

**Department of Electrical Engineering**

**DEC50132 – internet based controller**

**PRACTICAL WORK NO : 3**

**TITLE : ESP32 using HTTP Protocol DATE :**

**LECTURER’S NAME : 1)**

**2)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PRACTICAL SKILL ASSESSMENT *(PLO5, LD2,CLO 3)*** | |  | **ATTAINMENT** | **LAB REPORT ASSESSMENT** | | **ATTAINMENT** | | |
| Able to make correct hardware connection | | S1 | ⑩ ⑧ ⑥ ④ ② ⓪ | **Result** | | ⑩ ⑧ ⑥ ④ ② ⓪ | | |
| S2 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S3 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S4 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| Able to establish connection to internet | | S1 | ⑩ ⑧ ⑥ ④ ② ⓪ | **Discussion** | | ⑩ ⑧ ⑥ ④ ② ⓪ | | |
| S2 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S3 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S4 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| Able to display data from sensor to web browser | | S1 | ⑩ ⑧ ⑥ ④ ② ⓪ | **Conclusion** | | ⑩ ⑧ ⑥ ④ ② ⓪ | | |
| S2 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S3 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S4 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| Able to get the suitable output for PBL | | S1 | ⑩ ⑧ ⑥ ④ ② ⓪ | **Score (30)** | |  | | |
| S2 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S3 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| S4 | ⑩ ⑧ ⑥ ④ ② ⓪ |
| **Score (40)** | | S1 |  | **Percentage (30%)** | |  | | |
| S2 |  |
| S3 |  | **Total CA Marks (100%)** | | S1 |  | |
| S4 |  | S2 |  | |
| **Percentage (70%)** | | S1 |  | S3 |  | |
| S2 |  |
| S3 |  | S4 |  | |
| S4 |  |
| **BIL.** | | **GROUP MEMBERS** | | | | **REGISTRATION NO.** | | |
| 1 | | CHONG KHENG CHEN | | | | 03DET22F1043 | | |
| 2 | |  | | | |  | | |
| 3 | |  | | | |  | | |
| 4 | |  | | | |  | | |

# LABORATORY

**PRACTICAL SKILL ASSESSMENT (PLO5, CLO2)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Score** | **Description** | | | | |
| **LAB RE**  10 | **PORT ASSESSMENT** **(PLO5, CLO2)**  Student can **complete all** tasks assigned **WITHOUT** errors | | | | |
| 8 | Student can **complete all** tasks assigned with **A FEW**  errors | | | | |
| 6 | Student can **complete all** tasks assigned with **MORE**  errors | | | | |
| 4 | Student can **complete partial** tasks assigned **WITHOUT** errors | | | | |
| 2 | Student can **complete partial** tasks assigned with **A FEW** errors | | | | |
| 0 | Student shows no response/task not attempted | | | | |
| **Report Component** | | | **Excellent** | **Very Good** | **Good** | **Fair** | | **Unsatisfactory** |
|  | | | **10** | **8** | **6** | **4** | | **2** |
| **Result**  Ability to record result correctly | | | Record the result  correctly Attached approved result that has been verify in the report. | Record result with less error. Attached approved result that has been verify in the report. | Record result with few errors. Attached result in the report. | Record result with error. Attached result in the report. | | result recorded partially Result not verify by lecturer. |
| **Discussion** Ability to present, interpret and analyze result. | | | All point of discussion on the results obtained covered and elaborated. | Most points of discussion on  results obtained covered and elaborated. | Some points of discussion on results obtained covered and elaborated. | Some points of discussion on results obtained covered and but not properly elaborated. | | Very few points of discussion, not properly elaborated. |
| **Conclusion** Provide answers to objectives stated earlier.    Ability to learn something from the experiment. | | | Conclusion includes whether the findings supported the hypothesis/objec tives, possible  sources of error, and what was learned from the experiment. | The closing paragraph summarizes and draws a sufficient conclusion. | The closing paragraph attempts to summarize but draws a weak conclusion. | The closing paragraph do not attempts to summarize the experiment OR shows little effort and reflection. | | No conclusion was included in the report. |

