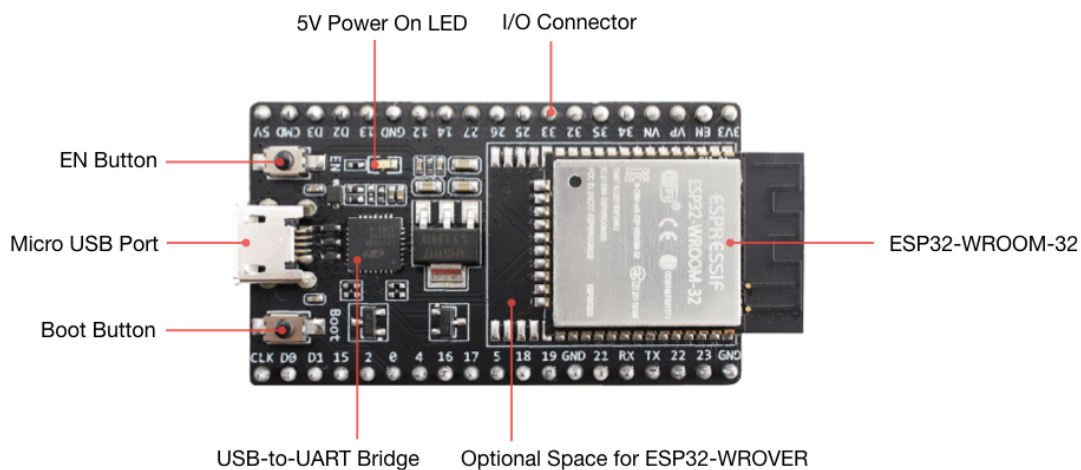


Part B

ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the clock frequency is adjustable from 80 MHz to 240 MHz. The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S and I2C.

- The ESP32 is dual core.
- It has Wi-Fi and Bluetooth built-in.
- It runs 32-bit programs.
- The clock frequency can go up to 240MHz and it has a 512 kB RAM.
- This particular board has 30 or 36 pins, 15 in each row.
- It also has wide variety of peripherals available, like: capacitive touch, ADCs, DACs, UART, SPI, I2C and much more.
- It comes with built-in hall effect sensor and built-in temperature sensor.



Key Component	Description
ESP32-WROOM-32	A module with ESP32 at its core. For more information
EN	Reset button.
Boot	Download button. Holding down Boot and then pressing EN initiates

	Firmware Download mode for downloading firmware through the serial port.
USB-to-UART Bridge	Single USB-UART bridge chip provides transfer rates of up to 3 Mbps.
Micro USB Port	USB interface. Power supply for the board as well as the communication interface between a computer and the ESP32-WROOM-32 module.
5V Power On LED	Turns on when the USB or an external 5V power supply is connected to the board.
I/O	Most of the pins on the ESP module are broken out to the pin headers on the board. You can program ESP32 to enable multiple functions such as PWM, ADC, DAC, I2C, I2S, SPI, etc.

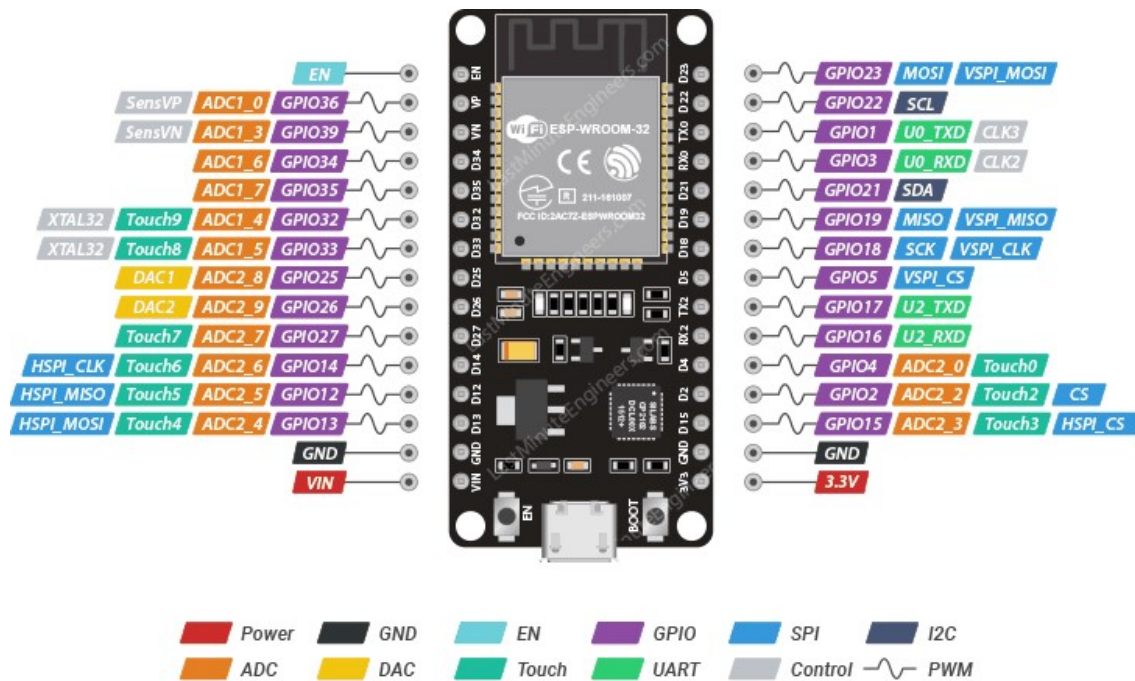
Note:The pins D0, D1, D2, D3, CMD and CLK are used internally for communication between ESP32 and SPI flash memory. They are grouped on both sides near the USB connector. Avoid using these pins, as it may disrupt access to the SPI flash memory / SPI RAM.

Features of the ESP32 include the following:

- **Processors:**
 - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
 - Ultra-low power (ULP) co-processor
- **Memory:** 320 KiB RAM, 448 KiB ROM
- **Wireless connectivity:**
 - Wi-Fi: 802.11 b/g/n
 - Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- **Peripheral interfaces:**
 - 34 × programmable GPIOs
 - 12-bit SAR ADC up to 18 channels
 - 2 × 8-bit DACs
 - 10 × touch sensors (capacitive sensing GPIOs)
 - 4 × SPI
 - 2 × I²S interfaces
 - 2 × I²C interfaces
 - 3 × UART
 - SD/SDIO/CE-ATA/MMC/eMMC host controller
 - SDIO/SPI slave controller
 - Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time Protocol support[4]
 - CAN bus 2.0
 - Infrared remote controller (TX/RX, up to 8 channels)
 - Motor PWM
 - LED PWM (up to 16 channels)
 - Hall effect sensor

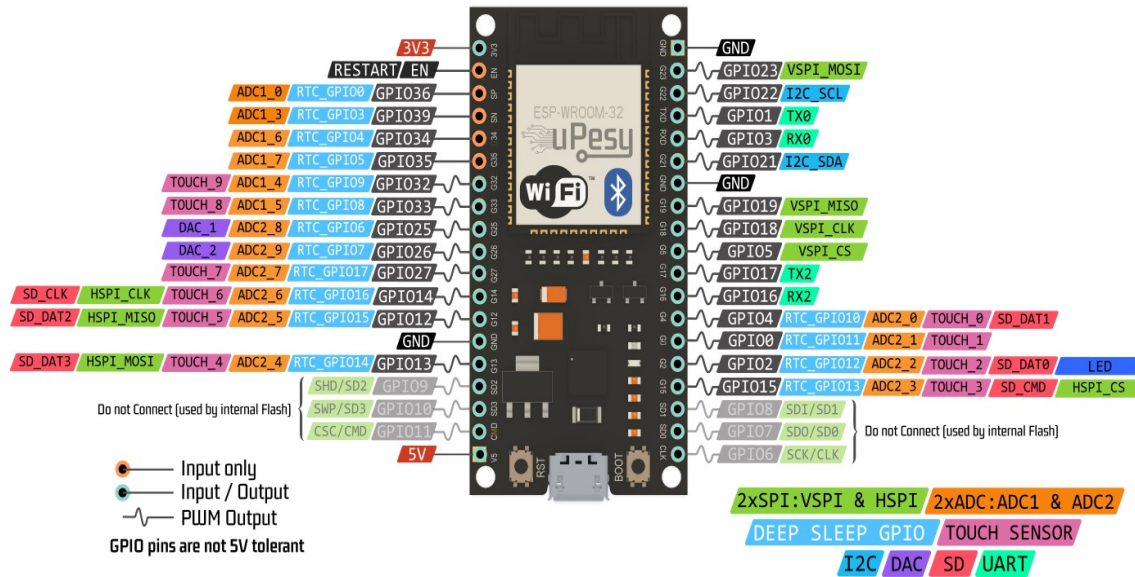
- Ultra-low power analog pre-amplifier
- **Security:**
 - IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on version)[5] and WLAN Authentication and Privacy Infrastructure (WAPI)
 - Secure boot
 - Flash encryption
 - 1024-bit OTP, up to 768-bit for customers
 - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
- **Power management:**
 - Internal low-dropout regulator
 - Individual power domain for RTC
 - 5 μ A deep sleep current
 - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Pin Layout:

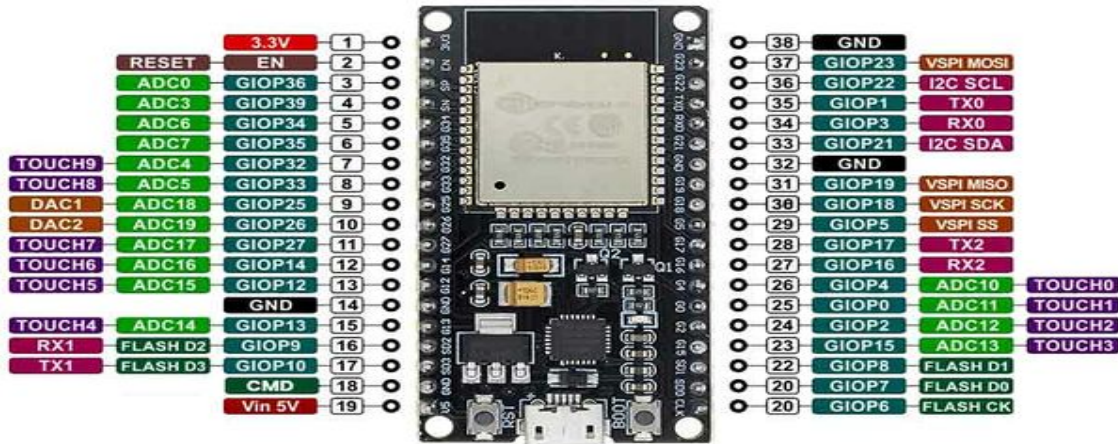


ESP32 Dev. Board Pinout

ESP32 Wroom DevKit Full Pinout



PINOUT ESP32 38 PINES ESP WROOM 32



For more details : <https://www.upesv.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide>

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Install the Arduino Desktop IDE:

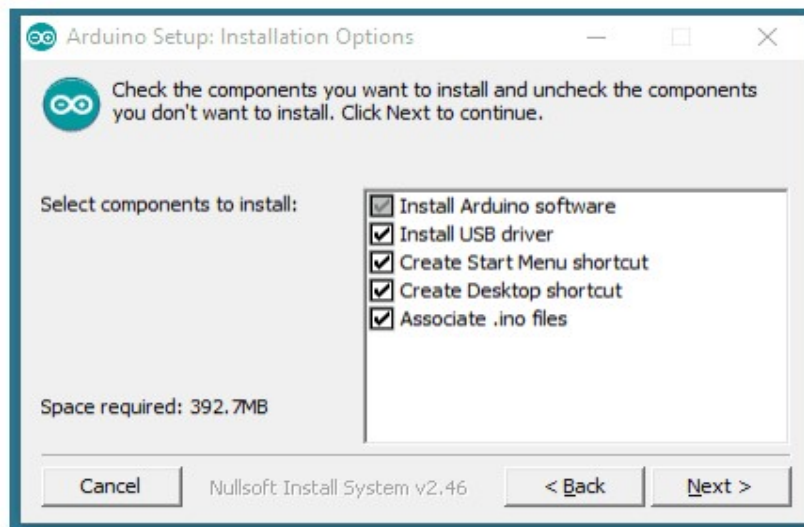
Arduino IDE Installation (Windows):

Step 1: Download the Arduino Software (IDE):

<https://www.arduino.cc/en/software>

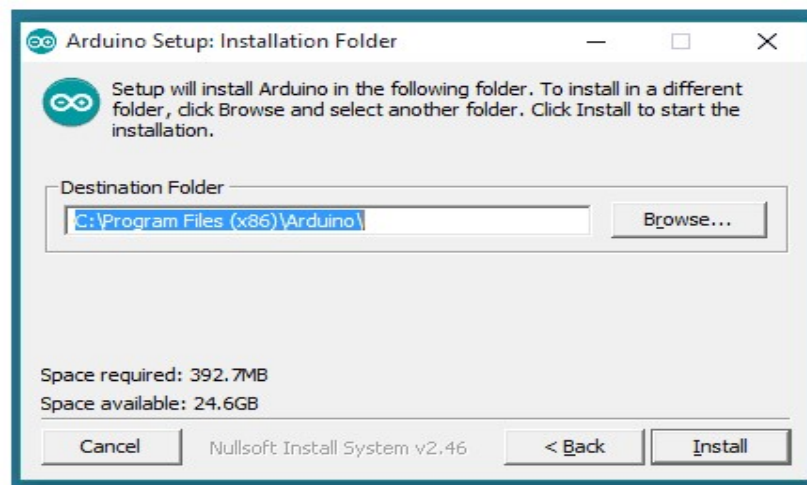
Step 2:

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.



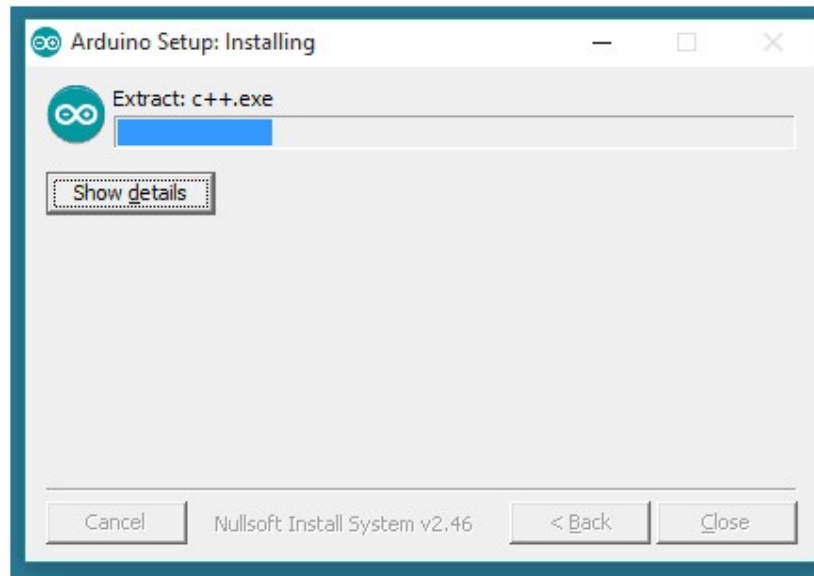
Choose the components to install.

Step 3:



Choose the installation directory.

Step 4:



Installation in progress.

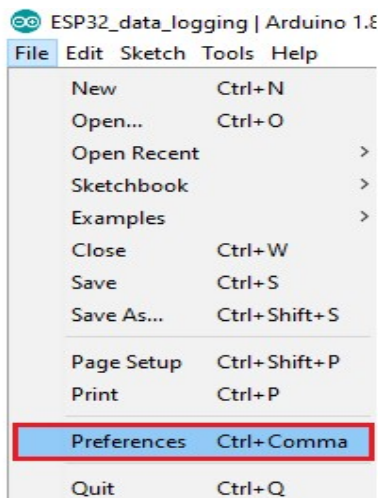
The process will extract and install all the required files to execute properly the Arduino Software (IDE)

Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language.

To install the ESP32 board in your Arduino IDE, follow these Steps:

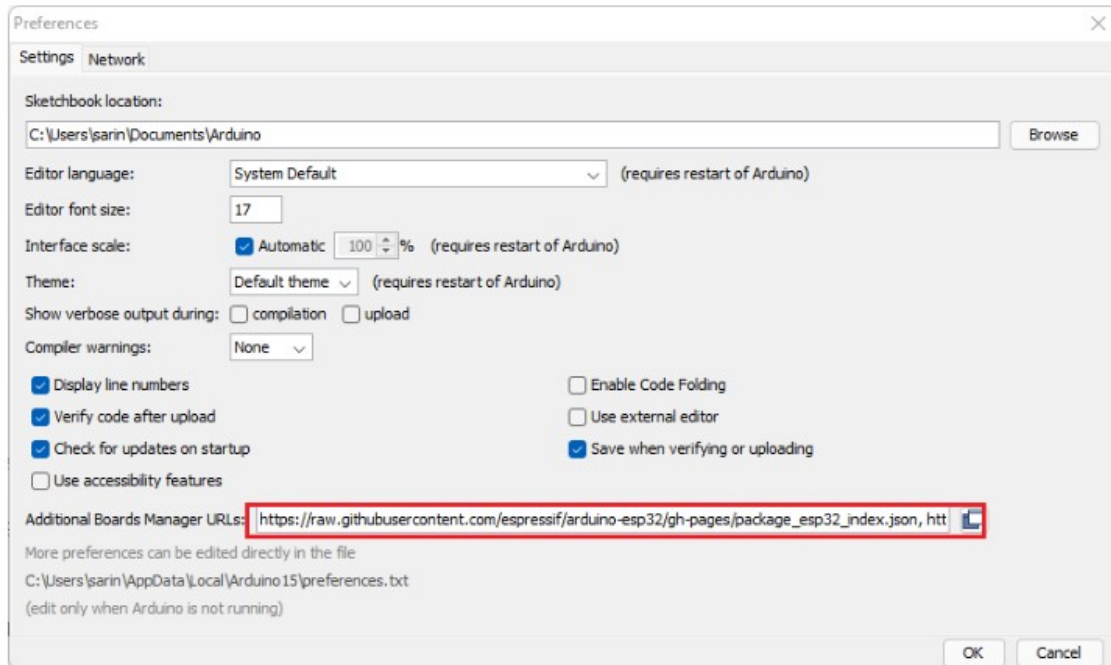
Step 1: In your Arduino IDE, go to File> Preferences



Step 2: Enter the following into the “Additional Board Manager URLs” field:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

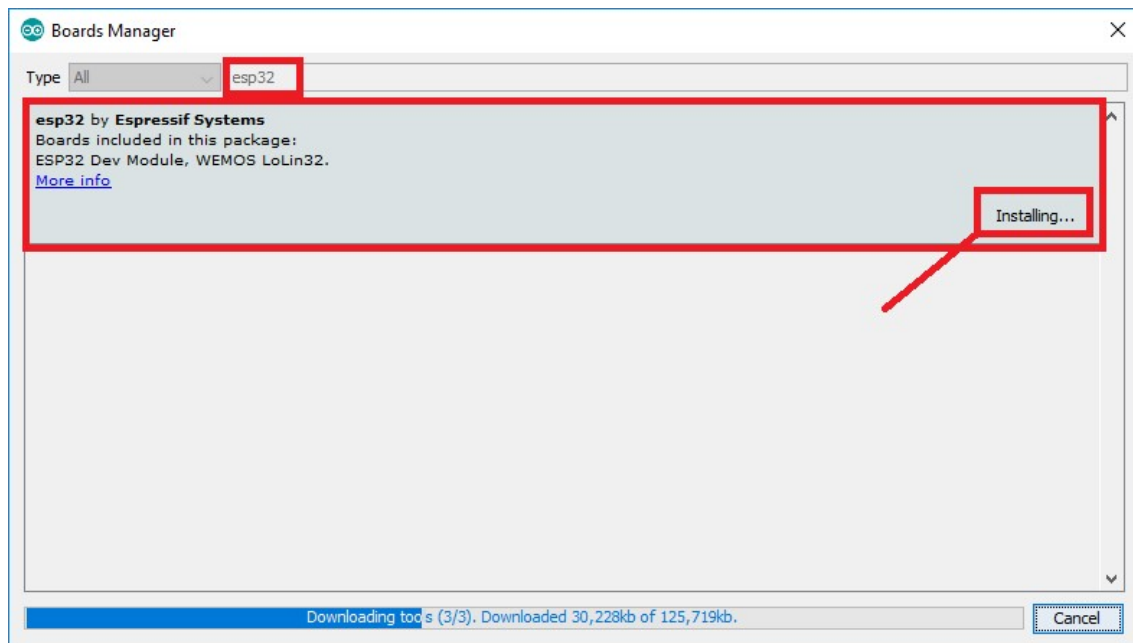
Then, click the “OK” button:



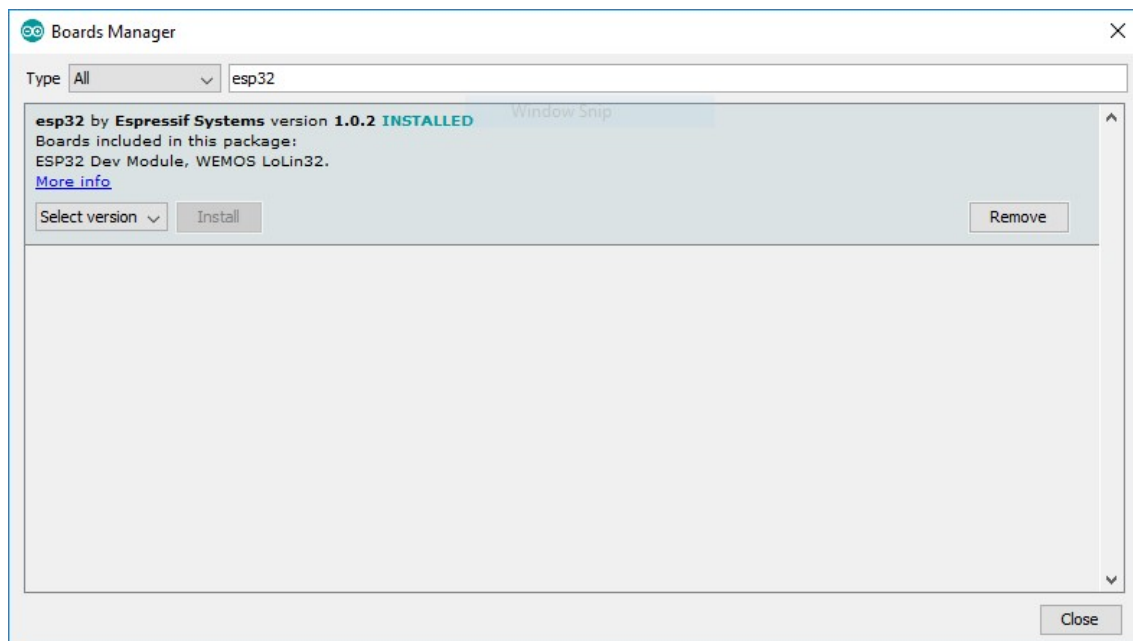
Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Step 3: Open the Boards Manager. **Go to Tools > Board > Boards Manager...**



