ESP32 WEB SERVER

What is our GOAL for this CLASS?

In this class, we created a **web server with an ESP32 WIFI Mode that controls outputs (one LED and one buzzer) on the local environment**.

What did we ACHIEVE in the class TODAY?

- We were introduced to **Wifi** Mode.
- We learned about the Creation of a **Web Server**.
- We learned how **to access** Web Server.
- We learned how to access **LED & Buzzer**.

Which CONCEPTS/ CODING BLOCKS did we cover today?

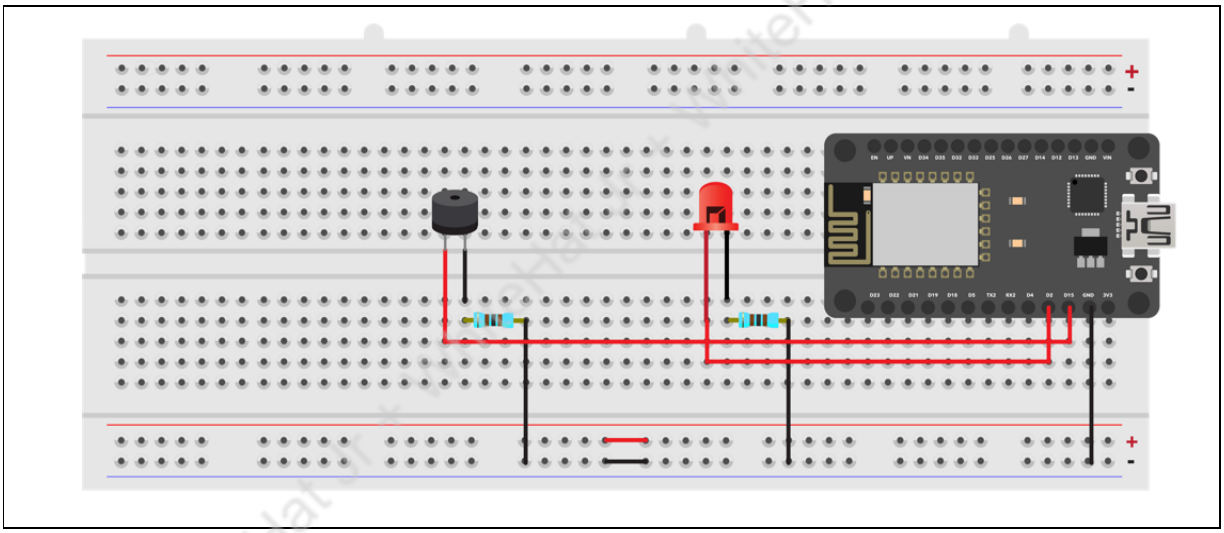
- We learned about automation in which we learned to **control LED & Buzzer** on the server just by toggling a button.
- We made a **local server**.

How did we DO the activities?

1. Gather the material from the IoT kit:
   - 1 x ESP32
   - 1 x USB Cable
   - 1 x Breadboard
   - 6 x Jumper wires

- 1 x Buzzer
- 1 x LED

2. Connections for **Circuit** Diagram
   - Supply **negative(GND (-ve))** from the ESP 32 to the breadboard negative terminal.
   - Insert buzzer and LED into the breadboard
   - Connect the resistor's one terminal with a longer leg of the LED and the other end of the resistor with **ESP32 GPIO pin** numbers 15
   - Connect the second resistor's one terminal with a longer leg of the buzzer. And the other end of the resistor with ESP32 GPIO pin numbers 15
   - Connect the short leg of LED and Buzzer with **GND(0V)** supply.