JKE POLIMAS

## Practical Work 3

|  |
| --- |
| **Title: ESP32 using HTTP Protocol** |
| **Objectives:**    Upon completion of this practical work, students should be able to:  ● Write codes to establish a connection to the internet and display data from sensor to web browser successfully.    **Equipments:**     1. Development Kit ESP32 2. Jumper wire 3. PC installed with Arduino IDE 4. Internet connection (Access Point dedicated to the Laboratory)       **Theory:**    **What is a Web server and how it works?**  Web server is a place which stores, processes and delivers web pages to Web clients. Web client is nothing but a web browser on our laptops and smartphones. The communication between client and server takes place using a special protocol called Hypertext Transfer Protocol (HTTP).    In this protocol, a client initiates communication by making a request for a specific web page using HTTP and the server responds with the content of that web page or an error message if unable to do so (like famous 404 Error). Pages delivered by a server are mostly HTML documents.  **ESP32 Operating Modes**  One of the greatest features ESP32 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices to connect directly to it and access web pages. This is possible because ESP32 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. This provides possibility of building [mesh networks.](https://en.wikipedia.org/wiki/Mesh_networking)  **Station (STA) Mode** |

The ESP32 that connects to an existing WiFi network (one created by your wireless router) i

s

called

Station

(

STA

)

In STA mode ESP32 gets IP from wireless router to which it is connected. With this IP address, it

can set up a web server and

deliver web pages to all connected devices under existing

WiFi

network

.

**Soft Access Point (AP) Mode**

The E

SP32 that creates its own WiFi network and acts as a hub (Just like WiFi router) for one or

more stations is called

Access Point

(

AP). Unlike WiFi router, it does not have interface to a wired

network. So, such mode of operation is called

Soft Access Point

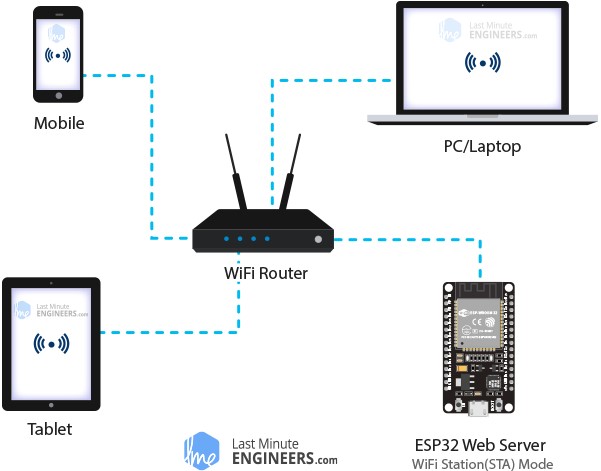
(

soft

-

AP). Also the maximum

number of stations that can connect to it is limited to five.



In AP mode ESP32 creates a new WiFi network and sets SSID (Name of the network) and IP

address to it. With this IP address, it can

deliver web pages to all connect

ed devices under its

own

network

.

Further reading available at:

[https://lastminuteengineers.com/creatin](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[g](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[-](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[2](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[esp3](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[-](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[we](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[b](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[-](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[serve](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[r](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[-](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

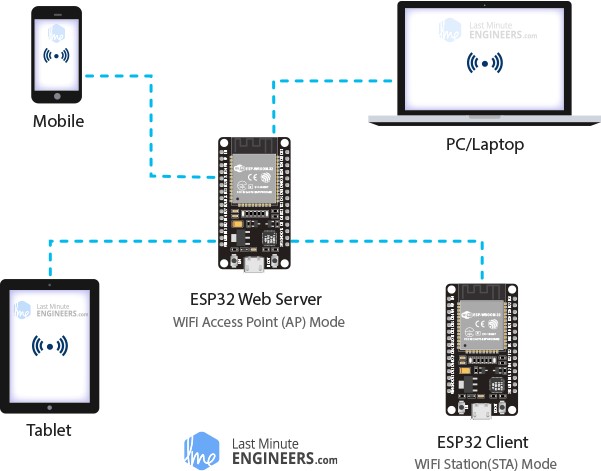
[arduin](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[o](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[-](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[ide](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)

[/](https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/)



**Procedures:**

A.