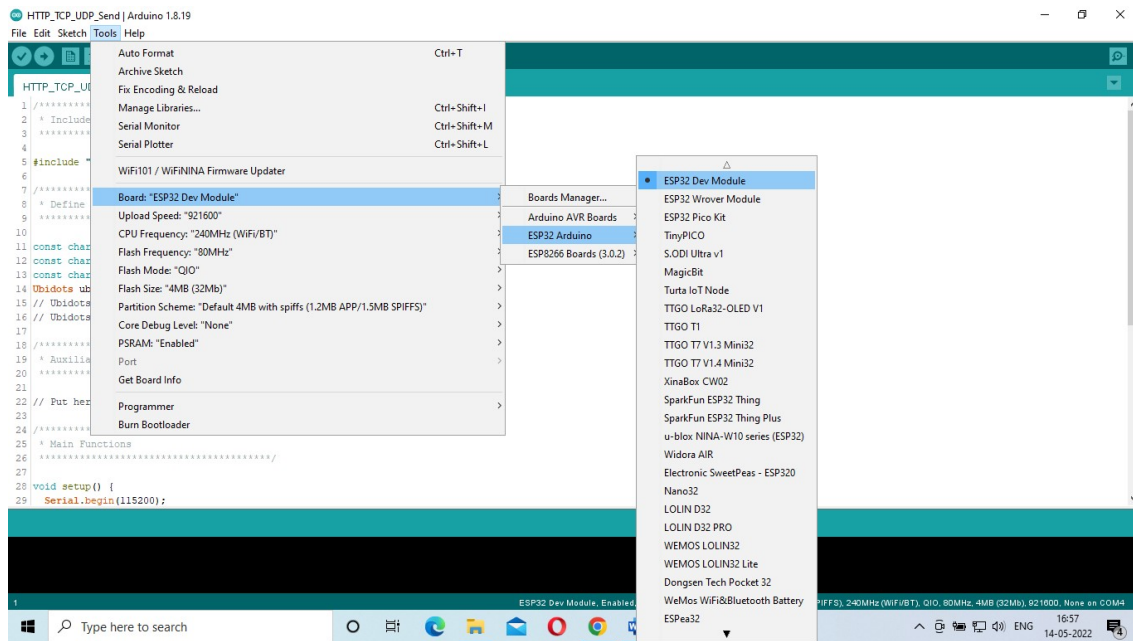
Step 5: That's it. It should be installed after a few seconds.



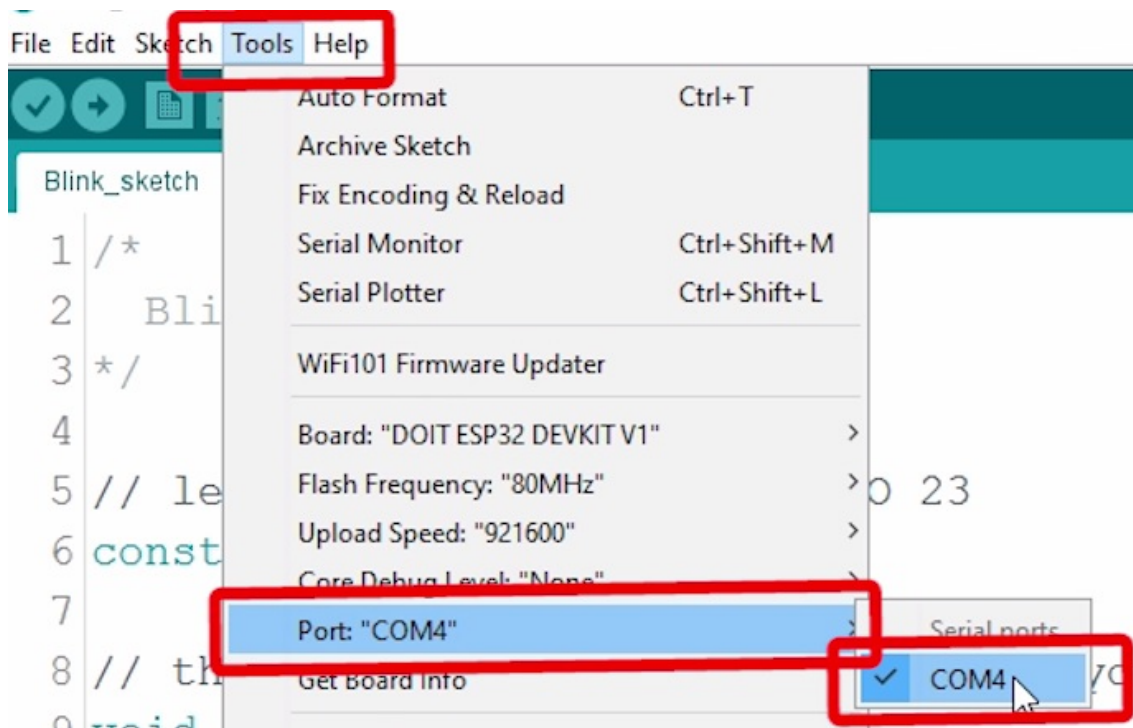
Testing the Installation:

Plug the ESP32 board to your computer. With your Arduino IDE open, follow these steps:

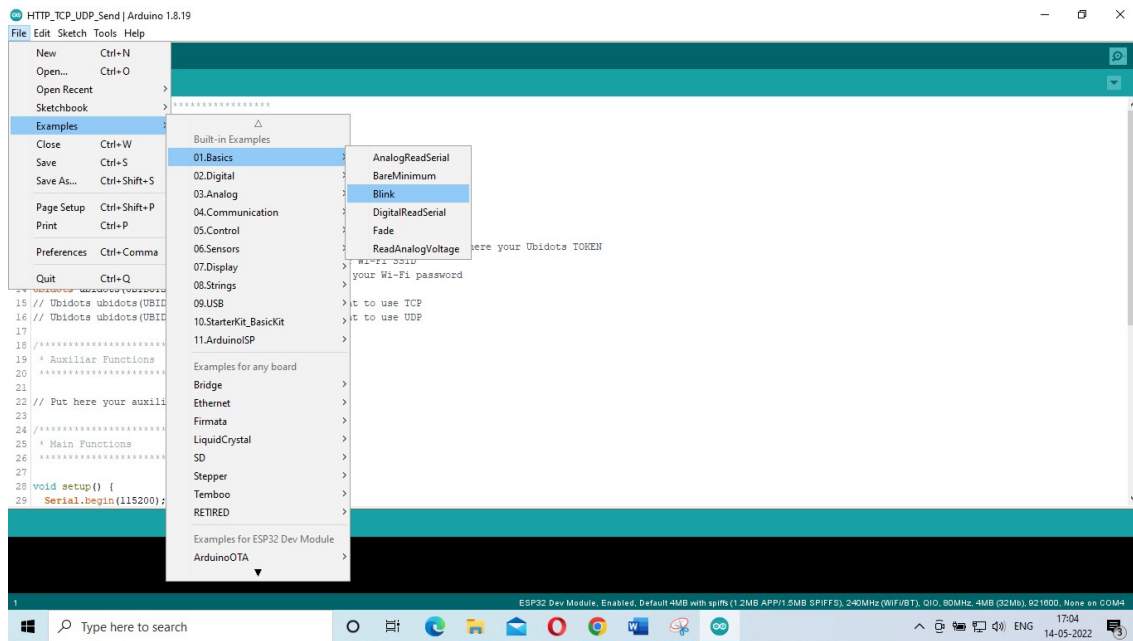
Step 1: Select your Board in Tools > Board menu (in my case it's the "ESP32 Dev Module")



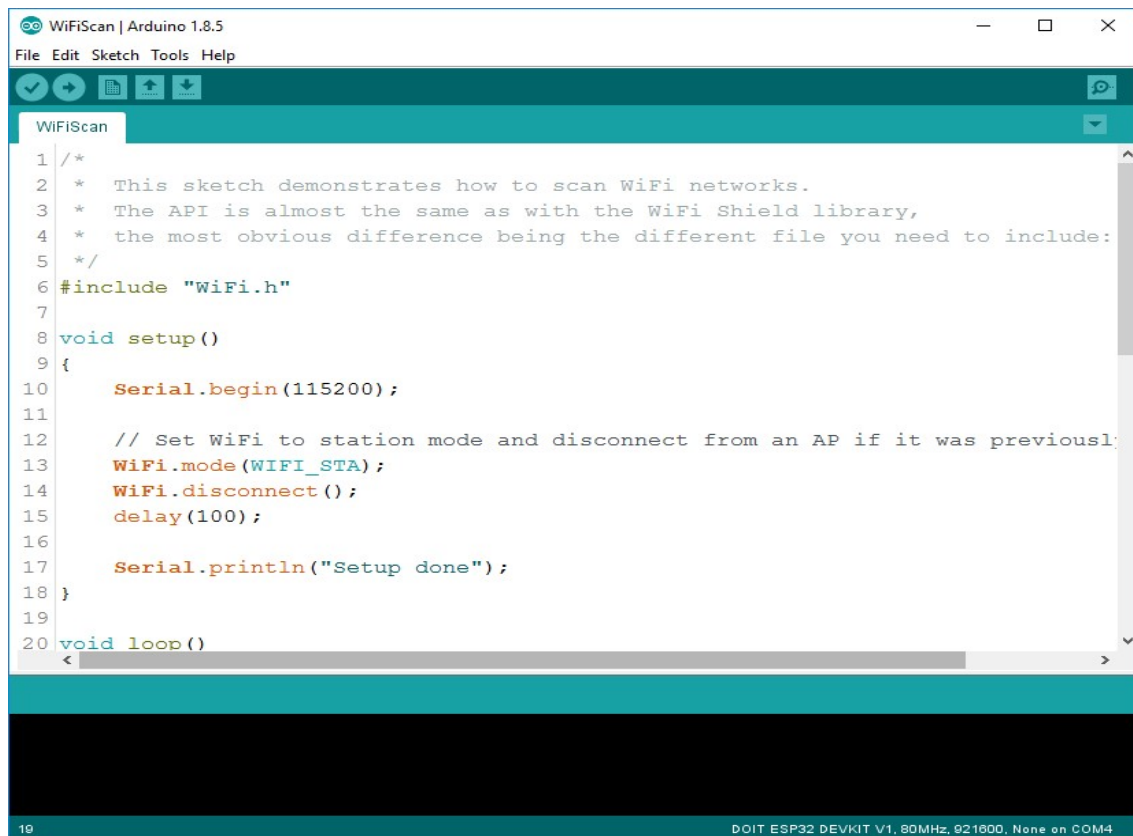
Step 2: Select the Port (if you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers):



