Università degli Studi di Messina

Department of Engineering

Master's Degree in Engineering and Computer Science

Embedded Systems Project

ESP32 - WiFi | Relay | Temperature | Humidity

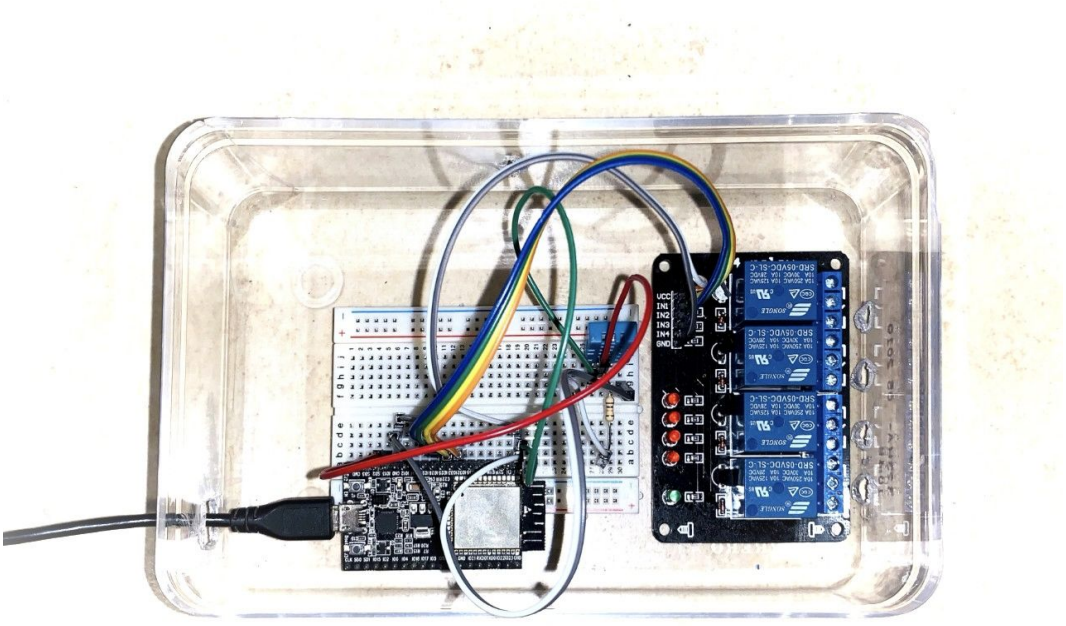
Shanezzar **Sharon**
Student

Prof. Francesco **Longo**
Teacher

Academic Year 2019/2020

Table of contents

1. Introduction



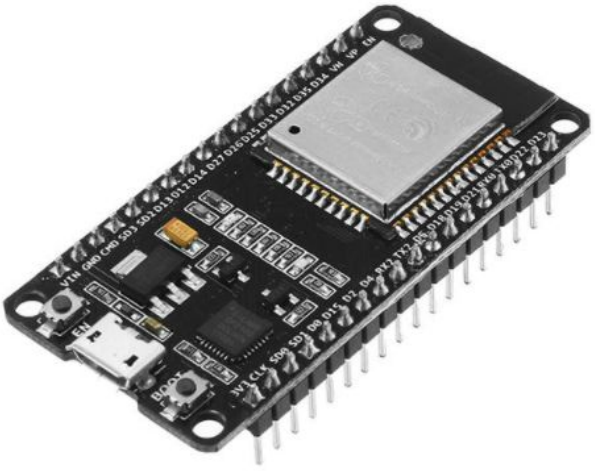
The above figure shows how the project looks at the delivered state. In this project, we are going to control a 4 Channel Relay module and read the temperature and humidity values of the surrounding environment with a DHT11 sensor using the ESP32 development board.

The ESP32 development board connects to the local Wi-Fi network and a DNS (http://shanezzar.local/) is initiated. Hitting this URL connects that client to the ESP32 development board by means of web sockets resulting in synchronized data exchange across all devices. The ESP32 development board also uses SPI Flash File System and a web page is burned in the board which displays temperature, humidity, and heat index gauges along with 4 buttons to toggle the relay modules ON/OFF. All the debug data is shown on the serial monitor at 115200.