3. Write a code:
   - Include **web server libraries**.
   - Using the **WiFi library**, the device will be able to answer an **HTTP request** with your **WiFI credentials**.
   - After opening a web browser and navigating to your **WiFi IP** address, the board will respond with HTML content along it will display the input values from the ESP32 board.
   - **include** keyword is used to import libraries in embedded language as we used to import in python language
     - Load **WiFi library**: WiFi library will be able to answer all HTTP request
     - Load **WiFiClient:** WiFiClient client helps to connect to a specified internet IP address and port.
     - **WebServer library** will help to create a web server on ESP32
     - **ESPmDNS** enabled DNS(Domain Name System)

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
```

4. Connect with **ESP32** with the WiFi. For that, we need to use **SSID**(Wi-Fi credentials i.e WiFi name and WiFi Password)
   - **Constant char** is a variable that is used to save WiFi credentials. Set the SSID and password
   - Load the HTML design string, this string will take the actual design of the HTML page so use all content of HTML in one string.
   - Set **webServer port** number to 80
   - **void handleroot()** function monitors the presence of a webpage request and delivers the requested webpage.
   - In response to an accepted request, server. send will send a **success message**
   - **200 means** the request is ok, usually, this will be the standard practice for sending messages for successful web pages.

```
const char* ssid = "WR3005N3-757E";
const char* password = "7002949";

String button = "<html><body id='bdy_1' style='height: 100px; width: 100px;'>

WebServer server(80); //http port number

void handleRoot(){
// (192.168.1.1/){
  server.send(200, "text/html", button); //here 200 is Success code
}
```

5. In case the HTTP **request fails**, the **handleNotFound()** function comes into play.
   - The output pin

```
void loop() {

  int analogValue = analogRead(4);

  float voltage = floatMap(analogValue, 0, 4095, 0, 3.3);

  Serial.print("Analog: ");
  Serial.print(analogValue);
  Serial.print(", Voltage: ");
  Serial.println(voltage*1000);

  delay(1000);
}
```

6. Compile and upload the program to **ESP32 board** using Arduino IDE
   - Verify the program on clicking **Tick option**
   - Upload the program on **clicking arrow option**
     - If the port is not selected, insert the USB cable in Computer's port and select the port
   - Go to Tools and select Serial Monitor
     - Rotate the **potentiometer** and see the output

   - By rotating the potentiometer knob, **observe the voltage value. The voltage will rise or fall**.

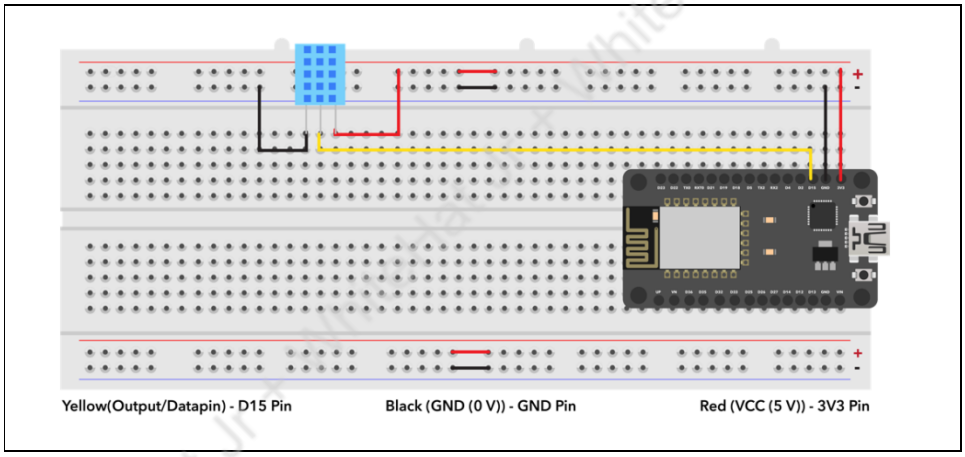
7. To read the temperature and humidity value from the **DHT11 sensor** and print it to Serial Monitor and check the same on the graph

   - Gather the material from the IoT kit:
     - 1 x ESP32
     - 1 x USB Cable

- 1 x Breadboard
- 4 x Jumper wires
- 1 x Potentiometer
- 1 x Rotary Potentiometer

8. Connections for **Circuit** Diagram

- Supply positive**(VCC (+ve))** from the ESP 32 to breadboard terminal
- Supply negative**(GND (-ve))** from the ESP 32 to breadboard negative terminal
- Take the **DHT11 sensor**, female jumper wires are already connected with DHT11
- Take three male jumper wires insert into **DHT11 sensor**
- connect **VCC (+ve)** of DHT11 with **VCC (+ve)** of the breadboard
- connect **GND(-ve)** of DHT11 with **GND(-ve)** of the breadboard
- Connect data/output pin of **DHT11** with **D15** of the **ESP32**



9. Write the code:
   - Define Pins
     - define **DHTPIN 15**
     - define **DHTPIN DHT11**

