

ENVIRONMENTAL MONITORING USING **IOT**

PHASE 4 :Development part 2

GOAL: To design the platform to receive and display real-time temperature and humidity data from IOT devices.



INTRODUCTION:

Using a network of interconnected sensors and devices, **IoT (Internet of Things)**-based environmental monitoring collects, transmits, and analyzes data pertaining to various environmental factors. To better understand and manage the environment, this technology is extensively employed in numerous applications, including agriculture, smart communities, industrial operations, and conservation efforts.

- **Sensors:** Various sensors are typically used for environmental monitoring to measure parameters such as temperature, humidity, air quality, water quality, soil moisture, and light levels. These sensors are deployable in the field, on structures, and even on vehicles and drones.
- **Data Collection:** Sensors collect information continuously at predetermined intervals. The data is then transmitted for processing and analysis to a central node or cloud-based platform. For data transmission, wireless communication protocols such as Wi-Fi, cellular, LoRa WAN, and Zigbee are frequently employed.
- **Data Processing and Storage:** Sensor data is processed and stored on cloud platforms or local servers. Cloud platforms provide scalability and simple data access from anywhere, making them a popular option for IoT environmental monitoring.

- **Analytics:** Advanced analytics and machine learning algorithms may be applied to the collected data to generate insightful conclusions. For instance, predictive models can be constructed to predict environmental changes or detect anomalies.
- **Visualization:** The data can be presented to consumers via web-based dashboards, mobile applications, or other user interfaces. Researchers, government agencies, and the general public are able to comprehend environmental conditions and trends with the aid of visualization tools.
- **Alerts & Notification:** Environmental monitoring systems can be programmed to send alerts or notifications when predefined conditions are met or when anomalies are detected. This allows for swift response to critical situations, such as pollution increases and extreme weather events.

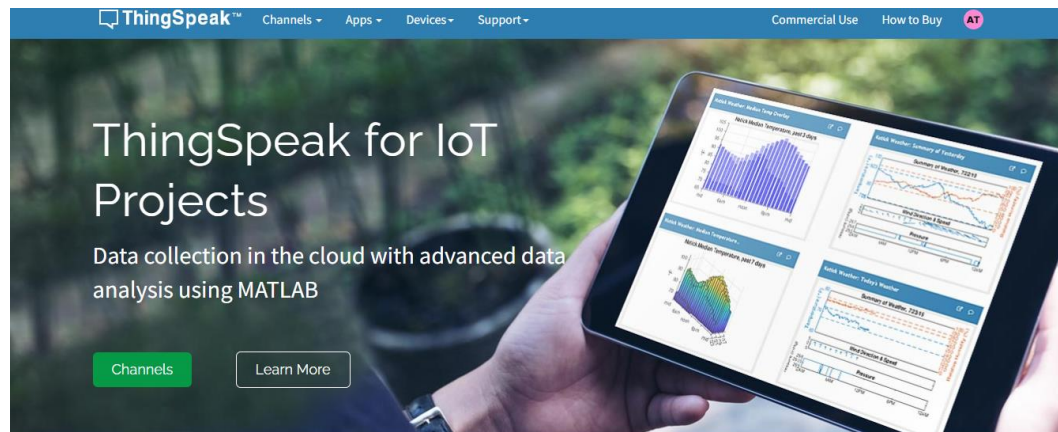
WOKWI :

Wokwi is an online simulator for Arduino, Raspberry Pi Pico, and ESP32 boards, or even your own custom microcontroller board designed to learn programming without the actual hardware.

Real-time temperature and humidity can be monitored using many methods here we use ThingSpeak and a Http Server.

1.THINGSPEAK:

ThingSpeak allows you to aggregate, visualize, and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices or equipment.



CREATE A CHANNEL :

ThingSpeak™

Channels

Apps

Devices

Support

Commercial Use

How to Buy

AT

ultrasonic

Channel ID: 2079754

Author: mwa000029642289

Access: Private

it is a temperature ,humidity and distance monitor

Private View

Public View

Channel Settings

Sharing

API Keys

Data Import / Export

Percentage complete

50%

Channel ID

2079754

Name

ultrasonic

Description

it is a temperature ,humidity monitor

Field 1

☐

Field 2

Temperature

☒

Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- Channel Name:** Enter a unique name for the ThingSpeak channel.
- Description:** Enter a description of the ThingSpeak channel.
- Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel has a maximum of 8 fields.