2. Hardware Aspects

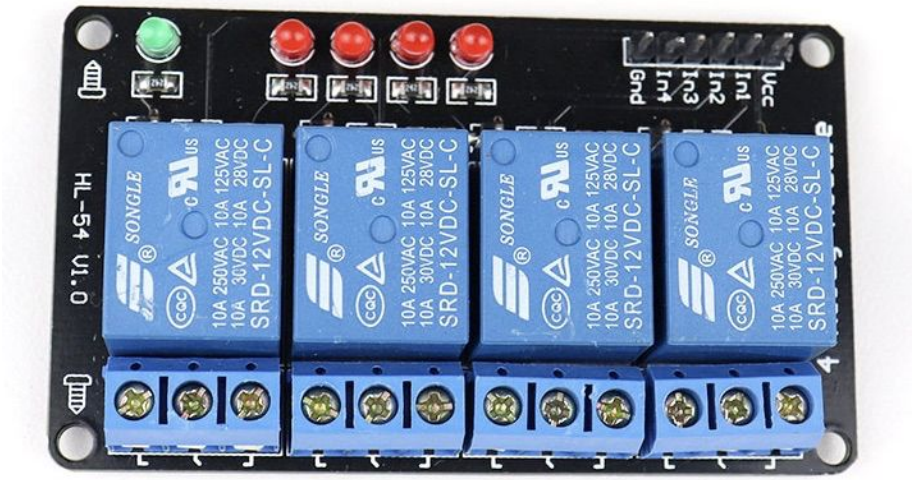
This section describes the hardware components used in this project.

2.1. ESP32 Development Board



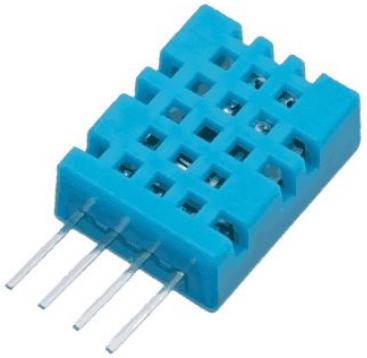
ESP32 is a series of low cost, low power systems-on-chip microcontrollers with integrated Wi-Fi & dual-mode Bluetooth. The ESP32 series employs a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company, and is manufactured by TSMC using their 40 nm process. It is a successor to the ESP8266 microcontroller.

2.2. 4-Channel Relay Module



A Relay is a digital switch to control much higher voltages and currents than your normal Arduino boards do. When inputting a logic voltage, the relay will switch to allow current to flow or cutoff, depending on your wiring. A Relay usually consists of a coil, 1 common terminal, 1 normally closed terminal, and one normally open terminal. When the coil is energized, the common terminal and the normally open terminal will have continuity. This 5V 4-channel relay is equipped with high-current relays that work under AC250V 10A or DC30V 10A.

2.3. DHT11 Sensor



DHT11 is a low cost embedded sensor, which is used to measure temperature (in a range from 0 to 50 degrees Celsius with an accuracy of +-2 C) and moisture (in a range from 20% to 80% with an accuracy of +-5%). It consists of a capacitive humidity sensor that measures humidity in the air.

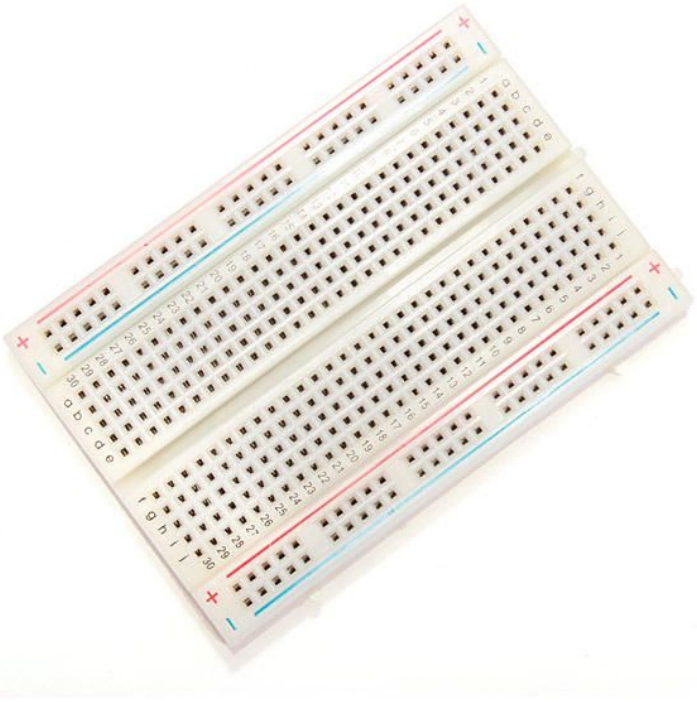
2.4. 10K-OHM Resistor

A passive device that resists the flow of electricity. This resistor will provide 10K Ohms of resistance wherever it is placed and will handle 1/4 watts. Commonly used in PCBs and perf boards, these 10K resistors make excellent pull-ups, pull-downs, and current limiters.

