Experiment No-4

Subject-Computer Laboratory II-Industrial Internet of Things

Class-BE AI &DS

Aim: Write a Program to design and develop a user interface for monitoring and controlling CPS system

Software Requirement: Arduino IDE

Hardware Requirement: ESP-WROOM 32 board, Micro USB Data Cable, bread board, Resistor (220 Ω), Male to female wires, Laptop/PC, LED Lights, Jumper wires.

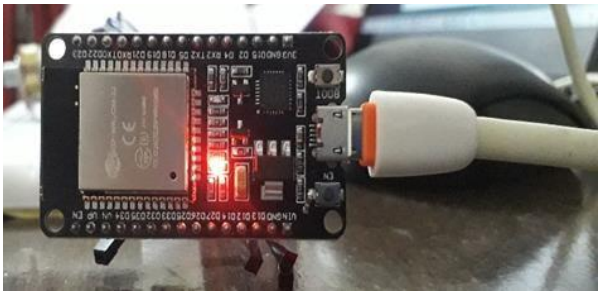
Theory:

Cyber-Physical Systems: CPS refers to systems that integrate physical processes with computer-based control and monitoring. These systems can be found in various applications, such as industrial automation, autonomous vehicles, smart cities, and more. CPS combines real-world, physical components with computational elements to make decisions and control actions.

In this practical an ESP32 LED web server is built from scratch. The following steps are performed-first setting up the ESP32, connecting the LEDs, and creating a web interface to control them.

C**onnections:**

1. First connect the data cable to the ESP 32 board, check out the notch and insert the cable in straight manner, without any tilt.

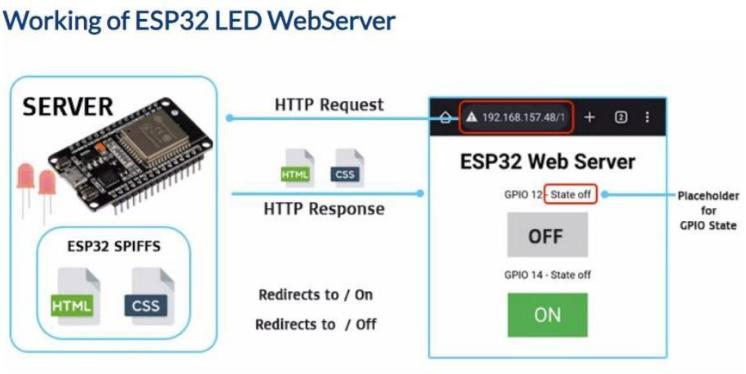


2. Connect the other end of the cable to the USB port of Laptop /PC and you should see a blue light glows.

3. In order to make our web server work:

- Connect LEDs on GPIO12 and GPIO14 on ESP32 so that you can control them
- After the coding is done. Try to access the control webpage by inputting the IP address of esp32 in the browser
- Clicking the buttons on the webpage to turn the LED “ON” or “OFF”.

Working of ESP32 LED WebServer



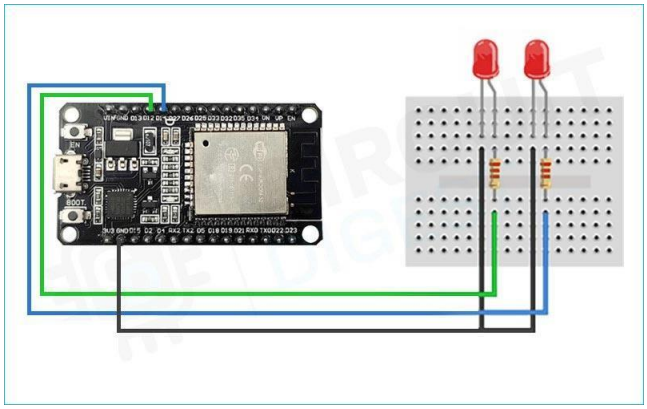
The program works by creating a web server on the ESP32 microcontroller that serves a web page to a client device, such as a computer or mobile phone, over Wi-Fi. The web page contains controls that allow you to turn the LED lights on or off.