Controlling LED using web server

**1.**

**The circuit**

Connect two LEDs to the ESP32 as shown in the following schematic diagram

–

one LED connected to

GPIO

26

,

and the other to

GPIO 27.

**2.**

**The source code**

//

Load Wi

-

Fi library

#include <WiFi.h>

// Replace

with your network credentials

const char\* ssid

=

"

politeknik

";

const char\* password = "

politeknik2020

";

// Set web server port number to 80

WiFiServer server(80);

// Variable to store the HTTP request

String header;

// Auxiliar variables to store th

e current output state

String output26State = "off";

String output27State = "off";

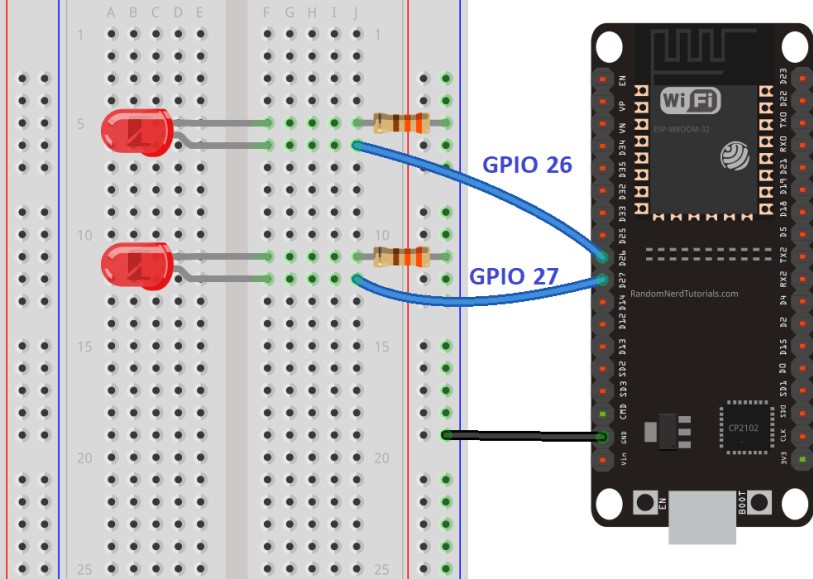
// Assign output variables to GPIO pins

const int output26 = 26;

const int output27 = 27;

void setup() {

Serial.begin(115200);



|  |
| --- |
| // Initialize the output variables as outputs  pinMode(output26, OUTPUT); pinMode(output27, OUTPUT);  // Set outputs to LOW digitalWrite(output26, LOW); digitalWrite(output27, LOW);    // 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,  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();    // turns the GPIOs on and off  if (header.indexOf("GET /26/on") >= 0) {  Serial.println("GPIO 26 on"); output26State = "on";  digitalWrite(output26, HIGH);  } else if (header.indexOf("GET /26/off") >= 0) {  Serial.println("GPIO 26 off"); output26State = "off";  digitalWrite(output26, LOW);  } else if (header.indexOf("GET /27/on") >= 0) {  Serial.println("GPIO 27 on"); output27State = "on";  digitalWrite(output27, HIGH);  } else if (header.indexOf("GET /27/off") >= 0) {  Serial.println("GPIO 27 off"); output27State = "off"; |

|  |
| --- |
| digitalWrite(output27, LOW);  }    // 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 on/off buttons  // Feel free to change the background-color and font-size attributes to fit your preferences client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}"); client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;"); client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}"); client.println(".button2 {background-color: #555555;}</style></head>");    // Web Page Heading  client.println("<body><h1>ESP32 Web Server</h1>");    // Display current state, and ON/OFF buttons for GPIO 26 client.println("<p>GPIO 26 - State " + output26State + "</p>"); // If the output26State is off, it displays the ON button if (output26State=="off") {  client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");  } else {  client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");  }  // Display current state, and ON/OFF buttons for GPIO 27 client.println("<p>GPIO 27 - State " + output27State + "</p>"); // If the output27State is off, it displays the ON button if (output27State=="off") {  client.println("<p><a href=\"/27/on\"><button class=\"button\">ON</button></a></p>");  } else {  client.println("<p><a href=\"/27/off\"><button class=\"button button2\">OFF</button></a></p>");  }  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("");  } } |

**3.**

**Uploading the Code**

Now, you can upload the code and and the web server will work straight away. Follow the next steps to

upload code to the ESP32:

a.