2.5. Jumper Wires

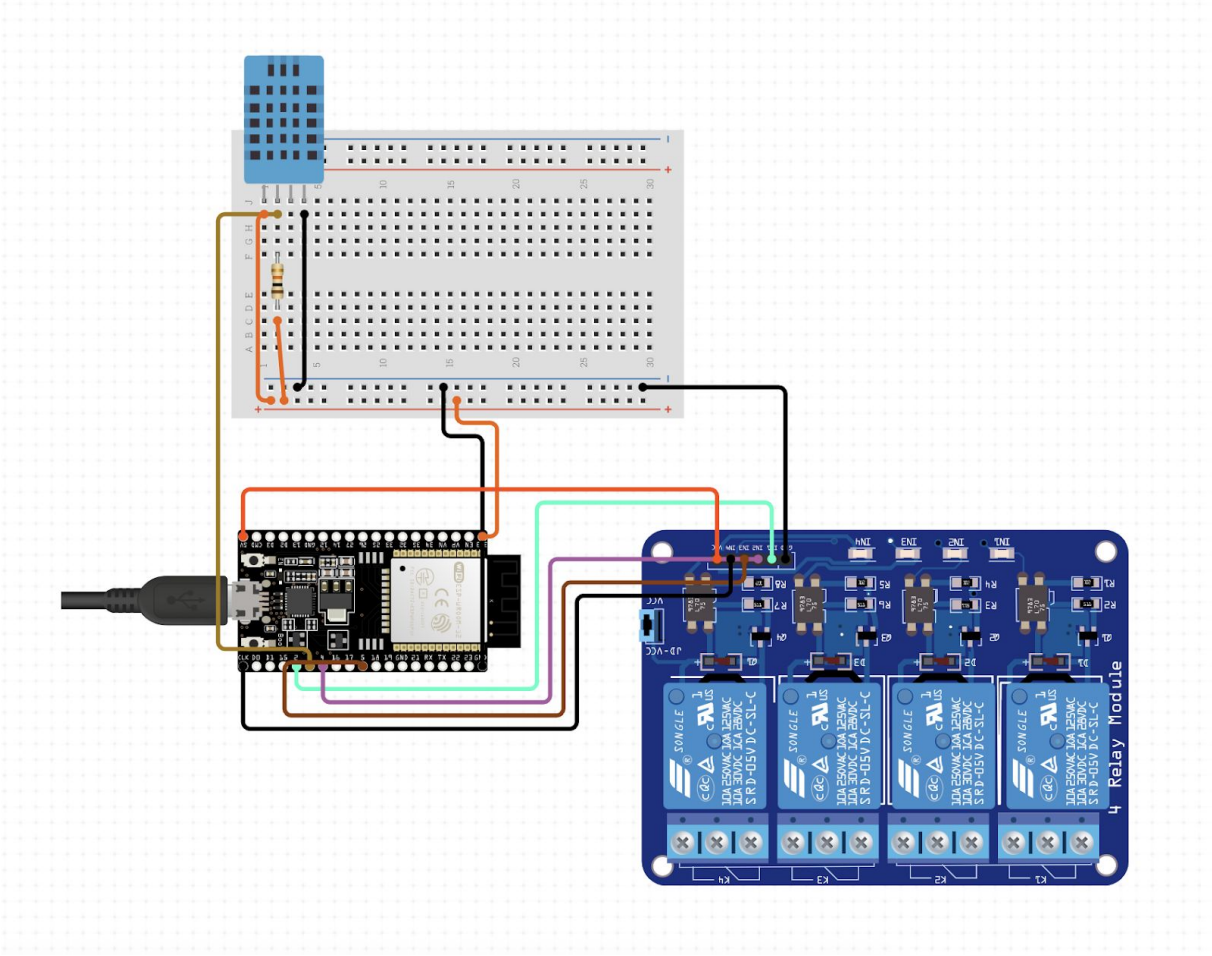
Jumper wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. Fairly simple. In fact, it doesn't get much more basic than jumper wires.

2.6. Bread Board



The breadboard is a circuit construction technique that is designed to allow the rapid creation of circuits without the need for soldering or making permanent connections used for prototyping of electronics. It is a rectangular plastic board with a bunch of tiny holes in it. These holes let you easily insert electronic components to prototype.

