [HTTP status codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)), which corresponds to the OK response. Then we specify the content type as “text/html,” and finally we pass the **Send HTML ()** custom function, which generates a dynamic HTML page with the temperature and humidity readings.

void handle\_OnConnect() {

Temperature = dht.readTemperature(); // Gets the values of the temperature

Humidity = dht.readHumidity(); // Gets the values of the humidity

server.send(200, "text/html", SendHTML(Temperature,Humidity));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

### **Displaying the HTML Web Page**

Whenever the ESP8266 web server receives a request from a web client, the **sendHTML()** function generates a web page. It simply concatenates HTML code into a long string and returns to the **server.send()** function we discussed earlier. The function uses temperature and humidity readings as parameters to generate HTML content dynamically.

The first text you should always send is the **<!DOCTYPE>** declaration, which indicates that we’re sending HTML code.

String SendHTML(float Temperaturestat ,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

The **<meta>** viewport element makes the web page responsive, ensuring that it looks good on all devices. The title tag determines the page’s title.

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

### **Styling the Web Page**

Following that, we have some CSS to style the overall appearance of the web page. We select the Helvetica font and define the content to be displayed as an inline-block, center-aligned.

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

The code that follows then sets the color, font, and margin around the body, H1, and p tags.

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

### **Setting the Web Page Heading**

Next, the heading of the web page is set. You can change this text to anything that works for your application.

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Report</h1>\n";

### **Displaying Temperature and Humidity on Web Page**

We used the paragraph tag to display temperature and humidity values. Note that these values ​​are converted to integers using type casting.

ptr +="<p>Temperature: ";

ptr +=(int)Temperaturestat;

ptr +="°C</p>";

ptr +="<p>Humidity: ";

ptr +=(int)Humiditystat;

ptr +="%</p>";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

## **Styling Web Page to Look More Attractive**

Programmers tend to overlook aesthetics, but with a little effort, our webpage can appear more attractive. The screenshot below will give you an idea of what we’re going to do.



**Code :**

String SendHTML(float TempCstat,float TempFstat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<link href=\"https://fonts.googleapis.com/css?family=Open+Sans:300,400,600\" rel=\"stylesheet\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #333333;}\n";

ptr +="body{margin-top: 50px;}\n";

ptr +="h1 {margin: 50px auto 30px;}\n";

ptr +=".side-by-side{display: inline-block;vertical-align: middle;position: relative;}\n";

ptr +=".humidity-icon{background-color: #3498db;width: 30px;height: 30px;border-radius: 50%;line-height: 36px;}\n";

ptr +=".humidity-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".humidity{font-weight: 300;font-size: 60px;color: #3498db;}\n";

ptr +=".temperature-icon{background-color: #f39c12;width: 30px;height: 30px;border-radius: 50%;line-height: 40px;}\n";

ptr +=".temperature-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".temperature{font-weight: 300;font-size: 60px;color: #f39c12;}\n";

ptr +=".superscript{font-size: 17px;font-weight: 600;position: absolute;right: -20px;top: 15px;}\n";

ptr +=".data{padding: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Report</h1>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side temperature-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_1\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n";

ptr +="width=\"9.915px\" height=\"22px\" viewBox=\"0 0 9.915 22\" enable-background=\"new 0 0 9.915 22\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M3.498,0.53c0.377-0.331,0.877-0.501,1.374-0.527C5.697-0.04,6.522,0.421,6.924,1.142\n";

ptr +="c0.237,0.399,0.315,0.871,0.311,1.33C7.229,5.856,7.245,9.24,7.227,12.625c1.019,0.539,1.855,1.424,2.301,2.491\n";

ptr +="c0.491,1.163,0.518,2.514,0.062,3.693c-0.414,1.102-1.24,2.038-2.276,2.594c-1.056,0.583-2.331,0.743-3.501,0.463\n";

ptr +="c-1.417-0.323-2.659-1.314-3.3-2.617C0.014,18.26-0.115,17.104,0.1,16.022c0.296-1.443,1.274-2.717,2.58-3.394\n";

