ECE LABORATORY

DREXEL UNIVERSITY

To: **Dr. Peters**

From: **Ehi Simon**

Re: **ECE 304 Lab 4 - JSON, APIs, and Thunder Client**

PURPOSE:

The purpose of this week's lab is to understand how JSON, APIs, and Thunder Client work. The lab is supposed to help build these skills especially how they interact with the ESP32 to build a webpage and to control a circuit.

Discussion:

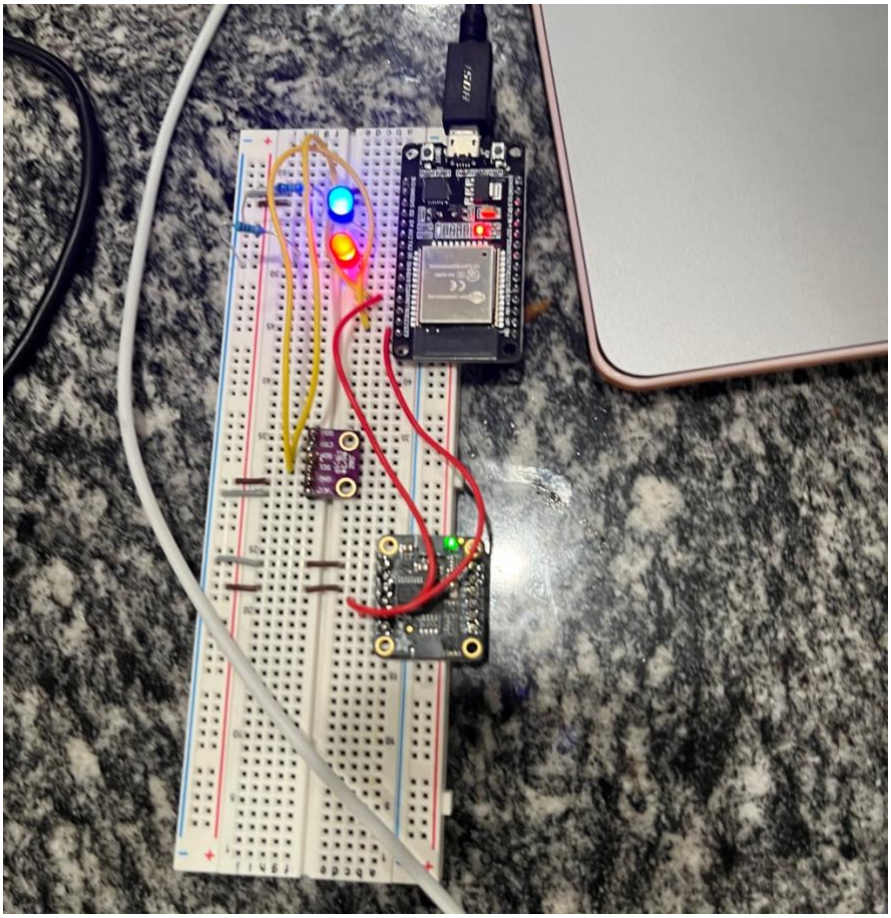

Fig. 1. Circuit Connection for Project 2

The circuit for the lab was built like the one above. It consists of 2 330Ω resistors, a red LED, a blue LED, a BME280 Environmental Sensor, an Adafruit BNO085 IMU, and an ESP32S microcontroller.

Main.cpp

In my main.cpp file, I initialized the multiple libraries that were needed for the sensors to work and provide readings. I also included libraries to get the ESP32 to connect to STA, and libraries needed to interact with a server in JSON. I had to install the Adafruit JSON library prior. The LEDs are defined, the sea level reference pressure is defined and the BNO08X chip is reset. The BME object is created and the network SSID and password. The web server is then opened on port 80. This can all be found in the figure below:

src > main.cpp > ...

```
1   // Import Libraries
2   #include <Arduino.h>
3   #include <WiFi.h>
4   #include <WebServer.h>
5   #include <Arduino_JSON.h>
6   #include <Adafruit_BME280.h>
7   #include <Arduino.h>
8   #include <Adafruit_BNO08x.h>
9
10  // Define constants
11  const int blue_LED_pin=4;
12  const int red_LED_pin=5;
13  const int freq = 5000;
14  const int ledChannel = 0;
15  const int resolution = 8;
16
17  // Define Sea level reference pressure (hPa)
18  #define SEALEVELPRESSURE_HPA (1013.25)
19  #define BNO08X_RESET -1
20
21  struct euler_t {
22    float yaw;
23    float pitch;
24    float roll;
25  } ypr;
26
27  // Pin Request Status
28  bool LED_request = LOW;
29
30  // Create object for DHT
31  Adafruit_BME280 bme; // DHT object
32
33  /* Put your SSID & Password */
34  const char* ssid = "Verizon_C4ZKVV";  // Enter SSID here
35  const char* password = "dna-blear7-bow";  //Enter Password here
36
37  WebServer server(80); // Web Server open on port 80
38
```

Fig. 2. Figure Showing Initialization of Libraries and Variable Definitions

```
/* Put your SSID & Password */
const char* ssid = "Verizon_C4ZKVV";  // Enter SSID here
const char* password = "dna-blear7-bow";  //Enter Password here

WebServer server(80); // Web Server open on port 80

Adafruit_BNO08x  bno08x(BNO08X_RESET);
sh2_SensorValue_t sensorValue;
sh2_SensorId_t reportType = SH2_ARVR_STABILIZED_RV;
long reportIntervalUs = 5000;

void setReports(sh2_SensorId_t reportType, long report_interval) {
  Serial.println("Setting desired reports");
  if (! bno08x.enableReport(reportType, report_interval)) {
    Serial.println("Could not enable stabilized remote vector");
  }
}

void quaternionToEuler(float qr, float qi, float qj, float qk, euler_t* ypr, bool degrees = false) {

    float sqr = sq(qr);
    float sqi = sq(qi);
    float sqj = sq(qj);
    float sqk = sq(qk);

    ypr->yaw = atan2(2.0 * (qi * qj + qk * qr), (sqi - sqj - sqk + sqr));
    ypr->pitch = asin(-2.0 * (qi * qk - qj * qr) / (sqi + sqj + sqk + sqr));
    ypr->roll = atan2(2.0 * (qj * qk + qi * qr), (-sqi - sqj + sqk + sqr));

    if (degrees) {
      ypr->yaw *= RAD_TO_DEG;
      ypr->pitch *= RAD_TO_DEG;
      ypr->roll *= RAD_TO_DEG;
    }
}
```

Fig. 3. Figure Showing Setup for Yaw, Pitch, and Roll Values

The figure above shows the setup to obtain the yaw, pitch, and roll. The BNO08X chip is reset, and all the necessary initializations and calculations are performed to obtain the values.

In the figure below, I modified the handle_OnConnect function to include the yaw, pitch, and roll values in the doc under the JSONVar class. This class also contains the temperature, humidity, altitude, and pressure taken from the BME 280 sensor.

```
void handle_OnConnect() {
  while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Reading events");
  Serial.println("Getting BME280 Measurements");

  LED_request = LOW;
  digitalWrite(blue_LED_pin, LOW);
  if (bno08x.getSensorEvent(&sensorValue)) {
  quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

  static long last = 0;
  long now = micros();
  long time = now - last;
}
  JSONVar doc;
  doc["yaw"]= ypr.yaw;
  doc["pitch"] = ypr.pitch;
  doc["roll"] = ypr.roll;
  doc["Temperature"] = bme.readTemperature();
  doc["Humidity"] = bme.readHumidity();
  doc["Pressure"] = bme.readPressure()/1000;
  doc["Altitude"] = bme.readAltitude(SEALEVELPRESSURE_HPA);

  delay(2000);
  server.send(200, "application/json", JSON.stringify(doc));
}
```