Plug your ESP32 board in your computer;

b.

In the Arduino IDE select your board

in

**Tools**

>

**Board**

(

in our case we’re using the ESP32

DEVKIT DOIT board);

c.

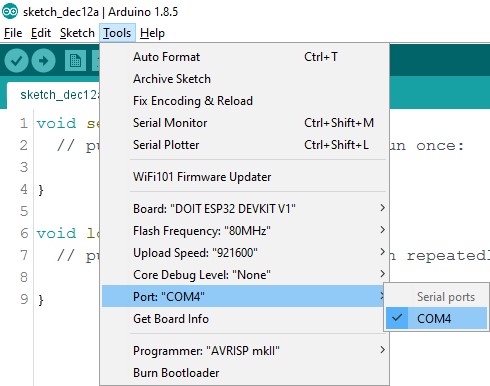
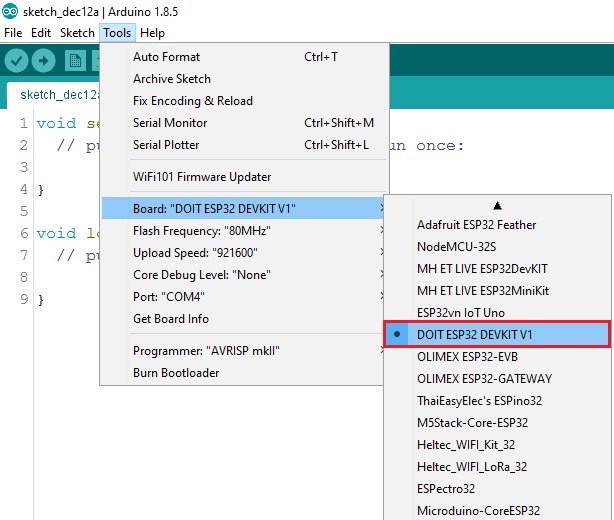
Select the COM port in

**Tools**

>

**Port**

.



d.

Press the

**Upload**

button in the Arduino IDE and wait a few seconds while the code compiles and

uploads to your board.

e.

Wait for the “

**Done**

**uploading**

” message.

**4.**

**Finding the ESP32 IP Address**

After uploading the code, open the Serial Monitor at a baud rate of 115200.

Press the ESP32 EN button (reset).

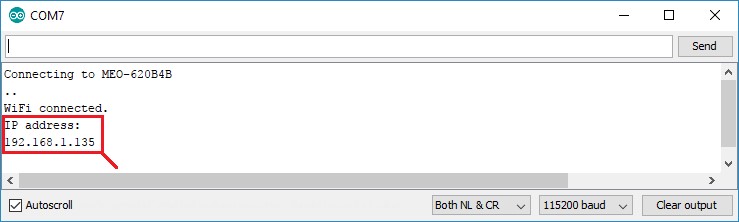
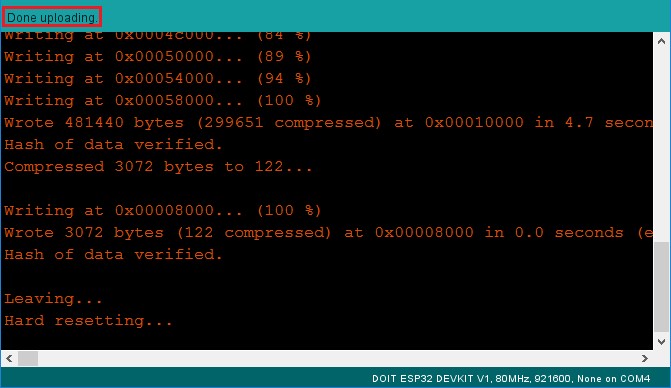
The ESP32 connects to Wi

-

Fi, and outputs the ESP IP address on the

Serial Monitor. Copy that

IP address, because you need it to access the ESP32 web server.