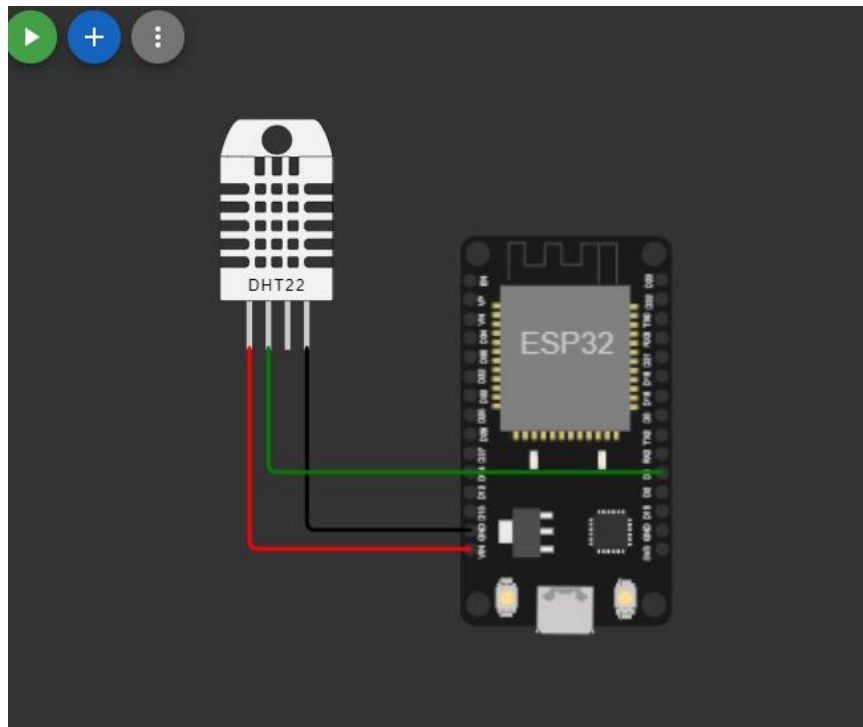
2.HTTP SERVER:

A HTTP or web server processes requests via HTTP, a network protocol used to exchange information on the World Wide Web (WWW). The main function of a HTTP server is to store, process and deliver web pages to clients.

Datacenter server needs temperature and humidity to be maintain in certain range. Precision air conditioner installed in Datacenter to maintain temperature and humidity. Desired temperature level in datacenter is 18 to 22 deg C. High temperature can cause breakdown of server. High humidity can cause condensation and failure or corrosion of equipment and too low humidity can cause Electrostatic discharge. With Environmental monitoring system connected with humidity & temperature monitoring sensor, user can set high and low values of humidity & temperature. In case room monitoring sensors detects high or low values continuously for more than set delay time then room monitoring system triggers alarm and send notification to listed users by Pop up message, Email, SMS or phone call.

CONNECT THE COMPONENTS:

Connect the ESP32 and DHT22 sensor as shown in the below figure :-



WRITE THE ARDUINO CODE :

1,Code for Thingspeak:

```
#include "DHTesp.h"
```

```
#include <ThingSpeak.h>
```

```
#include <WiFi.h>
```

```
DHTesp dhtsensor;
```

```
TempAndHumidity data;

const int pin = 15;

long dur;

unsigned long myChannelNumber = 2079754;

const char*myWriteAPIKey = "A51GV3A8YQDLEA7T";

int statusCode;

char ssid[] = "Wokwi-GUEST";

char pass[] = "";

WiFiClient client;

void setup()

{

  Serial.begin(115200);

  dhtsensor.setup(pin, DHTesp::DHT22);

  ThingSpeak.begin(client);

  WiFi.mode(WIFI_STA);

}

void loop()

{

  if(WiFi.status() != WL_CONNECTED)

  {

    Serial.print("Attempting to connect");

    while(WiFi.status() != WL_CONNECTED)

    {

      WiFi.begin(ssid,pass);

      for(int i=0;i<5;i++)

      {
```

```

    Serial.print(".");

    delay(1000);

}

}

Serial.println("\nConnected.");

}

data = dhtsensor.getTempAndHumidity();

Serial.println("Humi : " +String(data.humidity));

Serial.println("Temp : " +String(data.temperature));

Serial.println("_____");

ThingSpeak.setField(2,data.temperature);

ThingSpeak.setField(3,data.humidity);

statusCode = ThingSpeak.writeFields(myChannelNumber,myWriteAPIKey);

if(statusCode ==200)

Serial.println("Channel update successful.");

else

Serial.println("Problem Writing data. HTTP error code :"+String(statusCode));

delay(10000);

}