ptr +="c0.013-3.44,0-6.881,0.007-10.322C2.674,1.634,2.974,0.955,3.498,0.53z\"/>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side temperature-text\">Temperature</div>\n";

ptr +="<div class=\"side-by-side temperature\">";

ptr +=(int)TempCstat;

ptr +="<span class=\"superscript\">°C</span></div>\n";

ptr +="</div>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side humidity-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_2\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n\"; width=\"12px\" height=\"17.955px\" viewBox=\"0 0 13 17.955\" enable-background=\"new 0 0 13 17.955\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M1.819,6.217C3.139,4.064,6.5,0,6.5,0s3.363,4.064,4.681,6.217c1.793,2.926,2.133,5.05,1.571,7.057\n";

ptr +="c-0.438,1.574-2.264,4.681-6.252,4.681c-3.988,0-5.813-3.107-6.252-4.681C-0.313,11.267,0.026,9.143,1.819,6.217\"></path>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side humidity-text\">Humidity</div>\n";

ptr +="<div class=\"side-by-side humidity\">";

ptr +=(int)Humiditystat;

ptr +="<span class=\"superscript\">%</span></div>\n";

ptr +="</div>\n";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

### **Code Explanation**

We already know that the **<!DOCTYPE>** declaration informs the browser that we are sending HTML code, and the **<meta>** viewport element makes the web page responsive. The only difference here is that we’ll be using [Google Fonts](https://fonts.google.com/). Google offers hundreds of free web fonts for commercial and personal use.

For our website, we will use the Google-commissioned [Open Sans](https://fonts.google.com/specimen/Open+Sans) web font. The Google font is embedded in your HTML document’s head using the **<link>** tag.

For our page, we’ve chosen font weights of 300 (Light), 400 (Regular), and 600 (Bold). You can choose as many as you want, but keep in mind that excessive font weights slow down page load time.

You can also add italic style by simply appending an i character to the end of the font weight, for example, 400i will embed regular-italic-style-font.

It is important to note that google fonts are loaded dynamically; you will not be able to see the Google font unless you have an active internet connection on the device from which you are accessing this page.

String SendHTML(float TempCstat,float TempFstat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<link href=\"https://fonts.googleapis.com/css?family=Open+Sans:300,400,600\" rel=\"stylesheet\">\n";

Next, we will apply the Open Sans font to the entire HTML document. We must also specify sans-serif as our fallback font. If the first specified font fails to load, the browser will attempt to load the fallback font.

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #333333;}\n";

ptr +="body{margin-top: 50px;}\n";

ptr +="h1 {margin: 50px auto 30px;}\n";

Next, we apply CSS for Humidity & Temperature – icons, titles, and values. All three of these elements are inline and vertically aligned. The icon background is made circular using a border radius of 50% and 30px height and width.

ptr +=".side-by-side{display: inline-block;vertical-align: middle;position: relative;}\n";

ptr +=".humidity-icon{background-color: #3498db;width: 30px;height: 30px;border-radius: 50%;line-height: 36px;}\n";

ptr +=".humidity-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".humidity{font-weight: 300;font-size: 60px;color: #3498db;}\n";

ptr +=".temperature-icon{background-color: #f39c12;width: 30px;height: 30px;border-radius: 50%;line-height: 40px;}\n";

ptr +=".temperature-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".temperature{font-weight: 300;font-size: 60px;color: #f39c12;}\n";

ptr +=".superscript{font-size: 17px;font-weight: 600;position: absolute;right: -20px;top: 15px;}\n";

ptr +=".data{padding: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

for temperature icon displayed above

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 NodeMCU Weather Report</h1>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side temperature-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_1\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n";

ptr +="width=\"9.915px\" height=\"22px\" viewBox=\"0 0 9.915 22\" enable-background=\"new 0 0 9.915 22\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M3.498,0.53c0.377-0.331,0.877-0.501,1.374-0.527C5.697-0.04,6.522,0.421,6.924,1.142\n";