Step 3: Open the following example under **File > Examples > Basics > Blink**



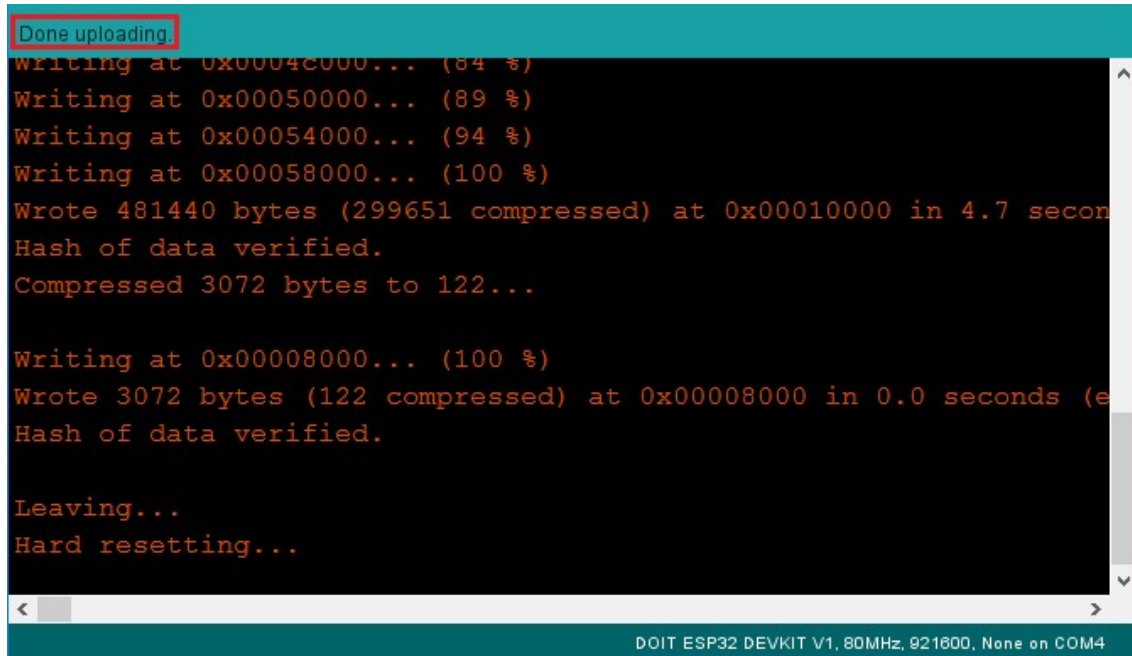
Step 4: A new sketch opens in your Arduino IDE:



Step 5: Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.



Step 6: If everything went as expected, you should see a “Done uploading.” message.



```
Done uploading.
Writing at 0x00042000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e
Hash of data verified.

Leaving...
Hard resetting...

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4
```

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

Experiment -1

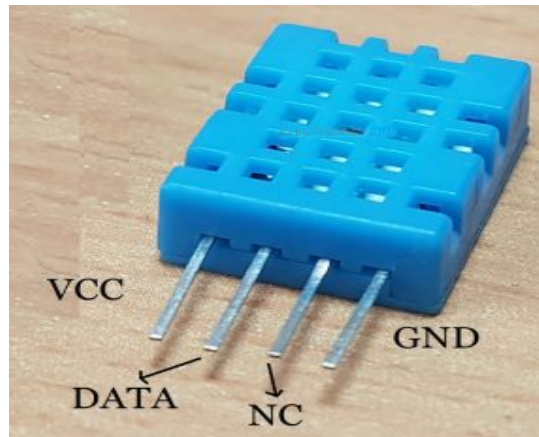
Aim: To measure the Temperature, Humidity and Heat index using ESP32 Development Board and display the values in serial monitor.

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	DHT11 or DHT22 temperature and humidity sensor	1
3	Jumper wires	3
4	Micro USB cable	1

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity.

- Temperature range: 0 to 50°C +/- 2°C
- Relative humidity range: 20 to 90% +/-5%
- Temperature resolution: 1°C
- Humidity resolution: 1%
- Operating voltage: 3 to 5.5 V DC
- Current supply: 0.5 to 2.5 mA
- Sampling period: 1 second



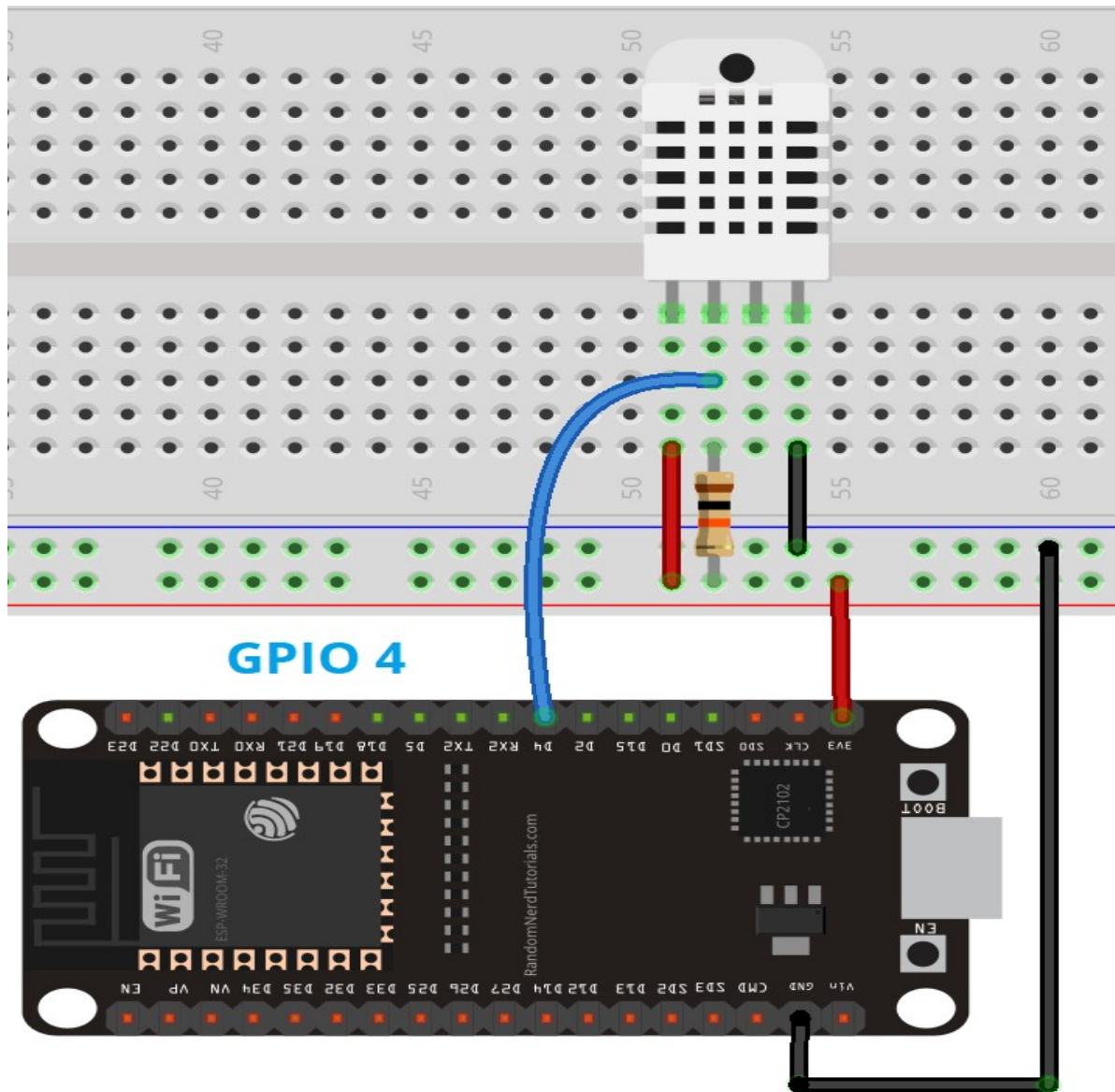


Figure.1 Circuit Diagram for measurement of Temperature and Humidity.

PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
 Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include "DHT.h"

#define DHTPIN 4 // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
#define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
// #define DHTTYPE DHT21 // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors. This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // 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);

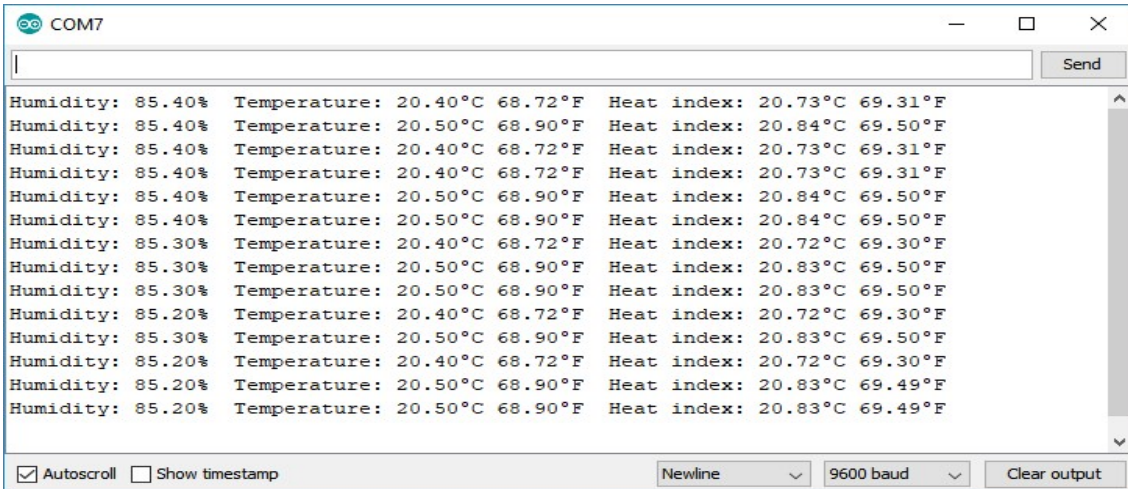
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
Serial.println(F("Failed to read from DHT sensor!"));
    return;
}

// 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("\xC2\xB0 C, "));
Serial.print(f);
Serial.print(F("\xC2\xB0 F, Heat index: "));
Serial.print(hic);
Serial.print(F("\xC2\xB0 C, "));
Serial.print(hif);
Serial.println(F("\xC2\xB0 F."));
}

```

Result:



```

COM7
Humidity: 85.40% Temperature: 20.40°C 68.72°F Heat index: 20.73°C 69.31°F
Humidity: 85.40% Temperature: 20.50°C 68.90°F Heat index: 20.84°C 69.50°F
Humidity: 85.40% Temperature: 20.40°C 68.72°F Heat index: 20.73°C 69.31°F
Humidity: 85.40% Temperature: 20.40°C 68.72°F Heat index: 20.73°C 69.31°F
Humidity: 85.40% Temperature: 20.50°C 68.90°F Heat index: 20.84°C 69.50°F
Humidity: 85.40% Temperature: 20.50°C 68.90°F Heat index: 20.84°C 69.50°F
Humidity: 85.30% Temperature: 20.40°C 68.72°F Heat index: 20.72°C 69.30°F
Humidity: 85.30% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.50°F
Humidity: 85.30% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.50°F
Humidity: 85.20% Temperature: 20.40°C 68.72°F Heat index: 20.72°C 69.30°F
Humidity: 85.20% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.49°F
Humidity: 85.20% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.49°F
Humidity: 85.20% Temperature: 20.40°C 68.72°F Heat index: 20.72°C 69.30°F
Humidity: 85.20% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.49°F
Humidity: 85.20% Temperature: 20.50°C 68.90°F Heat index: 20.83°C 69.49°F

```

☒ Autoscroll ☐ Show timestamp Newline 9600 baud Clear output

Experiment -2

2 a: Scan Wi-Fi networks and get Wi-Fi strength using ESP32 Development board.

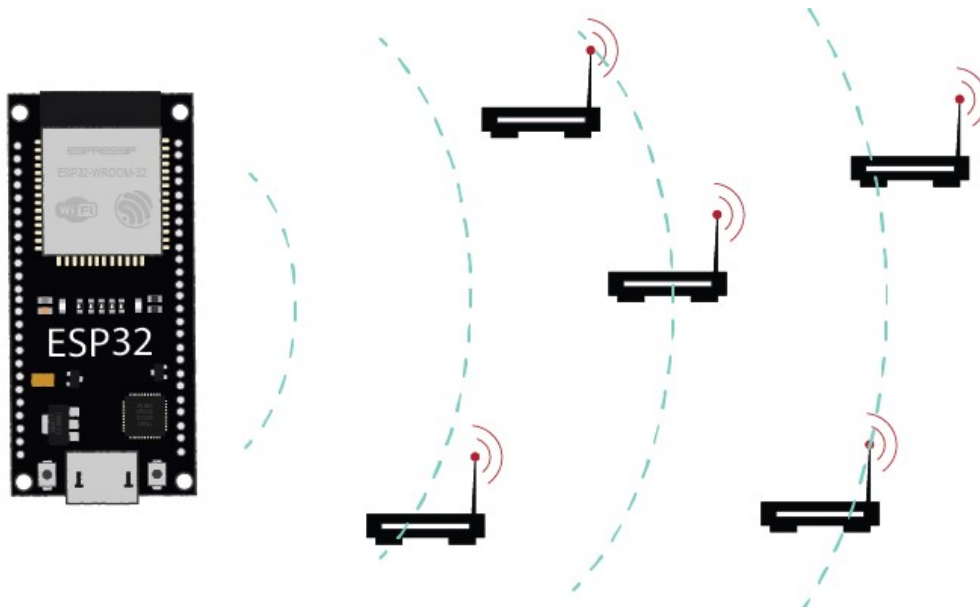
ESP32 Wi-Fi Modes:

The ESP32 board can act as Wi-Fi Station, Access Point or both. To set the Wi-Fi mode, use `WiFi.mode()` and set the desired mode as argument:

<code>WiFi.mode(WIFI_STA)</code>	station mode: the ESP32 connects to an access point
<code>WiFi.mode(WIFI_AP)</code>	access point mode: stations can connect to the ESP32
<code>WiFi.mode(WIFI_STA_AP)</code>	access point and a station connected to another access point

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32Development Board	1
2	Micro USB cable	1



PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory

Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**

4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include "WiFi.h"

void setup()
{
  Serial.begin(115200);

  // Set WiFi to station mode and disconnect from an AP if it was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");
}

void loop()
{
  Serial.println("scan start");

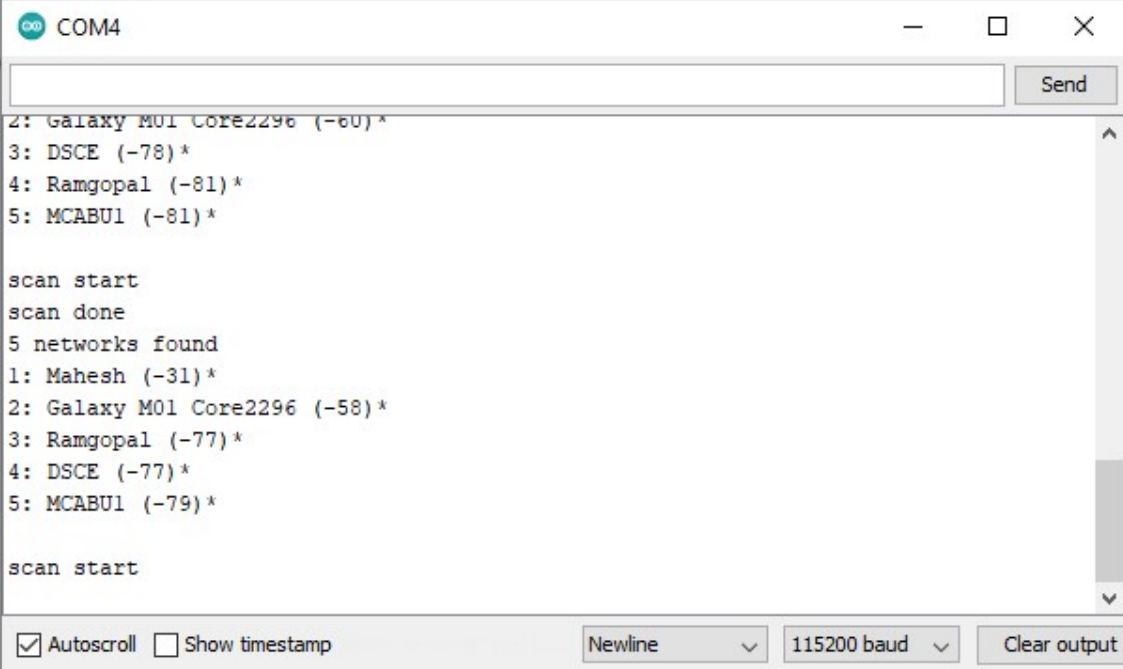
  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();
  Serial.println("scan done");
  if (n == 0) {
    Serial.println("no networks found");
  } else {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
    }
  }
}
```



```
delay(10);
    }
}
Serial.println("");

    // Wait a bit before scanning again
delay(5000);
}
```

Result:



```
COM4
2: Galaxy M01 Core2296 (-60)*
3: DSCE (-78)*
4: Ramgopal (-81)*
5: MCABU1 (-81)*

scan start
scan done
5 networks found
1: Mahesh (-31)*
2: Galaxy M01 Core2296 (-58)*
3: Ramgopal (-77)*
4: DSCE (-77)*
5: MCABU1 (-79)*

scan start
```

☒ Autoscroll ☐ Show timestamp Newline 115200 baud Clear output

2 b: Configure/Set Your ESP32 Development board as an access point

When you set your ESP32 board as an access point, you can be connected using any device with Wi-Fi capabilities without connecting to your router. When you set the ESP32 as an access point, you create its own Wi-Fi network, and nearby Wi-Fi devices (stations) can connect to it, like your smartphone or computer. So, you don't need to be connected to a router to control it.



PROCEDURE:

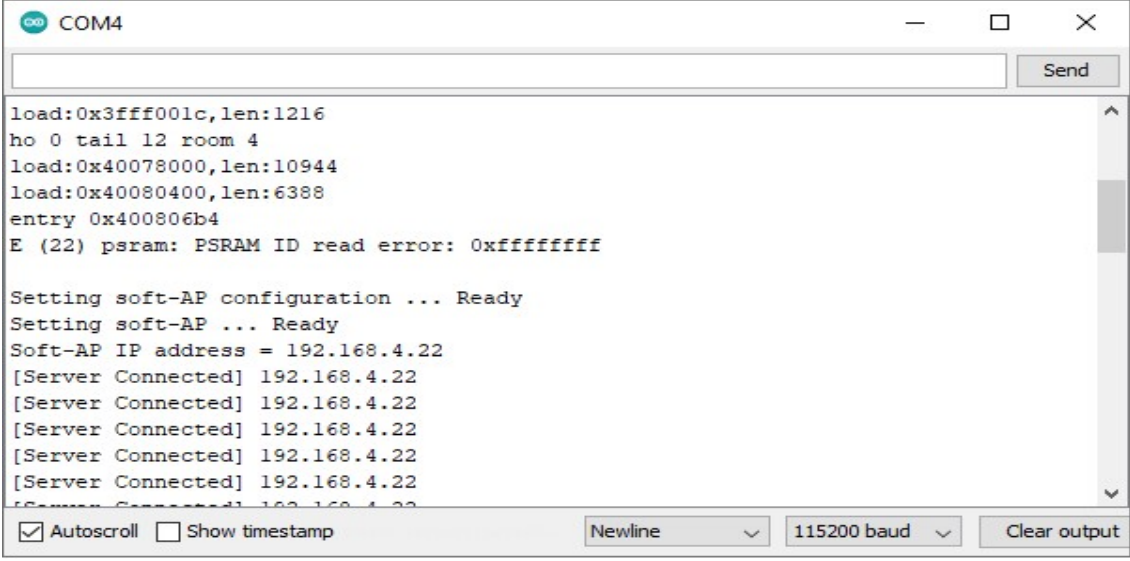
1. Plug the ESP32 development board to your PC
2. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**
3. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include <WiFi.h>
const char *ssid = "SSID_Name_your_Choice";
const char *password = "Your_Choice(min 6 character) ";
IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
IPAddress subnet(255,255,255,0);
void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.print("Setting soft-AP configuration ... ");
  Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
  Serial.print("Setting soft-AP ... ");
  Serial.println(WiFi.softAP(ssid,password) ? "Ready" : "Failed!");
  //WiFi.softAP(ssid);
  //WiFi.softAP(ssid, password, channel, ssid_hidden, max_connection)