```


OUTPUT :

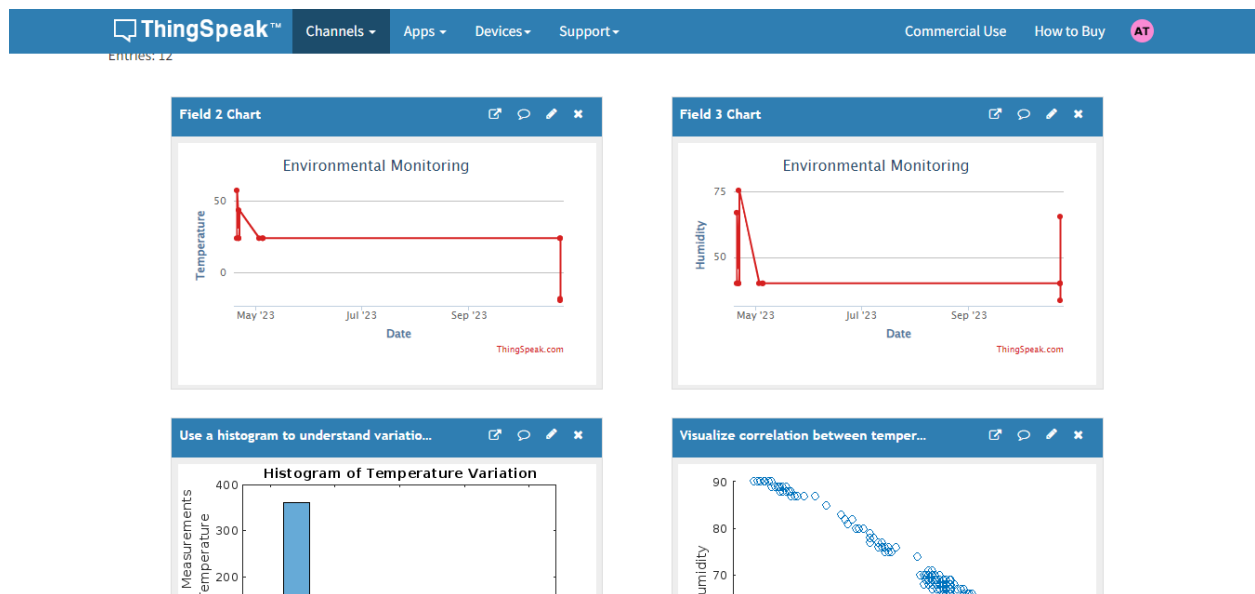
The screenshot shows the WOKWI web-based IDE. The left pane contains a C++ sketch for an ESP32 connected to a DHT22 sensor. The code includes libraries for DHTesp, ThingSpeak, and WiFi. It sets up a WiFi client and a DHT22 sensor on pin 15. The loop function checks for WiFi connection and prints data to the serial monitor. The right pane shows a simulation of the hardware with a DHT22 sensor and an ESP32 module. Below the simulation, a serial monitor displays the following output:

```
Temp : 24.10
Channel update successful.
Humi : 60.50
Temp : 13.40
Channel update successful.
```

ThingsSpeak output for the above code is :-

We can add different visualization graphs or plots for the given fields .

Eg: Histogram , Scatterplots



2. Code for http server :

```
#include <WiFi.h>

#include <Wire.h>

#include "DHT.h"

#define DHTPIN 4

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

// Replace with your network credentials

const char* ssid   = "Wokwi-GUEST"; //Your SSID

const char* password = ""; //Your Password

// Set web server port number to 80

WiFiServer server(80);

// Variable to store the HTTP request

String header;

void setup() {

    Serial.begin(9600);

    bool status;

    // default settings

    // (you can also pass in a Wire library object like &Wire2)

    //status = bme.begin();

    // Connect to Wi-Fi network with SSID and password

    Serial.print("Connecting to ");

    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {

        delay(500);

        Serial.print(".");

    }

}
```

```

// Print local IP address and start web server

Serial.println("");

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

server.begin();

}

void loop(){

    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,

        // Reading temperature or humidity takes about 250 milliseconds!

        // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)

        float h = dht.readHumidity();

        // Read temperature as Celsius (the default)

        float t = dht.readTemperature();

        // Read temperature as Fahrenheit (isFahrenheit = true)

        float f = dht.readTemperature(true);

        while(isnan(h) || isnan(t) || isnan(f)){

            delay(5000);

            h = dht.readHumidity();

            t = dht.readTemperature();

            f = dht.readTemperature(true);

        }

        // Compute heat index in Fahrenheit (the default)

        float hif = dht.computeHeatIndex(f, h);

        // Compute heat index in Celsius (isFahreheit = false)

        float hic = dht.computeHeatIndex(t, h, false);