Fig. 4. Figure Showing Handle_OnConnect Function

```
void handle_led_change()
  {
  Serial.println("Changing blue LED status");
  String LED_status;
  LED_request = !LED_request;
  digitalWrite(blue_LED_pin, LED_request);
  if (LED_request){
    LED_status = "ON";
  }
  else{
    LED_status = "OFF";
  }

   if (bno08x.getSensorEvent(&sensorValue)) {
  quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

  static long last = 0;
  long now = micros();
  long time = now - last;
}
  JSONVar doc;
  doc["yaw"]= ypr.yaw;
  doc["pitch"] = ypr.pitch;
  doc["roll"] = ypr.roll;
  doc["Temperature"] = bme.readTemperature();
  doc["Humidity"] = bme.readHumidity();
  doc["Pressure"] = bme.readPressure()/1000;
  doc["Altitude"] = bme.readAltitude(SEALEVELPRESSURE_HPA);
  doc["Red_LED"] = "ON";
  doc["Blue_LED"] = LED_status;
  delay(2000);
  server.send(200, "application/json", JSON.stringify(doc));
}
```

Fig. 5. Figure Showing Handle_led_change Function

The figure above shows the Handle_led_change function. This function does almost the same as the previous one, but this one has the ability to toggle the blue LED off. The doc class is also updated and has the states of the LED shown on the webpage. Red is set to on because it is not controlled by the user, and blue is set to LED_status, which changes every time the user loads the webpage.

The figure below shows the LED value post and get functions. This function allows us to change the LED value on the webpage and the circuit by using post and get commands. They also return values obtained from the sensors on the circuit.

```
void set_LED_value_post(){
  Serial.println("Setting Values of LEDs");
  Serial.println(server.hasArg("plain"));
  String body = server.arg("plain");
  Serial.println(body);
  JSONVar myObject = JSON.parse(body);

  if (myObject.hasOwnProperty("blueled")){
    digitalWrite(blue_LED_pin,(int) myObject["blueled"]);
    }

  if (myObject.hasOwnProperty("redled")){
    ledcWrite(ledChannel,(int) myObject["redled"]);
    }

  server.send(200,"application/json",JSON.stringify(myObject));
  }

void set_LED_value_get(){
  Serial.println("Setting Value of LEDs via query parameters");
  if (bno08x.getSensorEvent(&sensorValue)) {
  quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

  static long last = 0;
  long now = micros();
  long time = now - last;
}
  JSONVar doc;
  doc["yaw"]= ypr.yaw;
  doc["pitch"] = ypr.pitch;
  doc["roll"] = ypr.roll;
  doc["Temperature"] = bme.readTemperature();
  doc["Humidity"] = bme.readHumidity();
  doc["Pressure"] = bme.readPressure()/1000;
  doc["Altitude"] = bme.readAltitude(SEALEVELPRESSURE_HPA);

  if (server.hasArg("blueled")){
    String b=server.arg("blueled");
    doc["Blue_LED"]= b.toInt();
    digitalWrite(blue_LED_pin, b.toInt());
  }

  if (server.hasArg("redled")){
    String r=server.arg("redled");
    doc["Red_LED"]= r.toInt();
    ledcWrite(ledChannel,r.toInt());
  }
  delay(2000);
  server.send(200, "application/json", JSON.stringify(doc));
}
```

Fig. 6. Figure Showing POST and GET Functions

```
void setup() {
  Serial.begin(115200);
  delay(100);
  pinMode(blue_LED_pin, OUTPUT);
  ledcSetup(ledChannel, freq, resolution);
  ledcAttachPin(red_LED_pin, ledChannel);
  bool status;
    status = bme.begin(0x76);
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
    }

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

  //Connect to your local wifi network
  WiFi.begin(ssid, password);
  delay(100);

  //check wifi is connected to wifi network
  while (WiFi.status() != WL_CONNECTED){
    delay(1000);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected!");
  Serial.print("ESP32 Network Address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.on("/ledchange", handle_led_change);
  server.on("/led_set_get", HTTP_GET, set_LED_value_get);
  server.on("/led_set_post",HTTP_POST,set_LED_value_post);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");
}

void loop() {
  server.handleClient();
}
```

Fig. 7. Figure Showing Setup and Loop Functions

The figure above shows the setup function of the file. This setup function does more than those in previous projects. It sets the LED pin modes. It also initializes the BME280 sensor and the BNO08X chip. It then connects to the local Wi-Fi network and turns on the server. The setup function makes it possible to open the webpage and GET or POST from the page. The loop function simply keeps handling the client.