2.7. Project Setup



Connect the 4 Channel Relay the module VCC with ESP32 development board 3V3 and GND of the module with the GND of the board. Now connect the IN1 of the module with IO32 of the board, IN2 of the module with IO33 of the board, IN3 of the module with IO25 of the board, and IN4 of the module with IO26 of the board.

As for the DHT11 sensor, connect its DATA to IO13, VDD of the sensor with 3V3 of the board, and GND of the sensor with the GND of the board. Now connect ONE SIDE of the resistor with 3V3 of the board and the OTHER SIDE of the resistor with the DATA of the sensor.

3. Software Aspects

This section describes the software components used in this project. For the Arduino sketch I have used Arduino IDE and for the HTML, CSS and Javascript I have used Visual Studio Code.

3.1. Arduino Sketch

```
#include <WiFi.h>
#include "ESPAsyncWebServer.h"
#include <ESPmDNS.h>
#include "SPIFFS.h"
#include "DHTesp.h"

// defining constants
const char* wifi_name = "Nike0007"; // wifi network name
const char* wifi_pass = "nike0007"; // wifi network password
const char* host = "shanezzar"; // host name to access web

// defining board pins
int relay_pin1 = 32;
int relay_pin2 = 33;
int relay_pin3 = 25;
int relay_pin4 = 26;
int dhtPin = 13;

// defining server
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");
AsyncEventSource events("/events");

AsyncWebSocketClient * lastClient = NULL;

// defining DHT sensor
DHTesp dht;

void setup() {
  // serial port for debugging purposes
```

```
  Serial.begin (115200);

  setupRelaySensor();
  setupSPIFFS();
  setupWifiHost();

  // sockets events
  ws.onEvent(onWsEvent);
  server.addHandler(&ws);

  setupHTTPRoutes();

  // start server
  server.begin();
}

void loop() {
  dhtRead();
  delay(1000);
}

void setupRelaySensor() {
  // initialize relay pins
  pinMode (relay_pin1, OUTPUT);
  pinMode (relay_pin2, OUTPUT);
  pinMode (relay_pin3, OUTPUT);
  pinMode (relay_pin4, OUTPUT);

  // initialize dht
  dht.setup(dhtPin, DHTesp::DHT11);
}

void setupSPIFFS() {
  // initialize SPIFFS
  if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

}

void setupWifiHost() {
  // connecting to wifi and getting ip
  WiFi.begin(wifi_name, wifi_pass);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connection Successful");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // ip to host
  MDNS.begin(host);
  Serial.print("Open http://");
  Serial.print(host);
  Serial.println(".local");

}

void setupHTTPRoutes() {
  // main web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/index.html", "text/html");
  });

  server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/favicon.ico", "image/x-icon");
  });

  // css and javascript
  server.on("/css/bootstrap.min.css", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/css/bootstrap.min.css", "text/css");
  });

  server.on("/css/style.css", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/css/style.css", "text/css");
  });

  server.on("/js/jquery.min.js", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/js/jquery.min.js", "text/js");
  });

  server.on("/js/bootstrap.min.js", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/js/bootstrap.min.js", "text/js");
  });

  server.on("/js/gauge.min.js", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(SPIFFS, "/js/gauge.min.js", "text/js");
  });
```