  Serial.print("Soft-AP IP address = ");
  Serial.println(WiFi.softAPIP());
}
void loop() {
  Serial.print("[Server Connected] ");
  Serial.println(WiFi.softAPIP());
  delay(500);
}
```

Result:

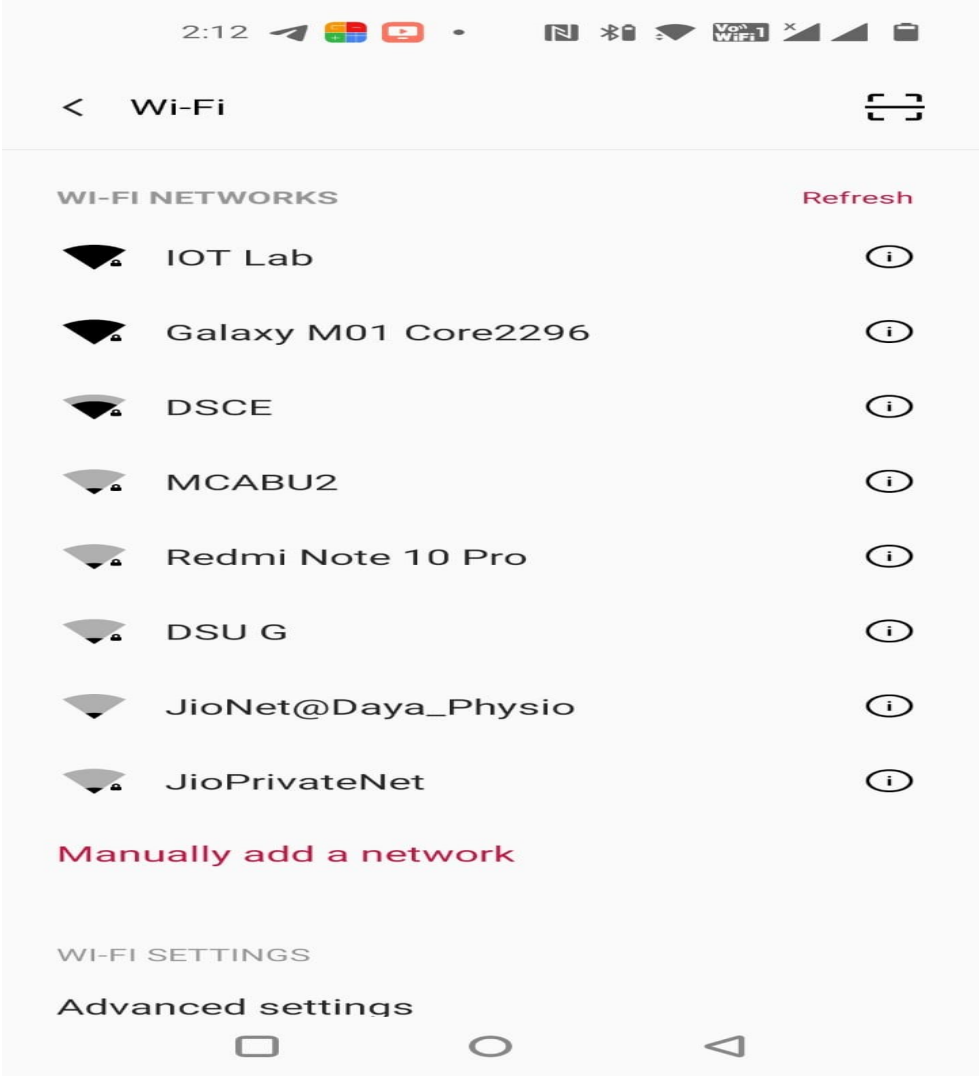


```
load:0x3fff001c,len:1216
no 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff

Setting soft-AP configuration ... Ready
Setting soft-AP ... Ready
Soft-AP IP address = 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
```

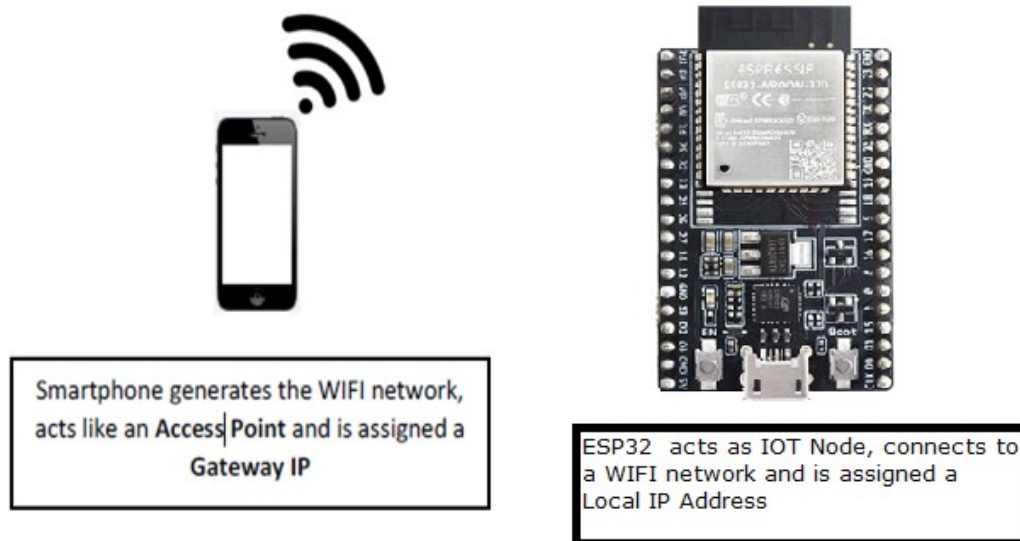
☒ Autoscroll ☐ Show timestamp Newline 115200 baud Clear output

Through Smart phone:



2 c: Establish connection between IoT Node and Access point and Display of local IP and Gateway IP

The goal of this experiment is to understand the configuration of IoT Node to connect to an AccessPoint. Any device when connected to a network is assigned an IP address, which is a unique number for each device on a network. We shall also see how to identify the IP network of our IoT Node. The Access Point also has an IP address, known as the Gateway IP. These concepts are fundamentals to understand the architecture of an IoT Network.



PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**
3. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include <WiFi.h>

char ssid[]="REPLACE_WITH_YOUR_SSID";
char password[]="REPLACE_WITH_YOUR_PASSWORD";

IPAddress ip;
IPAddress gateway;

void setup()
{
  Serial.begin(115200); //Initialize Serial Port
  //attempt to connect to wifi
  Serial.print("Attempting to connect to Network named: ");
  // print the network name (SSID);
  Serial.println(ssid);

  //Connect to WiFi
  WiFi.begin(ssid, password);

  //Wait untillwifi is connected
  while ( WiFi.status() != WL_CONNECTED)
  {
    // print dots while we wait to connect
    Serial.print(".");
    delay(300);
  }

  //If you are connected print the IP Address
  Serial.println("\nYou're connected to the network");

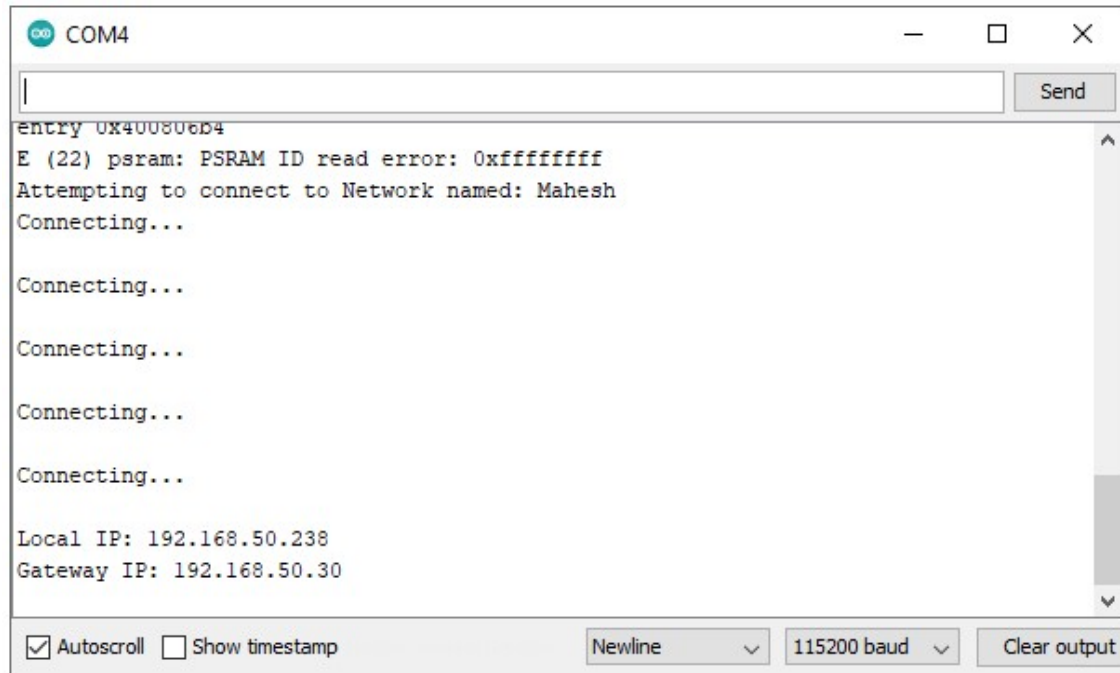
  //wait untill you get an IP address
  while (WiFi.localIP() == INADDR_NONE) {
    // print dots while we wait for an ipaddresss
    Serial.print(".");
    delay(300);
  }

  ip=WiFi.localIP();
  Serial.println(ip);
  gateway=WiFi.gatewayIP();
  Serial.println("GATEWAY IP:");
  Serial.println(gateway);
}

void loop()
{
  // put your main code here, to run repeatedly:

}
```

Result:



The screenshot shows a serial terminal window titled "COM4". At the top, there is a text input field and a "Send" button. The main area displays the following text: "entry 0x400806b4", "E (22) psram: PSRAM ID read error: 0xffffffff", "Attempting to connect to Network named: Mahesh", and five consecutive "Connecting..." lines. Below these, the network configuration is shown: "Local IP: 192.168.50.238" and "Gateway IP: 192.168.50.30". At the bottom, there are checkboxes for "Autoscroll" (checked) and "Show timestamp" (unchecked), followed by a "Newline" dropdown menu, a "115200 baud" dropdown menu, and a "Clear output" button.

```
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff
Attempting to connect to Network named: Mahesh
Connecting...

Connecting...

Connecting...

Connecting...

Connecting...

Local IP: 192.168.50.238
Gateway IP: 192.168.50.30
```

☒ Autoscroll ☐ Show timestamp Newline 115200 baud Clear output

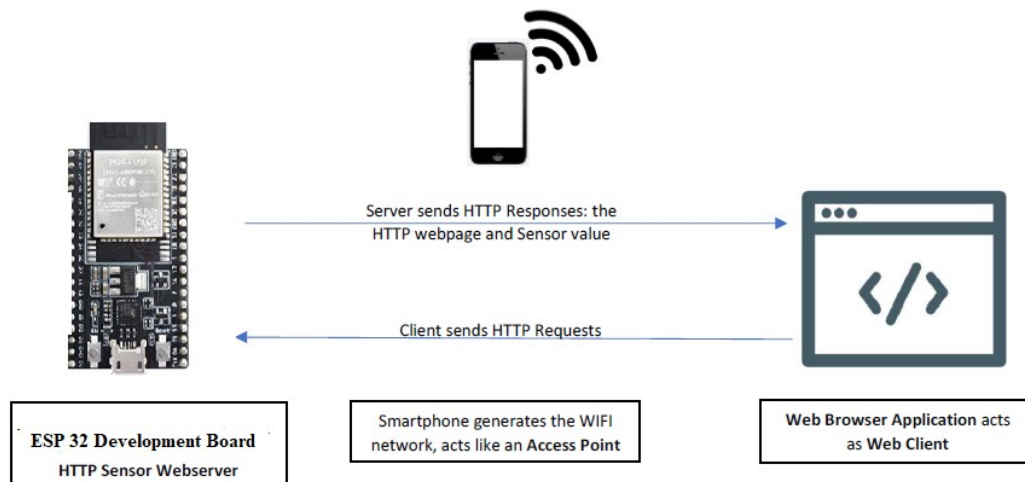
Experiment -3

3a) Design and Implementation of HTTP based IoT Web Server to control the status of LED

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	DHT11 or DHT22 temperature and humidity sensor	1
3	Jumper wires	3
4	Micro USB cable	1

An HTTP based webserver provides access to data to an HTTP client such as web browser. In IoT applications, the IoT Nodes are connected to sensors and actuators. The sensor data can be accessed using a web browser and also the actuators can be controlled from the web browser. This facilitates a wide variety of applications in IoT domain. The underlying protocol used for this communication between a Client and a Server is HTTP. The goal of this lab is to understand the configuration and working principle of an IoT web server. Using a browser, such as Google Chrome, we will send some HTTP based commands to the web server. Based on the type of command, the IoT Node web server will toggle the LEDs.