The final result consists of multiple webpages and changes from GET/POST requests. They can be found in the figures below.

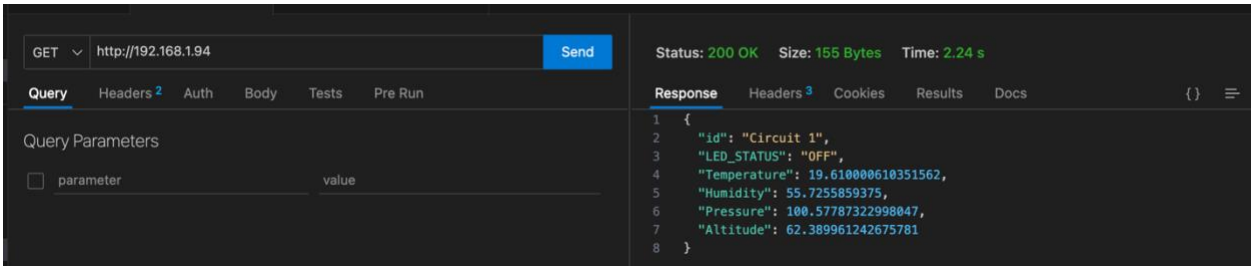

Fig. 7. Figure Showing GET request made to webpage

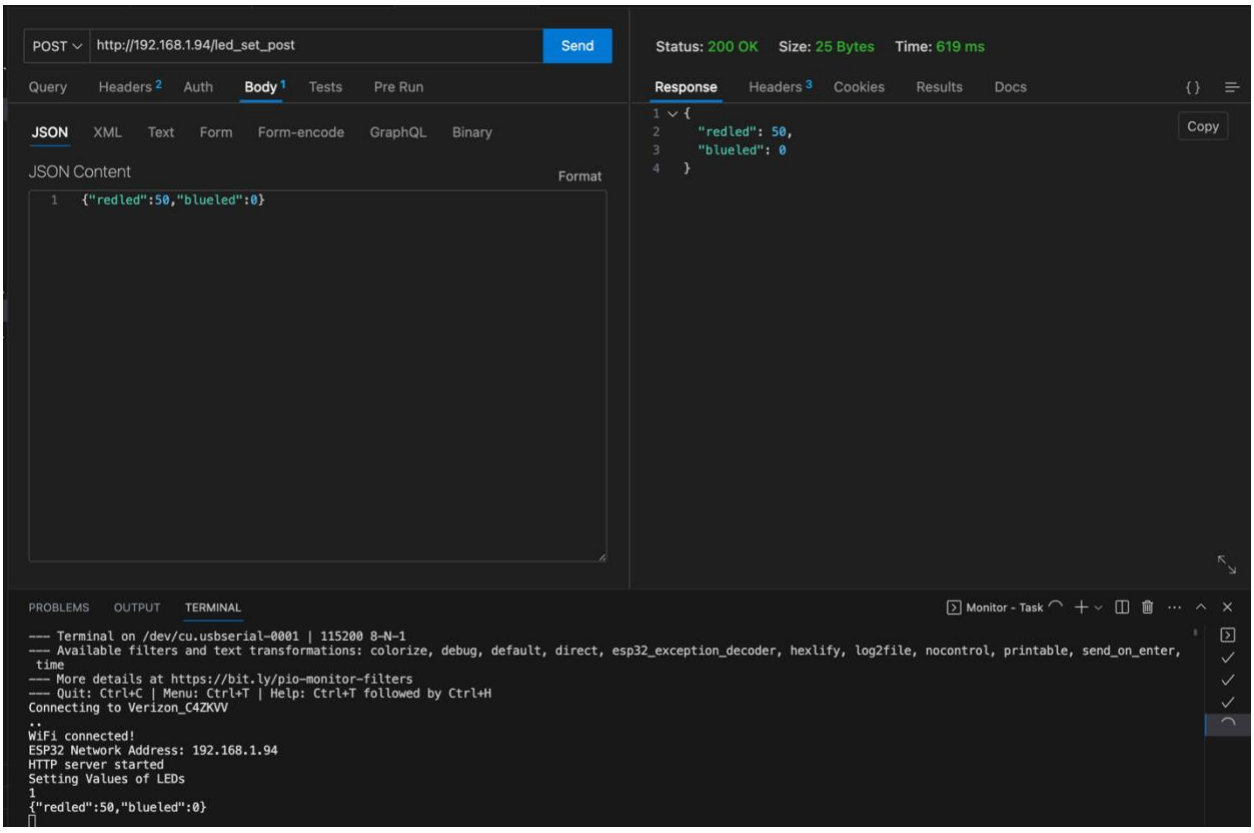

Fig. 8. Figure Showing POST request made to webpage