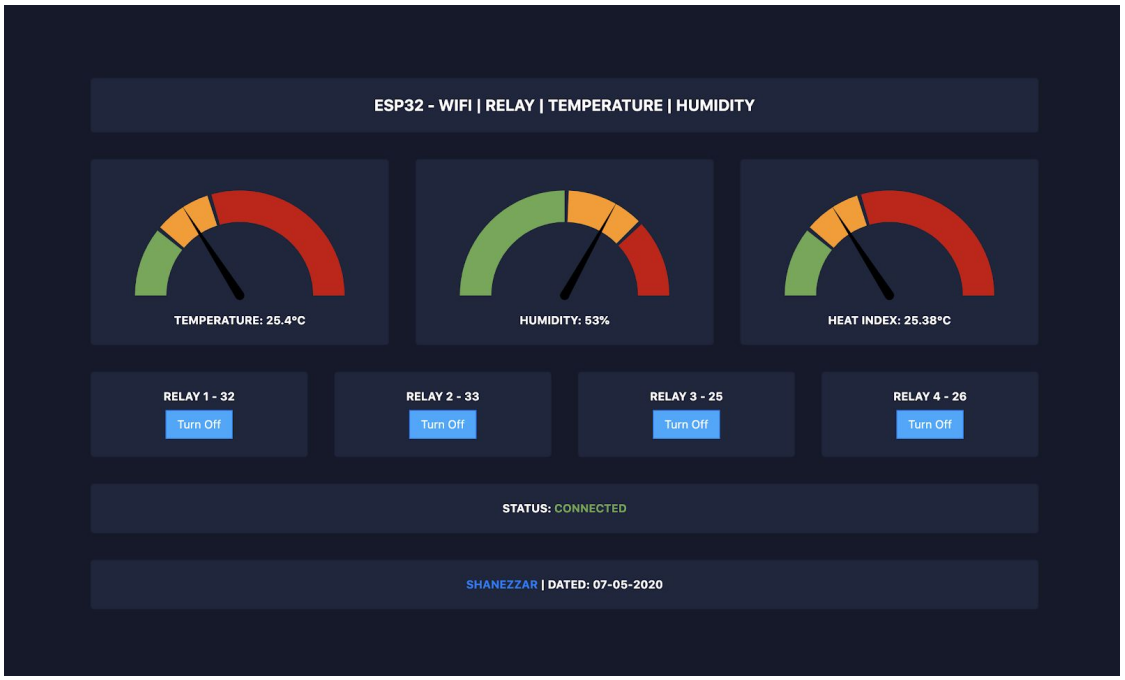
```
    server.on("/js/script.js", HTTP_GET, [](AsyncWebServerRequest * request) {
      request->send(SPIFFS, "/js/script.js", "text/js");
    });

  }

  void onWsEvent(AsyncWebSocket * server, AsyncWebSocketClient * client,
AwsEventType type, void * arg, uint8_t *data, size_t len) {
    // websocket connect event
    if (type == WS_EVT_CONNECT) {
      Serial.println("Websocket client connection received");
      lastClient = client;

      // checking state of relays and forwarding to all clients
      if (digitalRead(relay_pin1) == LOW) {
        sendDataToAllWS("on-1");
      } else if (digitalRead(relay_pin1) == HIGH) {
        sendDataToAllWS("off-1");
      }

      if (digitalRead(relay_pin2) == LOW) {
        sendDataToAllWS("on-2");
      } else if (digitalRead(relay_pin2) == HIGH) {
        sendDataToAllWS("off-2");
      }

      if (digitalRead(relay_pin3) == LOW) {
        sendDataToAllWS("on-3");
      } else if (digitalRead(relay_pin3) == HIGH) {
        sendDataToAllWS("off-3");
      }

      if (digitalRead(relay_pin4) == LOW) {
        sendDataToAllWS("on-4");
      } else if (digitalRead(relay_pin4) == HIGH) {
        sendDataToAllWS("off-4");
      }

    // websocket disconnect event
    } else if (type == WS_EVT_DISCONNECT) {
      Serial.println("Websocket client connection finished");

      // websocket data receive event
    } else if (type == WS_EVT_DATA) {
```

```
    String dataReceived = "";
    for (int i = 0; i < len; i++) {
      dataReceived += (char) data[i];
    }
    toggleRelay(dataReceived);
  }

}

void toggleRelay(String relayId) {
  // checking if any client is connected
  if (lastClient != NULL && lastClient->status() == WS_CONNECTED) {
    // if id matches then toggle relays
    if (relayId == "on-1") {
      digitalWrite(relay_pin1, LOW);
    }
    if (relayId == "off-1") {
      digitalWrite(relay_pin1, HIGH);
    }
    if (relayId == "on-2") {
      digitalWrite(relay_pin2, LOW);
    }
    if (relayId == "off-2") {
      digitalWrite(relay_pin2, HIGH);
    }
    if (relayId == "on-3") {
      digitalWrite(relay_pin3, LOW);
    }
    if (relayId == "off-3") {
      digitalWrite(relay_pin3, HIGH);
    }
    if (relayId == "on-4") {
      digitalWrite(relay_pin4, LOW);
    }
    if (relayId == "off-4") {
      digitalWrite(relay_pin4, HIGH);
    }

    // after changing state of relays forward to all clients to keep things synced
    sendDataToAllWS(relayId);
  }

}

void dhtRead() {
```

```
  // reading temperature and humidity values
  TempAndHumidity newValues = dht.getTempAndHumidity();

  // check if any reads failed and exit early
  if (dht.getStatus() != 0) {
    return;
  }

  // calculating heat index
  float heatIndex = dht.computeHeatIndex(newValues.temperature, newValues.humidity);

  // forwarding respective values to clients
  sendDataToAllWS("T: " + String(newValues.temperature));
  sendDataToAllWS("H: " + String(newValues.humidity));
  sendDataToAllWS("I: " + String(heatIndex));
}

void sendDataToAllWS(String dataReceived) {
  // sending data to all clients
  ws.textAll(dataReceived);
}

// coder: Shanezzar (www.shanezzar.com)
// dated: 07-05-2020
```

The above is the sketch code for the working of the project. It consists of comments at each level which informs the working of each block of code.