CODE:

```
#include <WiFi.h>

const char* ssid    = "Mahesh";
const char* password = "012345678";

WiFiServerserver(80);

void setup()
{
  Serial.begin(115200);
  pinMode(2, OUTPUT);    // set the LED pin mode

  delay(10);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  server.begin();
}

int value = 0;

void loop(){
  WiFiClient client = server.available(); // listen for incoming clients

  if (client) { // if you get a client,
    Serial.println("New Client."); // print a message out 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
        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();

            // the content of the HTTP response follows the header:
            client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 2 on.<br>");
            client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 2 off.<br>");

            // 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
        }

        // Check to see if the client request was "GET /H" or "GET /L":
        if (currentLine.endsWith("GET /H")) {
            digitalWrite(2, HIGH); // GET /H turns the LED on
        }
        if (currentLine.endsWith("GET /L")) {
            digitalWrite(2, LOW); // GET /L turns the LED off
        }
    }
}

// close the connection:
client.stop();
Serial.println("Client Disconnected.");
}
}

```

```

mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (26) psram: PSRAM ID read error: 0xffffffff

Connecting to Mahesh
.....
WiFi connected.
IP address:
192.168.28.238

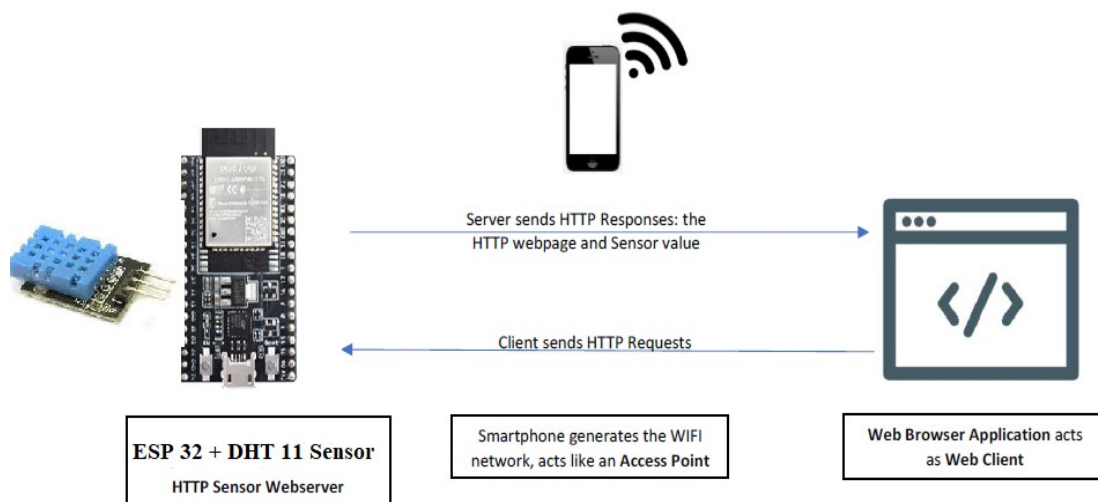
```


On Web browser/Web Client:



3b) Design and Implementation of HTTP based IoT Web Server to display sensor value

The goal of this experiment is to integrate the HTTP webserver with Sensor. As we know, most IoT Nodes are equipped with sensors and the IoT system should provide these sensor data to users. In this experiment, we used DHT11 Temperature and Humidity sensor connected to an IoT Node/ESP32 development board. The IoT Node is configured as a Webserver and the sensor data can be accessed via a web browser.



CODE:

```
#include <WiFi.h>
#include <WebServer.h>
#include "DHT.h"
```

// Uncomment one of the lines below for whatever DHT sensor type you're using!

```

#define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT21 // DHT 21 (AM2301)
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321

/*Put your SSID & Password*/
const char* ssid = "REPLACE_WITH_YOUR_SSID"; // Enter SSID here
const char* password = "REPLACE_WITH_YOUR_PASSWORD"; //Enter Password here

WebServer server(80);

// DHT Sensor
uint8_t DHTPin = 4;

// 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,floatHumiditystat){
    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>ESP32 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>ESP32 Weather Report</h1>\n";

    ptr += "<p>Temperature: ";
    ptr += (int)Temperaturestat;
    ptr += "\xC2\xB0 C</p>";
    ptr += "<p>Humidity: ";
    ptr += (int)Humiditystat;
    ptr += "%</p>";

    ptr += "</div>\n";
    ptr += "</body>\n";
    ptr += "</html>\n";
    return ptr;
}

```

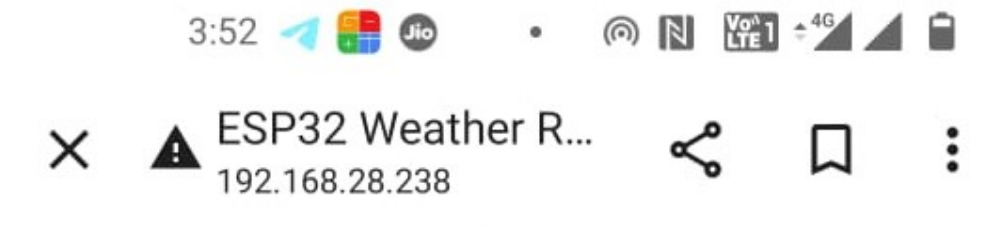
RESULT:

```
COM4
Send

clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (47) psram: PSRAM ID read error: 0xffffffff
Connecting to
Mahesh
...
WiFi connected...!
Got IP: 192.168.28.238
HTTP server started

☒ Autoscroll ☐ Show timestamp
Newline 115200 baud Clear output
```

On Web browser/Web Client:



ESP32 Weather Report

Temperature: 27° C

Humidity: 59%

Experiment -4

Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server.

Theory:

MQTT protocol

MQTT stands for Message Queuing Telemetry Transport. MQTT is a machine-to-machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications. Some of the features of an MQTT are given below:

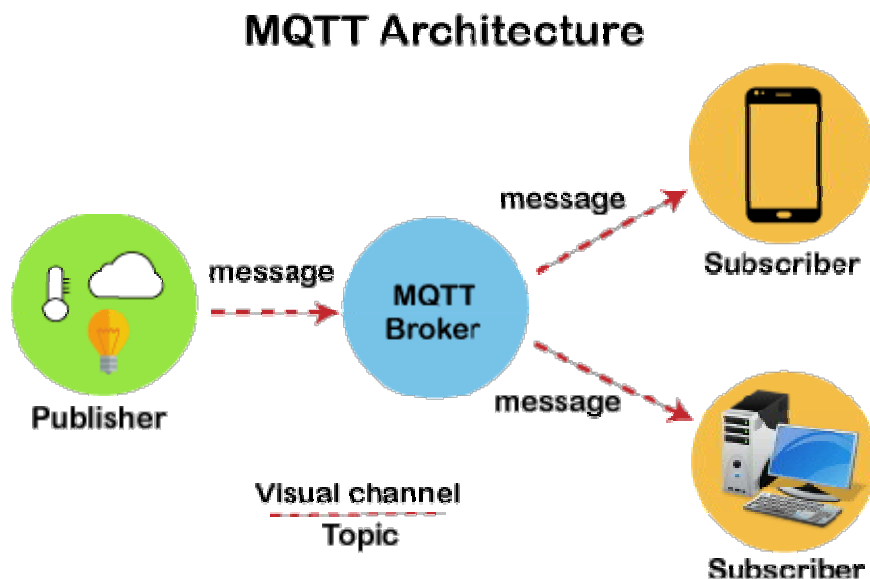
- It is a machine-to-machine protocol, i.e., it provides communication between the devices.
- It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe system to exchange the information between the client and the server.
- It does not require that both the client and the server establish a connection at the same time.
- It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
- It allows the clients to subscribe to the narrow selection of topics so that they can receive the information they are looking for.

MQTT Architecture:

To understand the MQTT architecture, we first look at the components of the MQTT.

- Message
- Client
- Server or Broker
- TOPIC

- **Message:** Message: The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters: Payload data, Quality of Service (QoS), Collection of Properties, Topic Name
- **Client:** In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In MQTT, the client performs two operations:
 1. **Publish:** When the client sends the data to the server, then we call this operation as a publish.
 2. **Subscribe:** When the client receives the data from the server, then we call this operation a subscription.

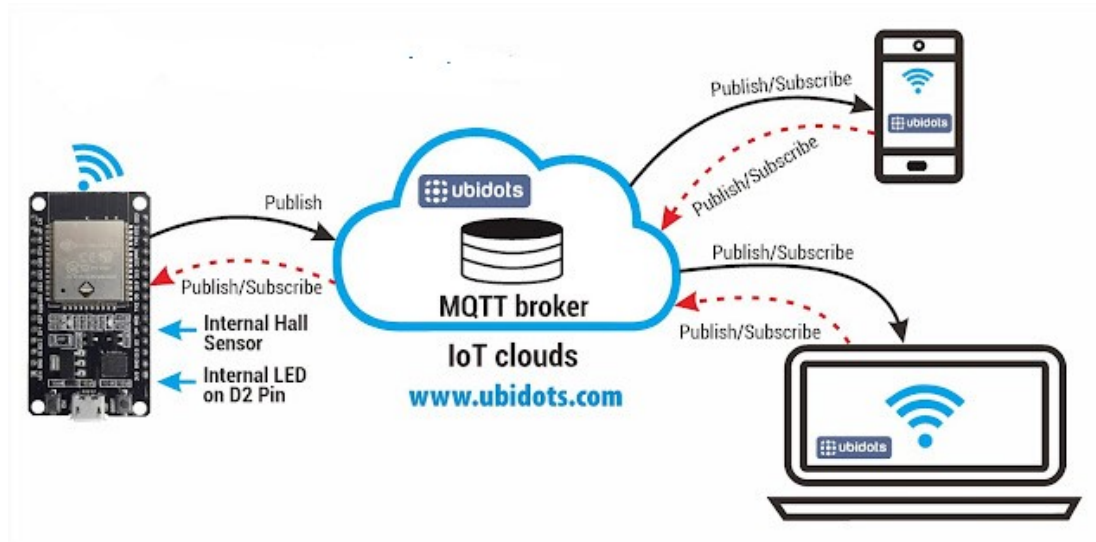


For more details: <https://www.javatpoint.com/mqtt-protocol>

Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server.