**5.**

**Accessing the Web Server**

To access the web server, open your browser, paste the ESP32 IP address, and you’ll see the following

page. In our case it is

**192.168.1.135**

.

If you take a look a

t the Serial Monitor, you can see what’s happening on the background. The ESP receives

an HTTP request from a new client (in this case, your browser).

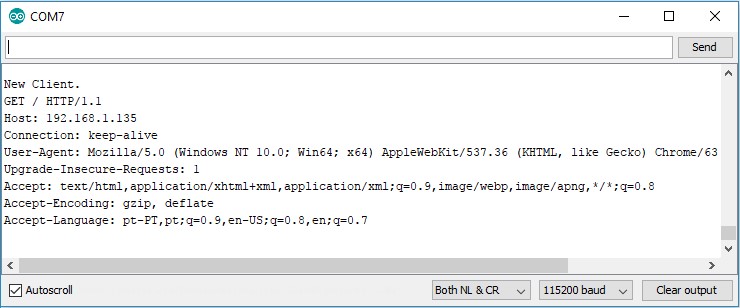
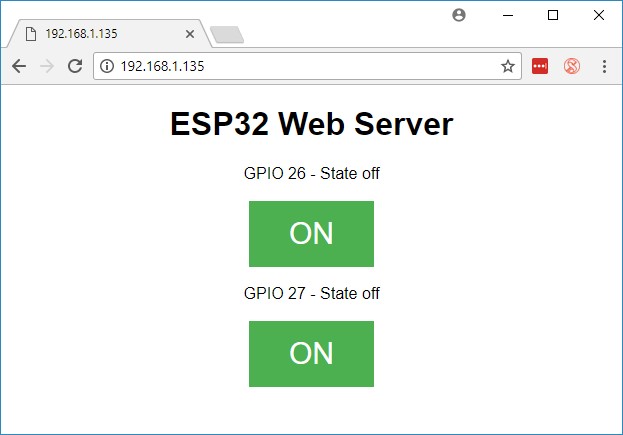
You can also see other information about the HTTP request.

**6.**

**Testing the Web Server**

Now you can test if

your web server is working properly. Click the buttons to control the LEDs.



At the same time, you can take a look at the Serial Monitor to see what’s going on in the background. For

example, when you click the button to turn

GPIO 26

ON, ESP32 receives a

request on the

**/26/on**

URL.