3.2. Web App



The web app which is burned using SPI Flash File System on the board consist of an index.html supported by bootstrap, jquery, and custom css and javascript. The connection is established via web sockets to the board when hitting the DNS URL (http://shanezzar.local/). It shows three gauges as shown above for temperature, humidity, and heat index which shows the values obtained from the ESP32 development board read by a DHT11 sensor connected to it. The web app also has four buttons which are used to toggle the relay modules ON/OFF. This web app can be found in a folder named "data" within the same folder where the sketch .ino is placed. The data folder is uploaded to the ESP32 development board using ESP32 Sketch Data Upload along with .ino sketch to the board.

3.3. HTML

```
<!--
coder: Shanezzar (www.shanezzar.com)
dated: 07-05-2020
-->
<!DOCTYPE html>
<html lang="en">
<head>
 <title>ESP32 - WiFi | Relay | Temperature | Humidity</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="icon" type="image/x-icon" href="favicon.ico">
 <link rel="stylesheet" href="css/bootstrap.min.css">
 <link rel="stylesheet" href="css/style.css">

 <script src="js/jquery.min.js"></script>
 <script src="js/bootstrap.min.js"></script>
 <script src="js/gauge.min.js"></script>
</head>
<body>

<div class="container">
 <div class="center-all">
   <div class="row">
     <div class="col-sm-12">
       <div class="card">
         <div class="card-body text-center">
           <h5 class="card-title heading">ESP32 - WiFi | Relay | Temperature |
Humidity</h5>
         </div>
       </div>
     </div>
   </div>
   <div class="row">
     <div class="col-sm-4">
       <div class="card">
         <div class="card-body text-center">
           <!-- gauge for temperature -->
```

```
          <canvas id="chartTemp"></canvas>
          <h5 class="card-title">Temperature: <span id="temp">0</span>&#8451;</h5>
        </div>
      </div>
    </div>
    <div class="col-sm-4">
      <div class="card">
        <div class="card-body text-center">
          <!-- gauge for humidity -->
          <canvas id="chartHumd"></canvas>
          <h5 class="card-title">Humidity: <span id="humd">0</span>&#37;</h5>
        </div>
      </div>
    </div>
    <div class="col-sm-4">
      <div class="card">
        <div class="card-body text-center">
          <!-- gauge for heat index -->
          <canvas id="chartIndex"></canvas>
          <h5 class="card-title">Heat Index: <span id="index">0</span>&#8451;</h5>
        </div>
      </div>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-3">
      <div class="card">
        <div class="card-body text-center">
          <!-- relay 1 switch -->
          <h5 class="card-title">Relay 1 - 32</h5>
          <button class="btn btn-primary relay1 relayOn">Turn Off</button>
        </div>
      </div>
    </div>
    <div class="col-sm-3">
      <div class="card">
        <div class="card-body text-center">
          <!-- relay 2 switch -->
          <h5 class="card-title">Relay 2 - 33</h5>
          <button class="btn btn-primary relay2 relayOn">Turn Off</button>
        </div>
```

```
      </div>
    </div>
    <div class="col-sm-3">
      <div class="card">
        <div class="card-body text-center">
          <!-- relay 3 switch -->
          <h5 class="card-title">Relay 3 - 25</h5>
          <button class="btn btn-primary relay3 relayOn">Turn Off</button>
        </div>
      </div>
    </div>
    <div class="col-sm-3">
      <div class="card">
        <div class="card-body text-center">
          <!-- relay 4 switch -->
          <h5 class="card-title">Relay 4 - 26</h5>
          <button class="btn btn-primary relay4 relayOn">Turn Off</button>
        </div>
      </div>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-12">
      <div class="card">
        <div class="card-body text-center">
          <!-- status of connection -->
          <h5 class="card-title">Status: <span id="status">...</span></h5>
        </div>
      </div>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-12">
      <div class="card">
        <div class="card-body text-center">
          <!-- info -->
          <h5 class="card-title"><a href="https://www.shanezzar.com/"
target="_blank">Shanezzar</a> | Dated: 07-05-2020</h5>
        </div>
      </div>
    </div>
```

```
   </div>
 </div>
</div>

<script src="js/script.js"></script>
</body>
</html>
```

The above is the HTML code of the index.html file. It consists of standard HTML structure with html, head, title, link, script, body tags. It uses bootstrap for styling and jquery for javascript functionality. The id "chartTemp", "chartHumd" and "chartHeatIndex" is used with the canvas tag to show gauge charts. The id "temp", "humd" and "heatIndex" is used for span tag to show numeric values with unit of temperature, humidity, and heat index received from ESP32 development board web socket events. The buttons uses "relay1", "relay2", "relay3" and "relay4" class to identify respective relays, also "relayOn" and "relayOff" class to toggle relay ON/OFF. An id "status" is also defined for the span tag to show the status of the web socket state.