ptr +="c0.237,0.399,0.315,0.871,0.311,1.33C7.229,5.856,7.245,9.24,7.227,12.625c1.019,0.539,1.855,1.424,2.301,2.491\n";

ptr +="c0.491,1.163,0.518,2.514,0.062,3.693c-0.414,1.102-1.24,2.038-2.276,2.594c-1.056,0.583-2.331,0.743-3.501,0.463\n";

ptr +="c-1.417-0.323-2.659-1.314-3.3-2.617C0.014,18.26-0.115,17.104,0.1,16.022c0.296-1.443,1.274-2.717,2.58-3.394\n";

ptr +="c0.013-3.44,0-6.881,0.007-10.322C2.674,1.634,2.974,0.955,3.498,0.53z\"/>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side temperature-text\">Temperature</div>\n";

ptr +="<div class=\"side-by-side temperature\">";

ptr +=(int)TempCstat;

ptr +="<span class=\"superscript\">°C</span></div>\n";

ptr +="</div>\n";

and for water or humidity icon

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side humidity-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_2\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n\"; width=\"12px\" height=\"17.955px\" viewBox=\"0 0 13 17.955\" enable-background=\"new 0 0 13 17.955\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M1.819,6.217C3.139,4.064,6.5,0,6.5,0s3.363,4.064,4.681,6.217c1.793,2.926,2.133,5.05,1.571,7.057\n";

ptr +="c-0.438,1.574-2.264,4.681-6.252,4.681c-3.988,0-5.813-3.107-6.252-4.681C-0.313,11.267,0.026,9.143,1.819,6.217\"></path>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side humidity-text\">Humidity</div>\n";

ptr +="<div class=\"side-by-side humidity\">";

ptr +=(int)Humiditystat;

ptr +="<span class=\"superscript\">%</span></div>\n";

ptr +="</div>\n";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

## **Improvement 1 – Auto Page Refresh**

automatically refresh the page to update the sensor value.

You can instruct the browser to automatically reload the page at a specified interval by inserting a single <meta> tag into your HTML document.

<meta http-equiv="refresh" content="2" >

Insert this code into your document’s **<head>** tag; this meta tag will instruct the browser to refresh every two seconds.

## **Improvement 2 – Dynamically load Sensor Data with AJAX**

Refreshing a web page is impractical if the web page is large. [Asynchronous Javascript And Xml](https://en.wikipedia.org/wiki/Ajax_(programming)) (AJAX) is a better method because it allows us to request data from the server asynchronously (in the background) without refreshing the page.

The [XMLHttpRequest](https://en.wikipedia.org/wiki/XMLHttpRequest" \t "_blank) object within JavaScript is commonly used to execute AJAX on webpages. It sends a silent GET request to the server and updates the page element. AJAX is not a new technology, nor is it a different programming language; rather, it is simply an existing technology that is used in a different way. It allows you to:

* Request data from a server after the page has loaded
* Receive data from a server after the page has loaded
* Send data to a server in the background

This is the AJAX script we’ll be using. Place this script before closing the <head> tag.

ptr +="<script>\n";

ptr +="setInterval(loadDoc,200);\n";

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.getElementById(\"webpage\").innerHTML =this.responseText}\n";

ptr +="};\n";

ptr +="xhttp.open(\"GET\", \"/\", true);\n";

ptr +="xhttp.send();\n";

ptr +="}\n";

ptr +="</script>\n";

### **Code Explanation**

Because the AJAX script is just javascript, it must be enclosed in the **<script>** tag. We’ll use the javascript **setInterval()** function to make this function call itself repeatedly. It requires two parameters: the function to be executed and the time interval (in milliseconds).

ptr +="<script>\n";

ptr +="setInterval(loadDoc,200);\n";

A**loadDoc()** function is the most important part of this script. In this function, an object of **XMLHttpRequest()** is created. This object is used to query a web server for data.

