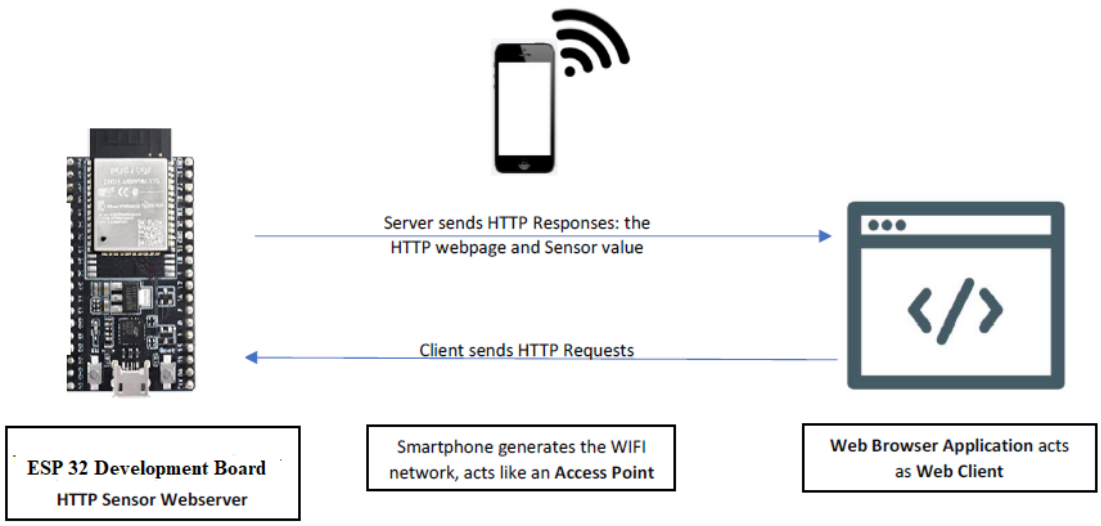
Experiment -3

3 a) 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

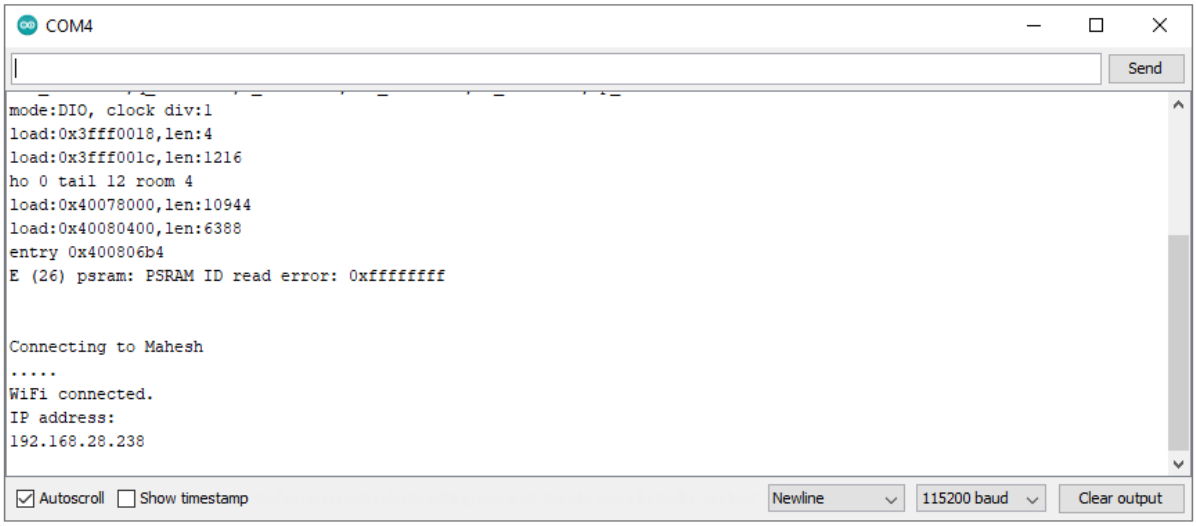
An HTTP based webserver provides access to data to an HTTP client such as web browser. In IoT applications, the IoT Nodes are connected to sensors and actuators. The sensor data can be accessed using a web browser and also the actuators can be controlled from the web browser. This facilitates a wide 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.