```
#define DHTPIN 15

#define DHTTYPE DHT11


DHT dht(DHTPIN, DHTTYPE);
```

   - Initialize the **setup()**
     - **Serial. begin(9600)** is used for data exchange speed.. This tells the **Arduino** to get ready to exchange messages with the Serial Monitor at a data rate of

9600 bits per second. That's 9600 binary ones or zeros per second, and is commonly called a baud rate.
- **Serial.println** used to print data.
- **dht.begin() is** used to begin the process

```
void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 sensor!");
  //call begin to start sensor
  dht.begin();
}
```

- **To execute the main process write the void loop()**
  - Create **float h** and **float t** variable to store decimal value
  - **readHumidity()** will read the sensor's humidity value.
  - **readTemperature()** will read the sensor's temperature value.
  - Check if any reads failed and exit early using **isnan()**
  - **Serial.println** used to print data. Print ("Failed to read from DHT sensor!")
  - Return the process using **return()**
  - **Serial.print()** is used to print the value, print Humidity, (h) , , Temperature, (t)
  - Set delay of **2000ms**

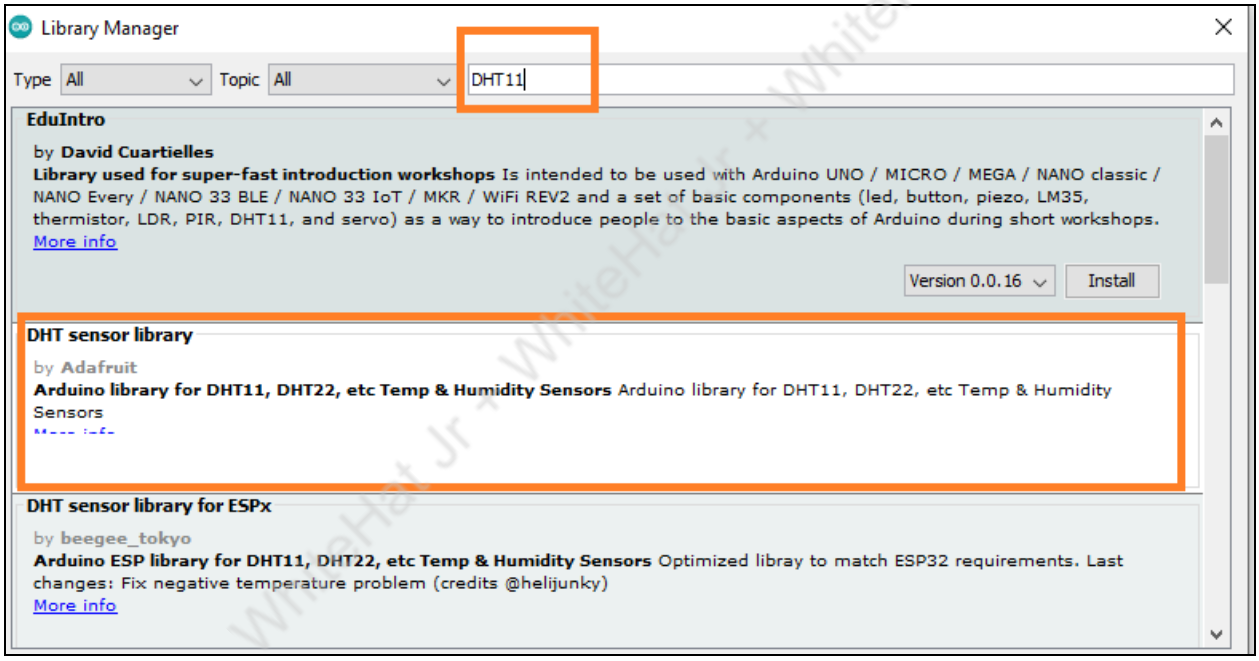
```
void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  // print the result to Terminal
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(",");
  Serial.print("Temperature: ");