3.4. CSS

```
body {
    text-transform: uppercase;
    background-color: #15192A;
    color: #ffffff;
}

.center-all {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
}

.heading {
    font-size: large !important;
}

.card {
    margin-top: 15px;
    margin-bottom: 15px;
    background-color: #1F263D;
}

.card-title {
    margin-bottom: 0px;
    font-size: small;
    font-weight: bold;
}

.btn-primary {
    background-color: #2CA8FF !important;
    border-radius: 0px;
    margin-top: 8px;
    font-size: small;
}

@media only screen and (max-width: 575px) {
    .center-all {
```

```
        position: absolute;
        top: 2%;
        transform: translateY(0%);
    }
}
```

There is a CSS file in the data > css folder namely style.css which consists of custom styling for the web app. The classes change font properties, colors, margins, borders of various elements of the HTML page. The class .center-all centers vertically the content in it and when the screen width reduces to 575px the .center-all properties are reset to starting from the top.

3.5. Javascript

```
$(document).ready(function () {
    // defining variables
    var targetTemp = document.getElementById("chartTemp");
    var targetIndex = document.getElementById("chartIndex");
    var targetHumd = document.getElementById("chartHumd");
    var gaugeTemp, gaugeIndex, gaugeHumd;

    var relay1 = $('.relay1');
    var relay2 = $('.relay2');
    var relay3 = $('.relay3');
    var relay4 = $('.relay4');

    initGauge();

    // webSockets
    var ws = new WebSocket("ws://" + window.location.hostname + "/ws");
    // webSocket connection open
    ws.onopen = function () {
        $('#status').html("Connected");
        $('#status').css('color', '#6AA84F');
    };
    // webSocket connection close
    ws.onclose = function () {
        $('#status').html("Disconnected");
        $('#status').css('color', '#CC0200');
    };
    // webSocket data receive
    ws.onmessage = function (evt) {
        var data = evt.data;

        // if id matches then toggle relays
        if (data == "on-1") {
            relayOn(relay1);
        }
        if (data == "off-1") {
            relayOff(relay1);
        }
```

```
    if (data == "on-2") {
        relayOn(relay2);
    }
    if (data == "off-2") {
        relayOff(relay2);
    }
    if (data == "on-3") {
        relayOn(relay3);
    }
    if (data == "off-3") {
        relayOff(relay3);
    }
    if (data == "on-4") {
        relayOn(relay4);
    }
    if (data == "off-4") {
        relayOff(relay4);
    }

    if (data.includes("T: ")) {
        var temperature = data.replace("T: ", "");
        // setting gauge temperature value
        gaugeTemp.set(temperature);
        $('#temp').html(Math.round(temperature * 10) / 10);
    }
    if (data.includes("H: ")) {
        var humidity = data.replace("H: ", "");
        // setting gauge humidity value
        gaugeHumd.set(humidity);
        $('#humd').html(Math.round(humidity));
    }
    if (data.includes("I: ")) {
        var heatIndex = data.replace("I: ", "");
        // setting gauge heat index value
        gaugeIndex.set(heatIndex);
        $('#index').html(heatIndex);
    }
};

// sending commands to websockets
relay1.click(function() {
```

```
    if (relay1.hasClass("relayOn")) {
        ws.send("off-1");
        relayOff(relay1);
    } else {
        ws.send("on-1");
        relayOn(relay1);
    }
});
relay2.click(function() {
    if (relay2.hasClass("relayOn")) {
        ws.send("off-2");
        relayOff(relay2);
    } else {
        ws.send("on-2");
        relayOn(relay2);
    }
});
relay3.click(function() {
    if (relay3.hasClass("relayOn")) {
        ws.send("off-3");
        relayOff(relay3);
    } else {
        ws.send("on-3");
        relayOn(relay3);
    }
});
relay4.click(function() {
    if (relay4.hasClass("relayOn")) {
        ws.send("off-4");
        relayOff(relay4);
    } else {
        ws.send("on-4");
        relayOn(relay4);
    }
});

// toggling relay classes
function relayOn(relay) {
    relay.removeClass("relayOff");
    relay.addClass("relayOn");
    relay.html('Turn Off');
```

```
}
function relayOff(relay) {
    relay.removeClass("relayOn");
    relay.addClass("relayOff");
    relay.html('Turn On');
}

function initGauge() {
    // color values for gauge
    var colorValuesTempIndex = [
        {strokeStyle: "#6AA84F", min: 0, max: 17},  // green
        {strokeStyle: "#FF9903", min: 18, max: 32}, // orange
        {strokeStyle: "#CC0200", min: 33, max: 80} // red
    ];
    var colorValuesHumd = [
        {strokeStyle: "#6AA84F", min: 0, max: 40},  // green
        {strokeStyle: "#FF9903", min: 41, max: 60}, // orange
        {strokeStyle: "#CC0200", min: 61, max: 80} // red
    ];

    // options for gauge
    var optsTempIndex = {
        angle: 0,
        lineWidth: 0.3,
        radiusScale: 1,
        pointer: {
            length: 0.6,
            strokeWidth: 0.035,
            color: '#000000'
        },
        limitMax: false,
        limitMin: false,
        colorStart: '#6FADCF',
        colorStop: '#8FC0DA',
        strokeColor: '#E0E0E0',
        generateGradient: true,
        highDpiSupport: true,
        staticZones: colorValuesTempIndex
    };
    var optsHumd = {
        angle: 0,
```

```
            lineWidth: 0.3,
            radiusScale: 1,
            pointer: {
                length: 0.6,
                strokeWidth: 0.035,
                color: '#000000'
            },
            limitMax: false,
            limitMin: false,
            colorStart: '#6FADCF',
            colorStop: '#8FC0DA',
            strokeColor: '#E0E0E0',
            generateGradient: true,
            highDpiSupport: true,
            staticZones: colorValuesHumd
        };

        // setting gauges
        gaugeTemp = new Gauge(targetTemp).setOptions(optsTempIndex);
        gaugeIndex = new Gauge(targetIndex).setOptions(optsTempIndex);
        gaugeHumd = new Gauge(targetHumd).setOptions(optsHumd);

        // initializing gauge with values
        gaugeTemp.set(0);
        gaugeIndex.set(0);
        gaugeHumd.set(0);
    }

});
```