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	LED	1
3	Jumper wires	2
4	Micro USB cable	1



Create Account on Ubidots Website:

1. <https://www.robotics-university.com/2019/05/internet-of-things-monitoring-sensor-and-controlling-led-on-ubidots-dashboard-using-mqtt-protocol.html>
2. <https://ubidots.com/>
3. <https://help.ubidots.com/en/articles/854333-ubidots-basics-devices-variables-dashboards-and-alerts>

CODE:

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

#define WIFISSID "Replace_With_Your_Ssid" // Put your WifiSSID here
#define PASSWORD "Replace_With_Your_Password" // Put your wifi password here
#define TOKEN "YOUR_TOKEN" // Put your Ubidots' TOKEN
#define MQTT_CLIENT_NAME "ESP32" // MQTT client Name, please enter your own 8-12 alphanumeric character ASCII string;
//it should be a random and unique ascii string and different from all other devices

/*****
* Define Constants
*****/
#define VARIABLE_LABEL "Variable Name 1" // Assing the variable label
#define VARIABLE_LABEL_SUBSCRIBE " Variable Name 2" // Assing the variable label
#define DEVICE_LABEL "Device Name" // Assig the device label

#define led 26 // Set the GPIO26 as LED

char mqttBroker[] = "things.ubidots.com";
char payload[100];
```

```

char topic[150];
char topicSubscribe[100];
// Space to store values to send
char str_sensor[10];

/*****
* Auxiliar Functions
*****/
WiFiClient ubidots;
PubSubClient client(ubidots);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");

    // Attempt to connect
    if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {
      Serial.println("Connected");
      client.subscribe(topicSubscribe);
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;
  String message(p);
  if (message == "0.0") {
    digitalWrite(led, LOW);
  } else {
    digitalWrite(led, HIGH);
  }

  Serial.write(payload, length);
  Serial.println();
}

/*****
* Main Functions
*****/
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFISSID, PASSWORD);

```



```

    // Assign the pin as INPUT
    pinMode(led, OUTPUT);

    Serial.println();
    Serial.print("Wait for WiFi...");

    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    Serial.println("");
    Serial.println("WiFi Connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    client.setServer(mqttBroker, 1883);
    client.setCallback(callback);
    sprintf(topicSubscribe, "/v1.6/devices/%s/%s/lv",
        DEVICE_LABEL, VARIABLE_LABEL_SUBSCRIBE);

    client.subscribe(topicSubscribe);
}

void loop() {
    if (!client.connected()) {
        client.subscribe(topicSubscribe);
        reconnect();
    }

    sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
    sprintf(payload, "%s", ""); // Cleans the payload
    sprintf(payload, "{\"%s\\\": ", VARIABLE_LABEL); // Adds the variable label

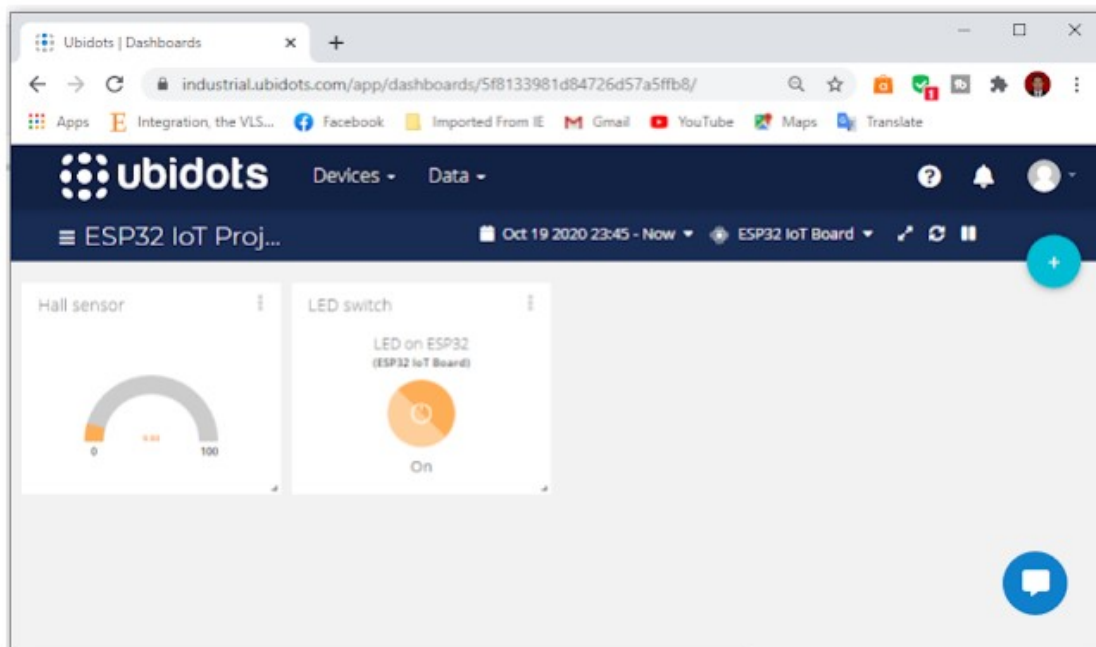
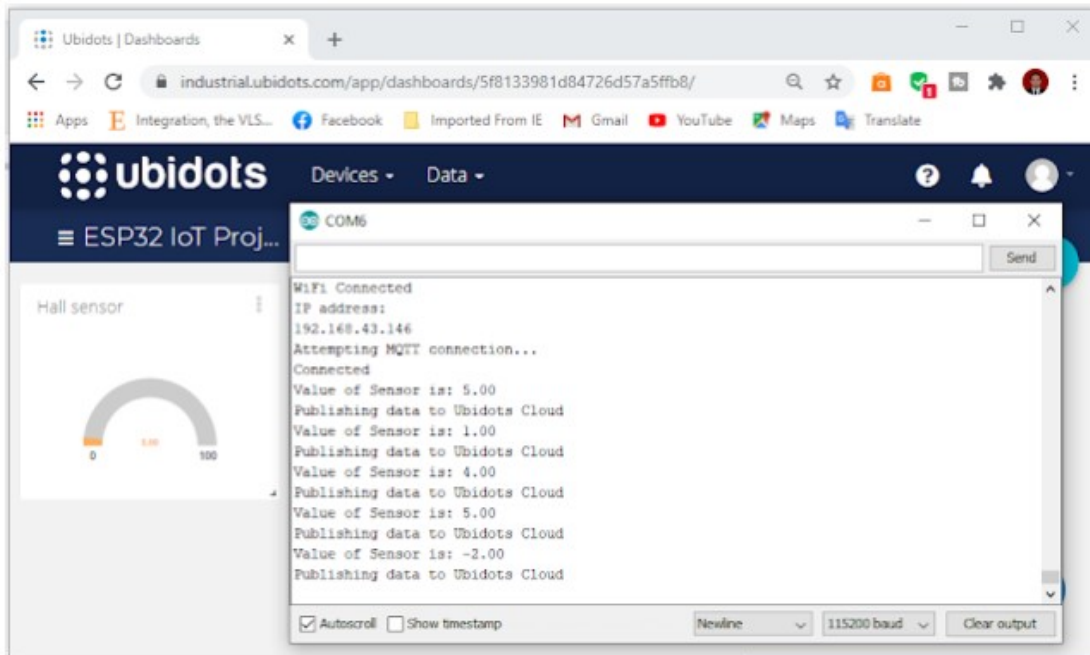
    float sensor = hallRead();
    Serial.print("Value of Sensor is:- "); Serial.println(sensor);

    /* 4 is minimum width, 2 is precision; float value is copied onto str_sensor */
    dtostrf(sensor, 4, 2, str_sensor);

    sprintf(payload, "%s {\\\"value\\\": %s} }", payload, str_sensor); // Adds the value
    Serial.println("Publishing data to Ubidots Cloud");
    client.publish(topic, payload);
    client.loop();
    delay(1000);
}

```

RESULT:



Realtime monitoring Hall sensor & control LED on Ubidots dashboard

Experiment -5

Establish a communication between client and IoT Web Server to POST and GET sensor values over HTTP, TCP or UDP

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32Development Board	1
2	Micro USB cable	1

PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Open the Arduino IDE in computer
3. Select the ESP32 board from Tools > Board > ESP32 Dev module.
4. Download the Ubidots library <https://github.com/ubidots/ubidots-esp32>
5. Now, click on Sketch -> Include Library -> Add .ZIP Library.
6. Select the .ZIP file of Ubidots and then "Accept".
7. Close the Arduino IDE and open it again and write the program. Save the new sketch in your working directory

Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**

8. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
9. After uploading the code, open the Serial Monitor at a baud rate of 115200.

POST values to Web Server /Ubidots:

CODE:

```
#include "Ubidots.h"
```

```
/******
```

```
* Define Instances and Constants
```

```
*****/
```

```
const char* UBIDOTS_TOKEN = "YOUR_TOKEN"; // Put here your Ubidots  
TOKEN
```

```
const char* WIFI_SSID = "Replace_With_Your_Ssid"; // Put here your Wi-Fi SSID
```

```
const char* WIFI_PASS = "Replace_With_Your_password" // Put here your Wi-Fi  
password
```

```
Ubidotsubidots(UBIDOTS_TOKEN, UBI_HTTP);
```

```

// Ubidotsubidots(UBIDOTS_TOKEN, UBI_TCP); // Uncomment to use TCP
// Ubidotsubidots(UBIDOTS_TOKEN, UBI_UDP); // Uncomment to use UDP

/*****
 * Auxiliar Functions
 *****/

// Put here your auxiliar functions

/*****
 * Main Functions
 *****/

void setup() {
  Serial.begin(115200);
  ubidots.wifiConnect(WIFI_SSID, WIFI_PASS);
  // ubidots.setDebug(true); // Uncomment this line for printing debug messages
}

void loop() {
  float value1 = random(0, 9) * 10;
  float value2 = random(0, 9) * 100;
  float value3 = random(0, 9) * 1000;
  ubidots.add("counter", value1); // Change for your variable name
  //ubidots.add("Variable_Name_Two", value2);
  // ubidots.add("Variable_Name_Three", value3);

  bool bufferSent = false;
  bufferSent = ubidots.send(); // Will send data to a device label that matches the
  device Id

  if (bufferSent) {
    // Do something if values were sent properly
    Serial.println("Values sent by the device");
  }

  delay(5000);
}

```

GET values from Web Server /Ubidots:

CODE:

```
#include "Ubidots.h"

/*****
 * Define Constants
 *****/

const char* UBIDOTS_TOKEN = "YOUR_TOKEN"; // Put here your Ubidots TOKEN
const char* WIFI_SSID = "_Your_SSID"; // Put here your Wi-Fi SSID
const char* WIFI_PASS = "Your_Password"; // Put here your Wi-Fi password
const char* DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM = "your device label";
// Replace with your device label
const char* VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM = "your variable";
// Replace with your variable label

Ubidotsubidots(UBIDOTS_TOKEN, UBI_HTTP);
// Ubidotsubidots(UBIDOTS_TOKEN, UBI_TCP); // Uncomment to use TCP

/*****
 * Auxiliar Functions
 *****/

// Put here your auxiliar functions

/*****
 * Main Functions
 *****/

void setup() {
  Serial.begin(115200);
  ubidots.wifiConnect(WIFI_SSID, WIFI_PASS);
  // ubidots.setDebug(true); // Uncomment this line for printing debug messages
}

void loop() {
  /* Obtain last value from a variable as float using HTTP */
  float value = ubidots.get(DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM,
  VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM);

  // Evaluates the results obtained
  if (value != ERROR_VALUE) {
    Serial.print("Value:");
    Serial.println(value);
  }
  delay(5000);
}
```