```

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

```
Serial.print(h);
```

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

```
Serial.print(t);
```

```
Serial.print(F("°C "));
```

```
Serial.print(f);
```

```
Serial.print(F("°F Heat index: "));
```

```
Serial.print(hic);
```

```
Serial.print(F("°C "));
```

```
Serial.print(hif);
```

```
Serial.println(F("°F"));
```

```
Serial.println("New Client.");    // print a message out in the serial port
```

```
String currentLine = "";        // make a String to hold incoming data from the client
```

```
while (client.connected()) {    // loop while the client's connected
```

```
  if (client.available()) {      // if there's bytes to read from the client,
```

```
    char c = client.read();       // read a byte, then
```

```
    Serial.write(c);              // print it out the serial monitor
```

```
    header += c;
```

```
    if (c == '\n') {              // if the byte is a newline character
```

```
      // if the current line is blank, you got two newline characters in a row.
```

```
      // that's the end of the client HTTP request, so send a response:
```

```
      if (currentLine.length() == 0) {
```

```
        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
```

```
        // and a content-type so the client knows what's coming, then a blank line:
```

```
        client.println("HTTP/1.1 200 OK");
```

```
        client.println("Content-type:text/html");
```

```
        client.println("Connection: close");
```

```
client.println();
```

```
// Display the HTML web page
```

```
client.println("<!DOCTYPE html><html>");
```

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
```

```
client.println("<link rel=\"icon\" href=\"data:;\">");
```

```
// CSS to style the table
```

```
client.println("<style>body { text-align: center; font-family: \"Trebuchet MS\", Arial; }");
```

```
client.println("table { border-collapse: collapse; width:35%; margin-left:auto; margin-right:auto; }");
```

```
client.println("th { padding: 12px; background-color: #0043af; color: white; }");
```

```
client.println("tr { border: 1px solid #ddd; padding: 12px; }");
```

```
client.println("tr:hover { background-color: #bcbcbc; }");
```

```
client.println("td { border: none; padding: 12px; }");
```

```
client.println(".sensor { color:white; font-weight: bold; background-color: #bcbcbc; padding: 1px; }");
```

```
// Web Page Heading
```

```
client.println("</style></head><body><h1>ESP32 with DHT11</h1>");
```

```
client.println("<table><tr><th>MEASUREMENT</th><th>VALUE</th></tr>");
```

```
client.println("<tr><td>Temp. Celsius</td><td><span class=\"sensor\">");
```

```
client.println(t);
```

```
client.println(" *C</span></td></tr>");
```

```
client.println("<tr><td>Temp. Fahrenheit</td><td><span class=\"sensor\">");
```

```
client.println(f);
```

```
client.println(" *F</span></td></tr>");
```

```
client.println("<tr><td>Humidity</td><td><span class=\"sensor\">");
```

```
client.println(h);
```

```
client.println(" %</span></td></tr>");
```

```
client.println("<tr><td>Heat Index (Celcius)</td><td><span class=\"sensor\">");
```

```

    client.println(hic);

    client.println(" *C</span></td></tr>");

    client.println("<tr><td>Heat Index (Fahrenheit)</td><td><span class=\"sensor\">");

    client.println(hif);

    client.println(" *F</span></td></tr>");

    client.println("</body></html>");


    // The HTTP response ends with another blank line

    client.println();

    // Break out of the while loop

    break;

} else { // if you got a newline, then clear currentLine

    currentLine = "";

}

} else if (c != '\r') { // if you got anything else but a carriage return character,

    currentLine += c;    // add it to the end of the currentLine

}

}

}

// Clear the header variable

header = "";

// Close the connection

client.stop();

Serial.println("Client disconnected.");