There is a JS file in the data > js folder namely script.js which consists of custom scripting for the web app. The target HTML elements are defined referencing to gauge and relay buttons under the "// defining variables" comment. Below that there is a function of "initGauge()" which sets options for the gauge like colors, minimum, medium, maximum values, etc. and initializes gauges with a default value of 0. Below the initGauge function, there is a function of web socket open "ws.onopen" defined which is called when the connection of that web socket opens to that client and when this happens the

value of id "status" HTML element is set to "Connected" and sets the color of status element green. Below this a function of web socket close "ws.onclose" is defined which is called when the connection of that web socket closes to that client and when this happens the value of id "status" HTML element is set to "Disconnected" and sets the color of status element red. Below the web socket close function, another web socket function is defined "ws.onmessage" which is called when the client receives data from the ESP32 development board. Based upon the received data there are conditions defined which turn ON/OFF the respective relays, sets the value of gauges, and the values of "temp", "humd" and "index" HTML elements. Below this, the individual relay1, relay2, relay3, and relay4 on click button functions are defined which when executed sends commands to the ESP32 development board to switch ON/OFF the relays from the hardware side. And finally, we have two more functions "relayON()" and "relayOFF" which toggles the classes of HTML elements from the web app side, and based on those toggle classes the web app knows the synced current state of the relays.

4. Conclusion

To recapitulate, the device works well and all the components involved in this project have fulfilled their purpose in the right manner. The values measured by the sensor were compared with the other devices and are accurate. All the clients connected at the same time to the board worked in synchronization and the relays toggled on all the clients when sending commands from one of them.

The main problem faced during the making of this project was the COVID-19 because everything is closed and getting components for the project is quite a hassle.

5. References

5.1. ESPAsyncWebServer:
https://github.com/me-no-dev/ESPAsyncWebServer

5.2. ESPmDNS:
https://github.com/espressif/arduino-esp32/tree/master/libraries/ESPmDNS

5.3. SPIFFS:
https://github.com/espressif/arduino-esp32/tree/master/libraries/SPIFFS

5.4. DHTesp: https://github.com/beegee-tokyo/DHTesp

5.5. ESP32 Sketch Data Upload:
https://github.com/me-no-dev/arduino-esp32fs-plugin