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

The **xhttp.onreadystatechange()** method is called whenever the **readyState** changes. The **readyState** property represents the XMLHttpRequest’s status. It has one of the values listed below.

* 0: request not initialized
* 1: server connection established
* 2: request received
* 3: processing request
* 4: request finished, and response is ready

The status property holds the XMLHttpRequest object’s status. It has one of the values listed below.

* 200: “OK”
* 403: “Forbidden”
* 404: “Page not found.”

When **readyState** is 4 and status is 200, the response is complete, and the content of the element with id ‘webpage’ (div containing temperature and humidity values) is updated.

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.getElementById(\"webpage\").innerHTML =this.responseText}\n";

ptr +="};\n";

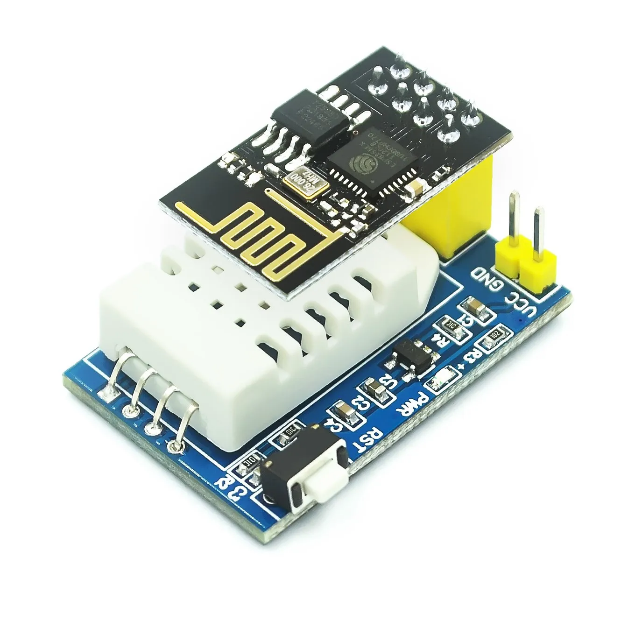
The **open()** and **send()** functions are then used to initiate the HTTP request.

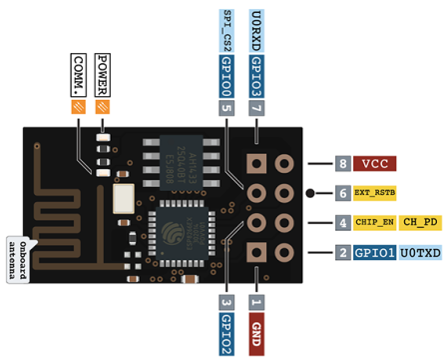
ptr +="xhttp.open(\"GET\", \"/\", true);\n";

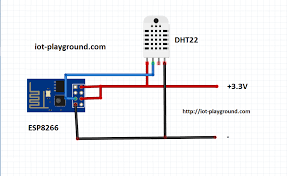
ptr +="xhttp.send();\n";

ptr +="}\n";

**Wiring DHT-11 Module with ESP-01 (ESP-8266)**

**** 





|  |  |
| --- | --- |
| **ESP-01 (Pin Number)** | **DHT-11 (Pin Number)** |
| GPIO2 | DATA/OUT/SIG |
| VCC | VCC |
| GND | GND |

Declare the pins normally as you declare in Arduino Uno/Nano or ESP8266 or other boards.

Rest remains the same as given above for ESP8266 Board.

**For Example :** if you are using GPIO2 , you need to write 2 for declaration.

REFRENCES

<https://lastminuteengineers.com/esp32-dht11-dht22-web-server-tutorial/#:~:text=Wiring%20a%20DHT11%2FDHT22%20sensor%20to%20an%20ESP32,-Connecting%20a%20DHT11&text=Connect%20the%20sensor's%20VCC%20pin,the%20sensor%20and%20the%20ESP32>.

<https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/>

<https://lastminuteengineers.com/dht11-module-arduino-tutorial/>