When the ESP32 receives that request, it turns the LED attached to

GPIO 2

6

ON and updates its state on the

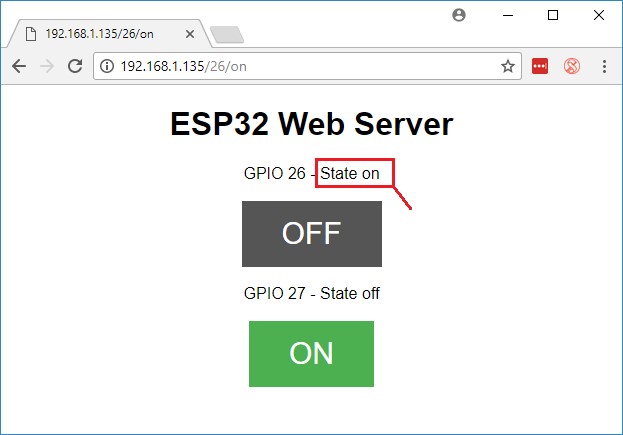
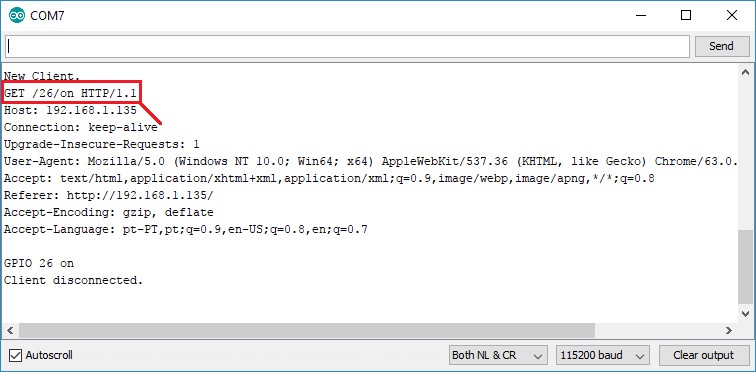
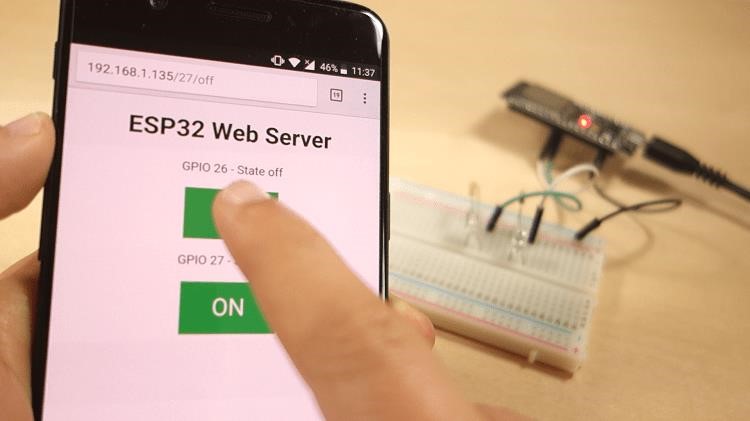
web page.

The button for

GPIO 2

7

works in a similar way. Test that it is working properly.



**B.**

**DHT11 web server**

**1.**

**The**

**circuit**

Before proceeding to the web server, you need to wire the DHT11 or DHT22 sensor to the ESP32 as shown

in the following schematic diagram.

In this case, we’re connecting the data pin to

GPIO 22

, but you can connect it to any other digital pin. You

can

use this schematic diagram for both DHT11 and DHT22 sensors.

You need to install a couple of libraries for this project:

**2.**

**Installing the libraries**

You need to install a couple of libraries for this project:

**Libraries name**

**Address**

[**DH**](https://github.com/adafruit/DHT-sensor-library)

[**T**](https://github.com/adafruit/DHT-sensor-library)

https://github.com/adafruit/DHT

-

sensor

-

library

[**Adafruit Unified Sensor Drive**](https://github.com/adafruit/Adafruit_Sensor)

[**r**](https://github.com/adafruit/Adafruit_Sensor)

https://github.com/adafruit/Adafruit\_Sensor

[**r**](https://github.com/me-no-dev/ESPAsyncWebServer)

[**ESPAsyncWebServe**](https://github.com/me-no-dev/ESPAsyncWebServer)

https://github.com/me

-

no

-

dev/ESPAsyncWebServer

[**P**](https://github.com/me-no-dev/AsyncTCP)

[**Async TC**](https://github.com/me-no-dev/AsyncTCP)