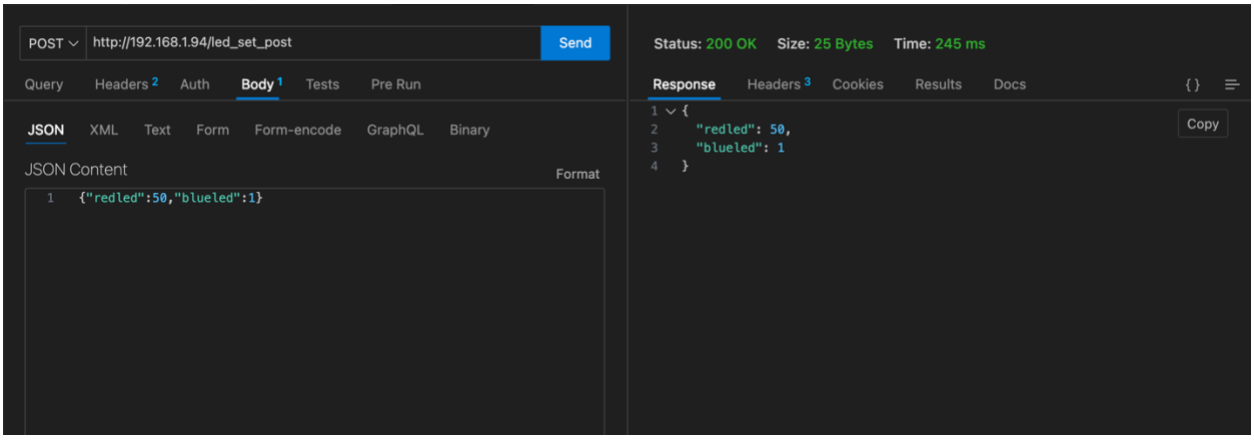

Fig. 9. Figure Showing POST request made to webpage led_set_post

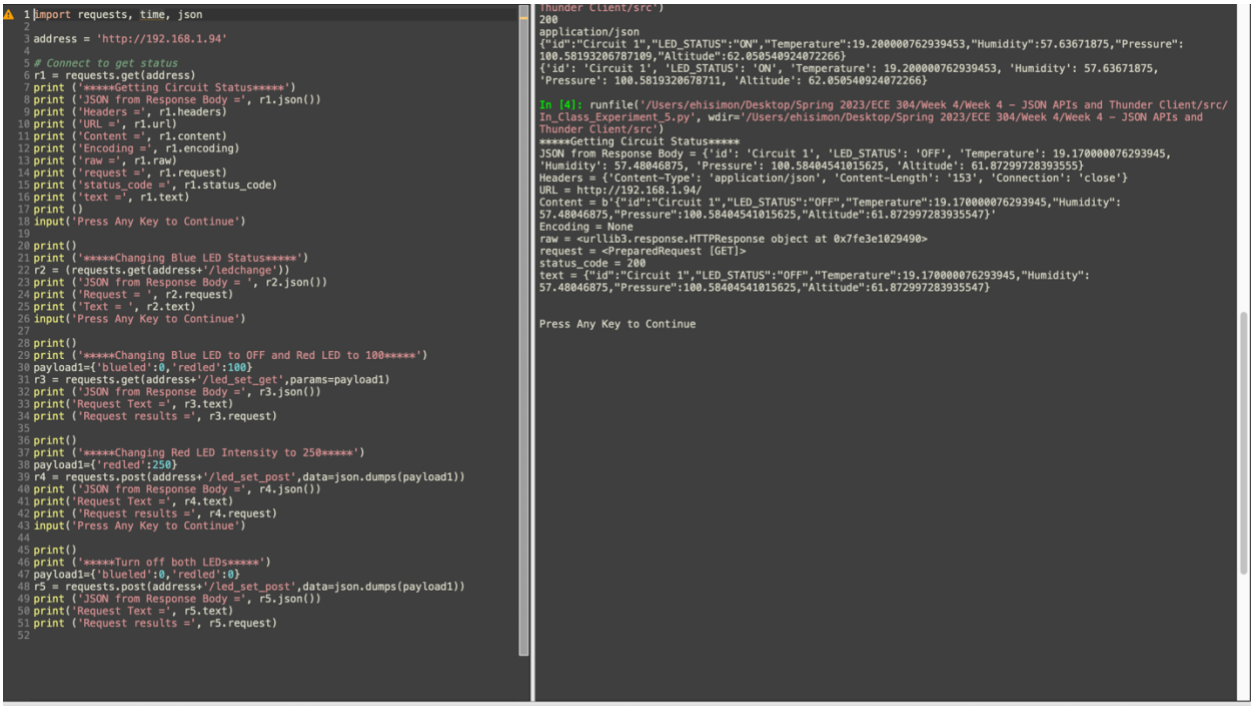

Fig. 10. Figure Showing Python Output After Changing Intensity and LED Status

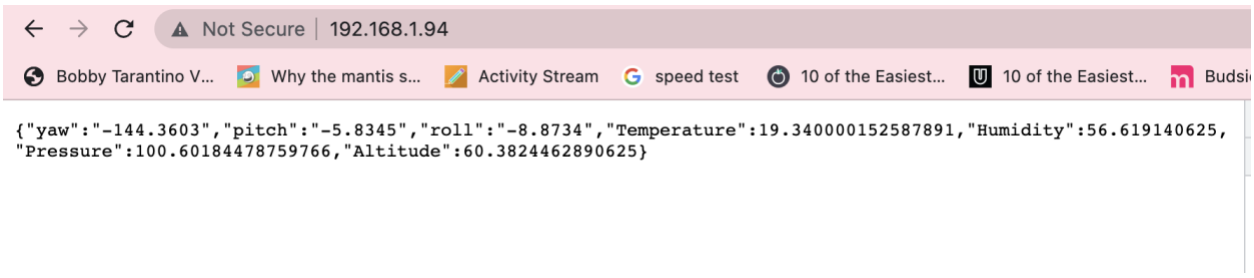