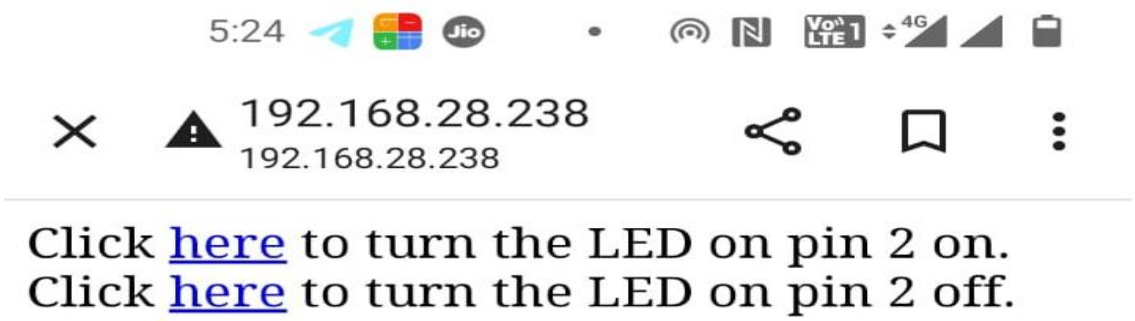
CODE:

```
#include <WiFi.h>

const char* ssid     = "Your_SSID";
const char* password = "Your_Password";

WiFiServer server(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.");
 }
}
```

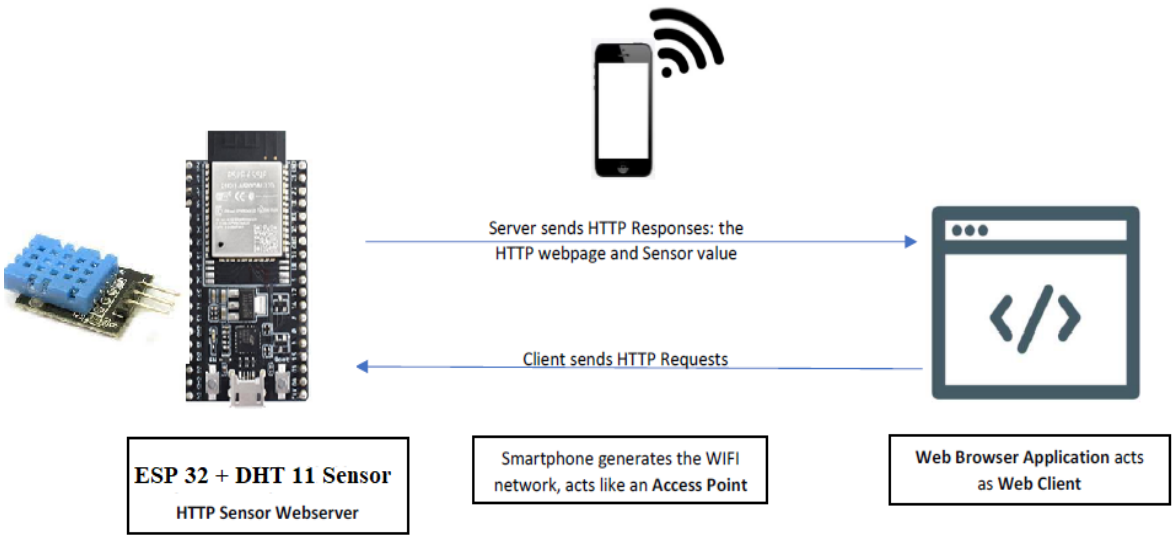


On Web browser/Web Client:



3 b) 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 systems 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,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>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;
}
```

On Web browser/Web Client:

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.

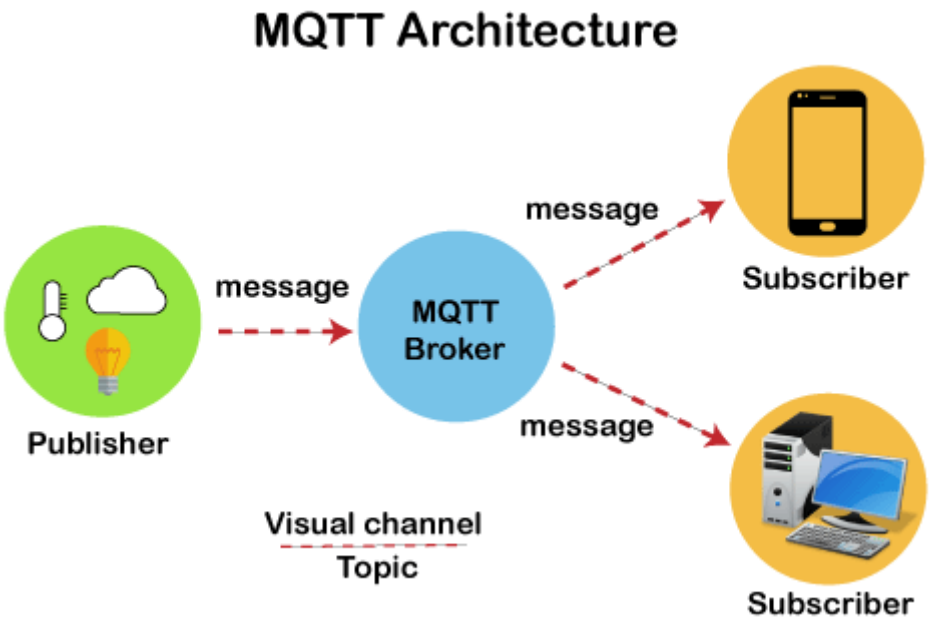
MQTT Architecture:

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.

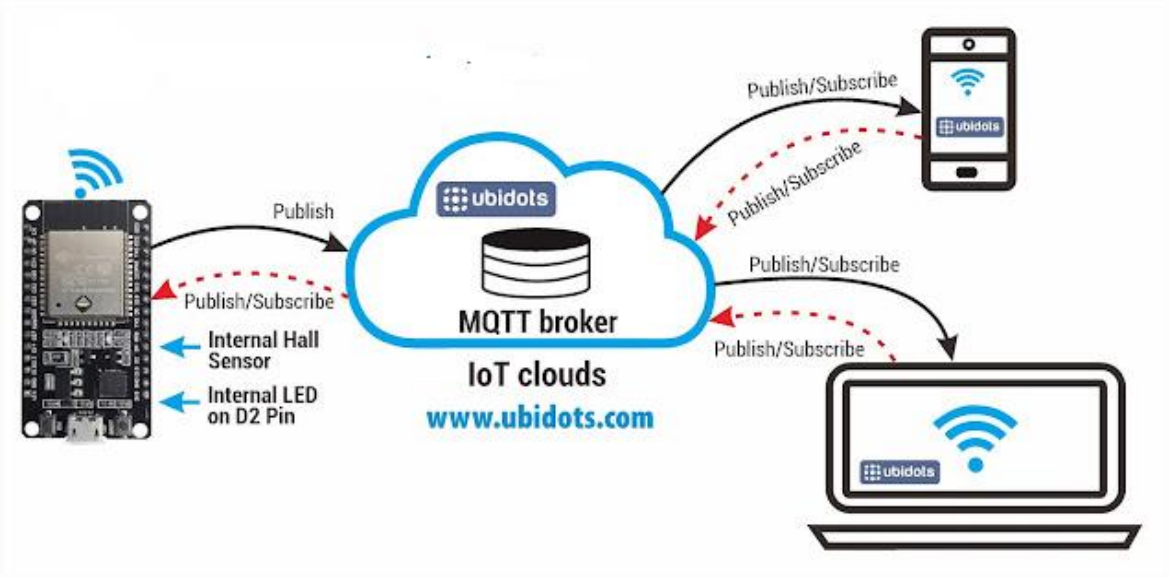


For more details: https://www.javatpoint.com/mqtt-protocol

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**

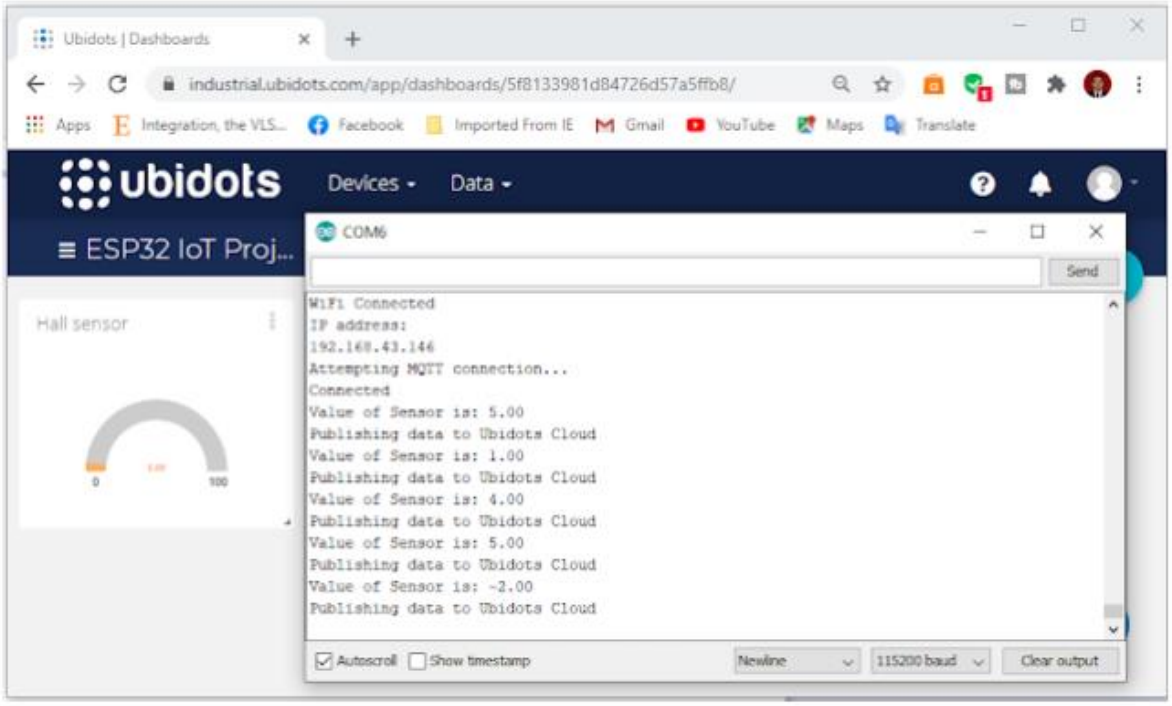
CODE:

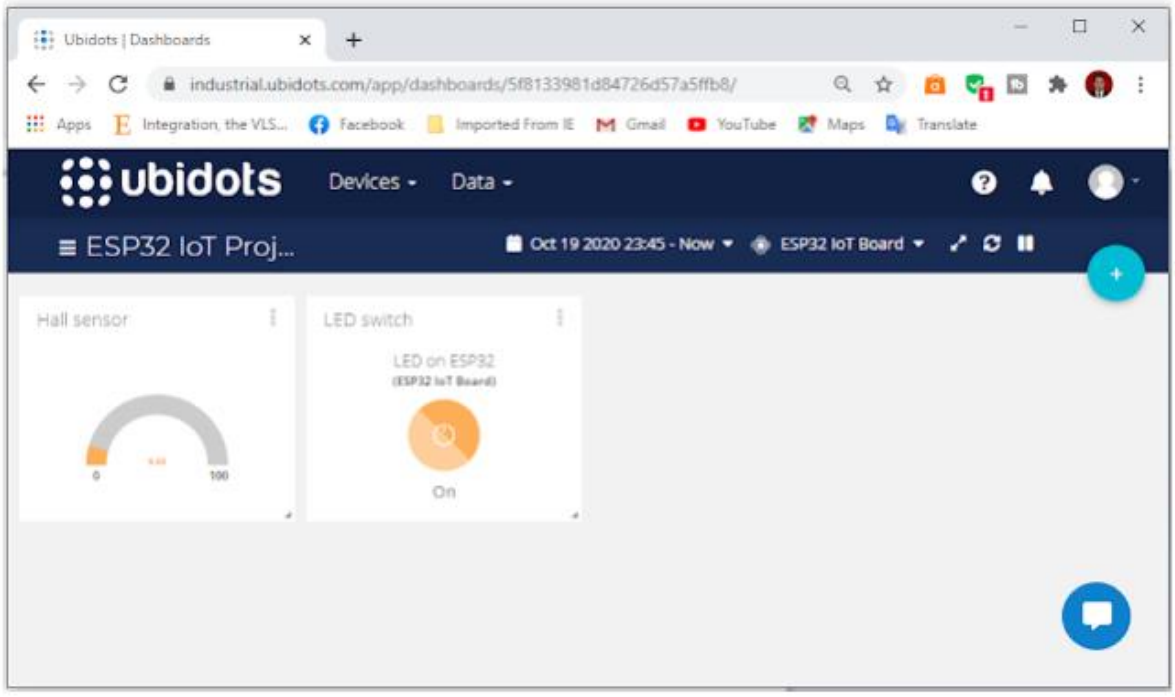
```
#include <WiFi.h>
#include <PubSubClient.h>

#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...");

    // Attemp 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 mininum 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	ESP32 Development 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
Ubidots ubidots(UBIDOTS_TOKEN, UBI_HTTP);
```

```
// Ubidots ubidots(UBIDOTS_TOKEN, UBI_TCP); // Uncomment to use TCP
// Ubidots ubidots(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

Ubidots ubidots(UBIDOTS_TOKEN, UBI_HTTP);
// Ubidots ubidots(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);
}
```