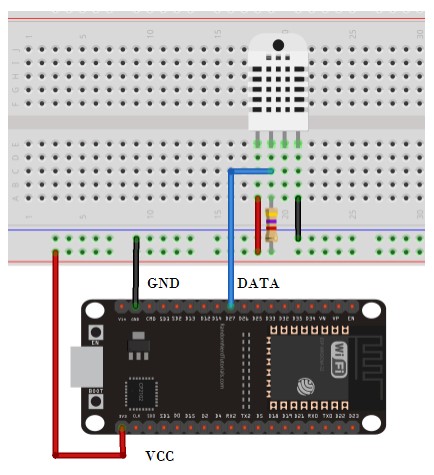
https://github.com/me

-

no

-

dev/AsyncTCP



|  |  |
| --- | --- |
| Follow the next instructions to install those libraries:   1. Copy the link, and start downloading the libraries. You should have a .zip folder in your *Downloads* folder 2. Unzip the .zip folder and you should get the libraries folder 3. Rename your folderDHT-sensor-library-master to *DHT\_sensor*, Adafruit\_sensor, ESPAsyncWebServer and AsyncTCP. 4. Move the *DHT\_sensor*, Adafruit\_sensor, ESPAsyncWebServer and AsyncTCP folders to your Arduino IDE installation libraries folder. 5. Finally, re-open your Arduino IDE   **3. The source code**  Open your Arduino IDE and copy the following code.   |  | | --- | | // Import required libraries  #include "WiFi.h"  #include "ESPAsyncWebServer.h"  #include <Adafruit\_Sensor.h>  #include <DHT.h>  // Replace with your network credentials const char\* ssid = "IntanShafi"; const char\* password = "x12x12x12";  #define DHTPIN 22 // Digital pin connected to the DHT sensor  // Uncomment the type of sensor in use:  #define DHTTYPE DHT11 // DHT 11  //#define DHTTYPE DHT22 // DHT 22 (AM2302)  //#define DHTTYPE DHT21 // DHT 21 (AM2301)  DHT dht(DHTPIN, DHTTYPE);  // Create AsyncWebServer object on port 80  AsyncWebServer server(80);  String readDHTTemperature() {  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)  // Read temperature as Celsius (the default) float t = dht.readTemperature();  // Read temperature as Fahrenheit (isFahrenheit = true)  //float t = dht.readTemperature(true);  // Check if any reads failed and exit early (to try again). if (isnan(t)) {  Serial.println("Failed to read from DHT sensor!"); return "--";  }  else {  Serial.println(t); return String(t);  } }  String readDHTHumidity() {  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor) float h = dht.readHumidity(); if (isnan(h)) {  Serial.println("Failed to read from DHT sensor!"); return "--";  } | |
| |  | | --- | | else {  Serial.println(h);  return String(h);  }  }  const char index\_html[] PROGMEM = R"rawliteral(  <!DOCTYPE HTML><html>  <head>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">  <style> html {  font-family: Arial; display: inline-block; margin: 0px auto;  text-align: center;  }  h2 { font-size: 3.0rem; } p { font-size: 3.0rem; } .units { font-size: 1.2rem; }  .dht-labels{ font-size: 1.5rem; vertical-align:middle; padding-bottom: 15px;  }  </style>  </head>  <body>  <h2>ESP32 DHT Server</h2>  <p>  <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>  <span class="dht-labels">Temperature</span>  <span id="temperature">%TEMPERATURE%</span>  <sup class="units">&deg;C</sup>  </p>  <p>  <i class="fas fa-tint" style="color:#00add6;"></i>  <span class="dht-labels">Humidity</span>  <span id="humidity">%HUMIDITY%</span>  <sup class="units">%</sup>  </p>  </body> <script>  setInterval(function ( ) {  var xhttp = new XMLHttpRequest(); xhttp.onreadystatechange = function() { if (this.readyState == 4 && this.status == 200) {  document.getElementById("temperature").innerHTML = this.responseText;  }  };  xhttp.open("GET", "/temperature", true); xhttp.send(); }, 10000 ) ;  setInterval(function ( ) {  var xhttp = new XMLHttpRequest(); xhttp.onreadystatechange = function() { if (this.readyState == 4 && this.status == 200) {  document.getElementById("humidity").innerHTML = this.responseText;  }  };  xhttp.open("GET", "/humidity", true); xhttp.send(); | | | |