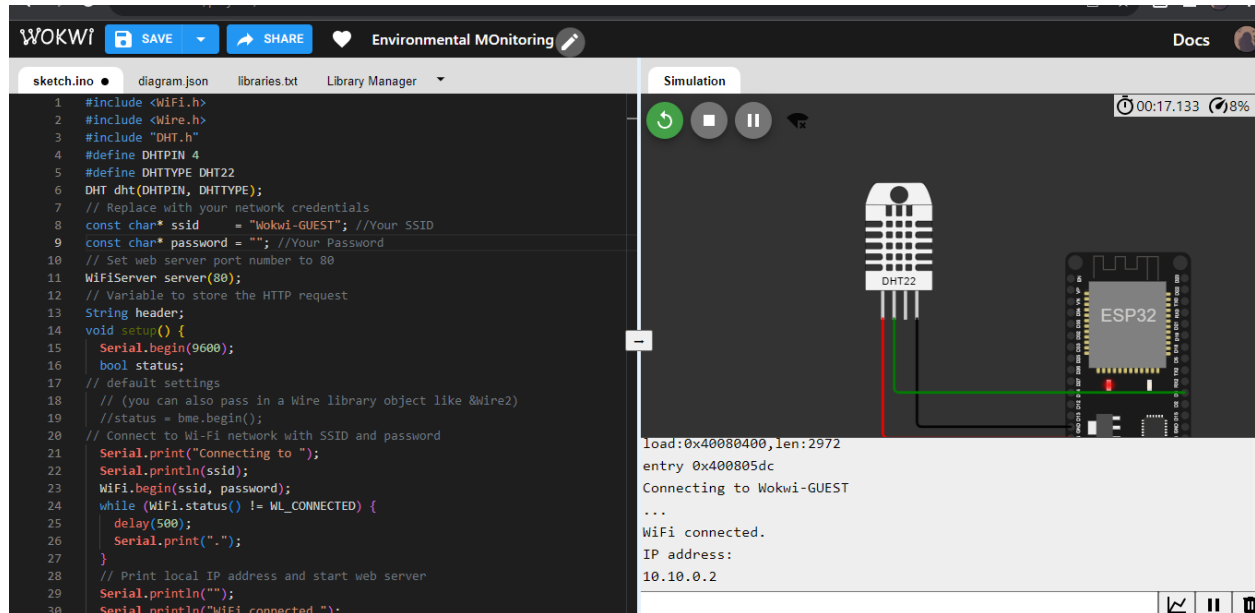
Serial.println("");

}

}

```

OUTPUT :



```
1 #include <WiFi.h>
2 #include <Wire.h>
3 #include "DHT.h"
4 #define DHTPIN 4
5 #define DHTTYPE DHT22
6 DHT dht(DHTPIN, DHTTYPE);
7 // Replace with your network credentials
8 const char* ssid = "Wokwi-GUEST"; //Your SSID
9 const char* password = ""; //Your Password
10 // Set web server port number to 80
11 WiFiServer server(80);
12 // Variable to store the HTTP request
13 String header;
14 void setup() {
15   Serial.begin(9600);
16   bool status;
17   // default settings
18   // (you can also pass in a Wire library object like &Wire2)
19   //status = bme.begin();
20   // Connect to Wi-Fi network with SSID and password
21   Serial.print("Connecting to ");
22   Serial.println(ssid);
23   WiFi.begin(ssid, password);
24   while (WiFi.status() != WL_CONNECTED) {
25     delay(500);
26     Serial.print(".");
27   }
28   // Print local IP address and start web server
29   Serial.println("");
30   Serial.println("Wifi connected.");
```

Simulation

load:0x40080400,len:2972
entry 0x400805dc
Connecting to Wokwi-GUEST
...
Wifi connected.
IP address:
10.10.0.2

- Copy the Ip address given on the wokwi output and paste it on any browser
- It will display the temperature and humidity values from the wokwi.
- By this technology anyone can able to monitor the environment from any part of the world.
- The following is the example of the readings which are displayed on the http webserver . This can be accessed on the pc or even on your mobiles phones .

ESP32 with DHT

MEASUREMENT	VALUE
Temp. Celsius	15.30 °C
Temp. Fahrenheit	59.54 °F
Humidity	132.00 %
Heat Index (Celcius)	16.33 °C
Heat Index (Fahrenheit)	61.40 °F

CONCLUSION :

The Internet of Things (IoT) has begun to emerge as it has helped human in controlling and monitoring essential conditions using devices which are able to capture, evaluate, and transmit information from the environment to the cloud, where the data will be stored. Environment monitoring system is one of the most important IoT systems which mainly includes data collections through sensors and data reviewing for short-term measure as well as remote management and observations

We can use this concept to publish our sensor data and see it from our local network. With a real server and proper internet connection, we can use access it from anywhere. We could also use multiple sensors to make the data more accurate.