  Serial.println(t);
  delay(2000);
}
```

- Compile and upload the program to ESP32 board using Arduino IDE
  - Verify the program on clicking Tick option
  - Upload the program on clicking arrow option
- If the port is not selected, insert the **USB cable** in Computer's port and select the port
  - Go to Tools and select **Serial Monitor**

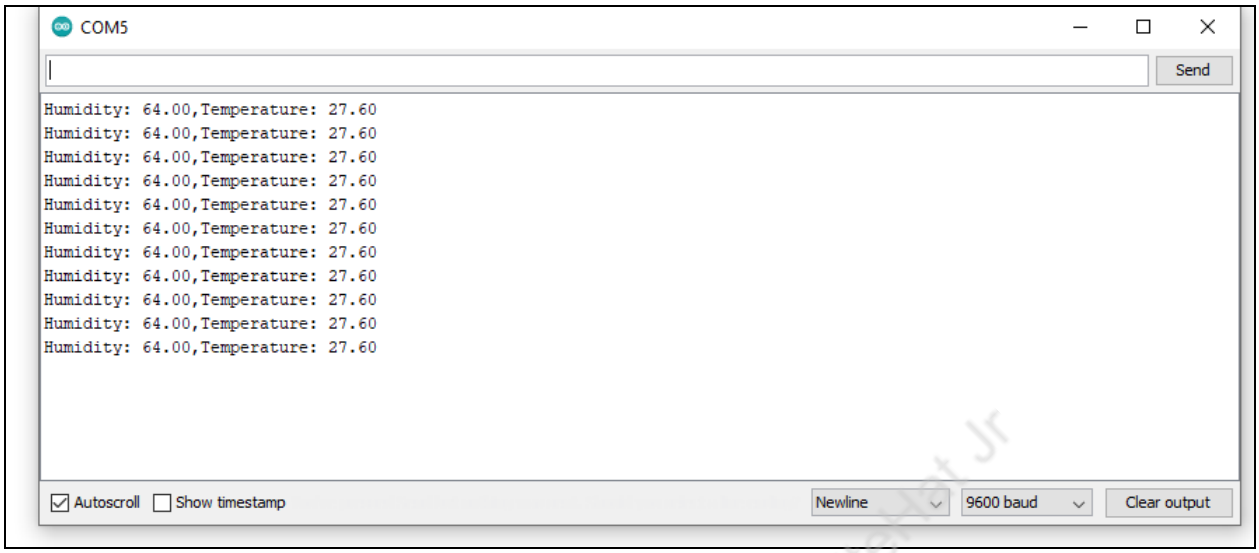
- Rotate the **potentiometer** and see the output
- If an **error message** comes like no such file or directory then use the below method to resolve this error

```
DHT.h: No such file or directory
compilation terminated.
exit status 1
DHT.h: No such file or directory
```

- Go to Tools
  - Click on Manage Libraries
  - Write the component name which need to install
  - Click on Install



- Go to Tools and select Serial Monitor
  - See the **Humidity and Temperature** value



- To verify the analog waveform
  - Go to Tools and select **Serial Plotter**
  - See the Humidity and Temperature value

- Observe the **reading of humidity and temperature**.
- Press the button and check the circuit. As the button is pressed **it conducts current through it** or makes the circuit. As the button is **released it breaks the circuit** and stops the flow of current.

What's NEXT?

In the next class, we will learn about IoT platform

Expand Your Knowledge

To know more about **Servers** click here.