|  |  |
| --- | --- |
| |  | | --- | | }, 10000 ) ;  </script>  </html>)rawliteral";    // Replaces placeholder with DHT values  String processor(const String& var){  //Serial.println(var);  if(var == "TEMPERATURE"){  return readDHTTemperature();  }  else if(var == "HUMIDITY"){  return readDHTHumidity();  }  return String();  }    void setup(){  // Serial port for debugging purposes  Serial.begin(115200);    dht.begin();    // Connect to Wi-Fi  WiFi.begin(ssid, password);  while (WiFi.status() != WL\_CONNECTED) { delay(1000);  Serial.println("Connecting to WiFi..");  }    // Print ESP32 Local IP Address  Serial.println(WiFi.localIP());    // Route for root / web page  server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){ request->send\_P(200, "text/html", index\_html, processor);  });  server.on("/temperature", HTTP\_GET, [](AsyncWebServerRequest \*request){ request->send\_P(200, "text/plain", readDHTTemperature().c\_str());  });  server.on("/humidity", HTTP\_GET, [](AsyncWebServerRequest \*request){ request->send\_P(200, "text/plain", readDHTHumidity().c\_str());  });    // Start server server.begin();  }    void loop(){  } |     **4. Uploading the code**  Now, upload the code to your ESP32. Make sure you have the right board and COM port selected.  After uploading, open the Serial Monitor at a baud rate of 115200. Press the ESP32 reset button. The ESP32 IP addre ss should be printed in the serial monitor. |

**5.**

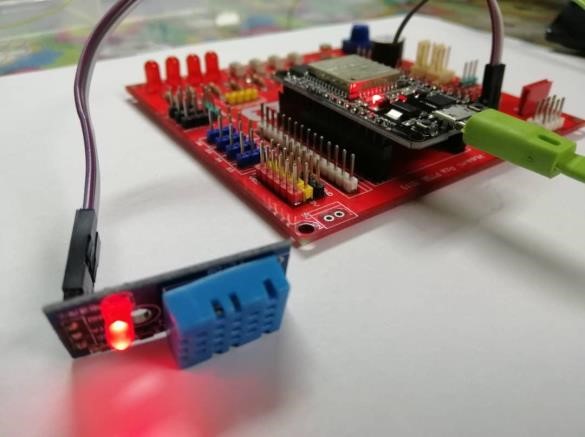
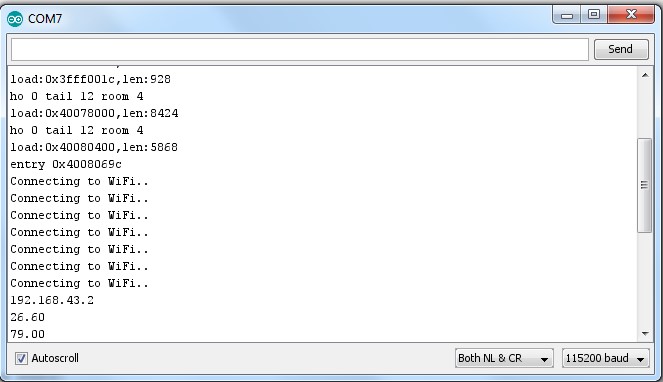
**Web server demonstration**

Open a browser and type the ESP32 IP address. Your web server shoul

d display the latest sensor readings.

Notice that the temperature and humidity readings are updated automatically without the need to refresh the

web page.



**C.**

**Problem Based Learning**

Write the code based on PART B (DHT11 Web Server)

If temperature

, led is on

>28

If temperature <28, led is off

**Results:**

**Discussion:**

Write your discussion based on written coding for the

**Part A, Part B**

and

**Part C**

. Explain how the

codes work.

**Conclusion:**

* on this practical work Students should be able to successfully deploy a practical IoT solution that involves connecting to the internet, reading sensor data, and presenting it on a web page. Gaining insights into the importance of real-time data visualization and control in IoT applications. Understanding the relationship between sensor data, web servers, and user interfaces in the context of IoT.

Result:

PART A

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated

A person holding a circuit board

Description automatically generated

A screenshot of a phone

Description automatically generated

A person holding a circuit board

Description automatically generated

A screenshot of a phone

Description automatically generated

A person holding a circuit board

Description automatically generated

PART B

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A hand holding a circuit board

Description automatically generated

A screenshot of a device

Description automatically generated

PART C

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen with numbers and symbols

Description automatically generated

A black background with white text

Description automatically generated

A computer screen with a computer and a circuit board

Description automatically generated

A computer screen with a computer and a computer with wires

Description automatically generated with medium confidence