Fig. 11. Figure Showing Output on Webpage on Chrome

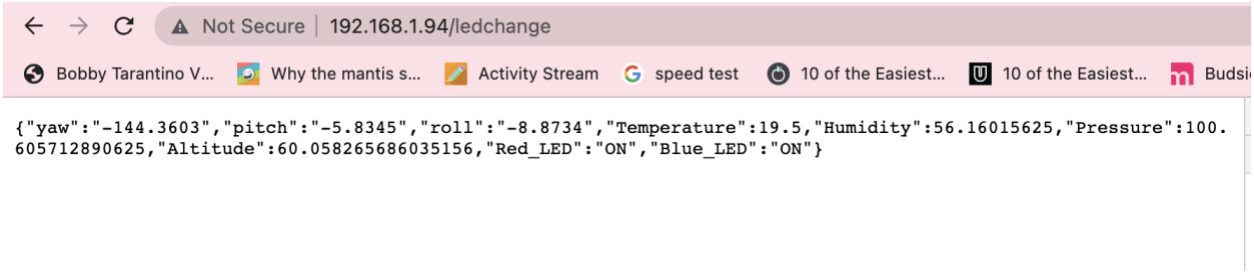

Fig. 12. Figure Showing Output on Webpage ledchange on Chrome

Conclusion

In this experiment, I learned about GET and POST requests. I also understand JSON and APIs more than before.