The step-by-step breakdown of how the ESP32 LED webserver program works:

1. The ESP32 microcontroller is connected to the LED lights and programmed to serve as a web server.
2. The web server code is uploaded to the ESP32 using the Arduino IDE or other programming tools.
3. The ESP32 is connected to a Wi-Fi network, allowing it to communicate with client devices over the network.
4. When a client device connects to the ESP32's IP address, the web server code serves a web page to the client device's web browser.
5. The web page contains controls for the LED lights, allowing the user to turn them on or off.
6. When the user interacts with the controls on the web page, the ESP32 microcontroller receives the command and adjusts the LED lights accordingly.
7. The ESP32 sends a response back to the client device, updating the web page with the current state of the LED lights.

Circuit Diagram of ESP32 LED WebServer

Below is the circuit diagram to control an **LED using ESP32 based webserver**:



The ESP32 microcontroller has many GPIO pins that can be used for interfacing with external devices, such as LED lights. In this example, we will interface the ESP32 with two LED lights connected to GPIO pins 12 and 14.

To interface the ESP32 with two LED lights:

- Connect one end of a 220-ohm resistor to GPIO pin 12 on the ESP32, and connect the other end of the resistor to the positive (anode) leg of one LED light.

- Connect the negative (cathode) leg of the LED light to the ground (GND) on the ESP32.

- Repeat steps 1 and 2 for GPIO pin 14 and the other LED light.

Uploading the code on the ESP32 board:

To upload code to an ESP32 board, you'll need to follow these steps:

1. Connect your ESP32 board to your computer using a USB cable.
2. Open the Arduino IDE and go to "Tools" > "Board" and select your ESP32 board from the list.
3. Make sure to also select the correct port under "Tools" > "Port".
4. Write or copy your code into the Arduino IDE.
5. Verify that your code compiles by clicking the "Verify" button (checkmark icon) in the top-left corner of the IDE. If there are any errors, fix them before proceeding.
6. Upload your code to the ESP32 board by clicking the "Upload" button (right arrow icon) in the top-left corner of the IDE. The IDE will compile your code and then upload it to the ESP32 board. The progress of the upload will be shown in the bottom of the IDE.
7. Once the upload is complete, the ESP32 board will automatically reset and start running your code.

Finding the IP address of the ESP32 board

1. Open the Serial Monitor in the Arduino IDE by clicking on the magnifying glass icon in the top right corner of the IDE window.

2. Select a baud rate of 115200 from the dropdown list in the bottom right corner of the Serial Monitor window.
3. Reset the ESP32 board by pressing the EN(RST) button on the board.
4. Wait for the ESP32 board to connect to the WiFi network. The Serial Monitor will display messages that indicate the progress of the connection.
5. Once the ESP32 board is connected to the WiFi network, it will obtain an IP address from the network. The Serial Monitor will display the IP address of the ESP32 board.

Connecting to the ESP32 Webserver and testing it

1. Open the same network through a device or use the same device to whose hotspot your ESP32 is connected
2. Open a web browser on your computer or mobile device and enter the IP address of the ESP32 board into the address bar.

3. After you've accessed the control page on your browser you can test it by clicking "ON" and "OFF" and checking the LED state and serial monitor simultaneously

4. Now if you click on the GPIO 12 button, you will see that the serial monitor gets a request on 12/on URL and then your led lights up.

5. Now if you click on the GPIO 12 button, you will see that the serial monitor gets a request on 12/on URL and then your led lights up.

6. After this is done, the Esp32 updates its state on the webpage as "ON" or "OFF".

7. This will work the same for GPIO 14 too.

Conclusion: Thus the experiment is performed by writing a program to design and develop a user interface for monitoring and controlling CPS system.

Program Code

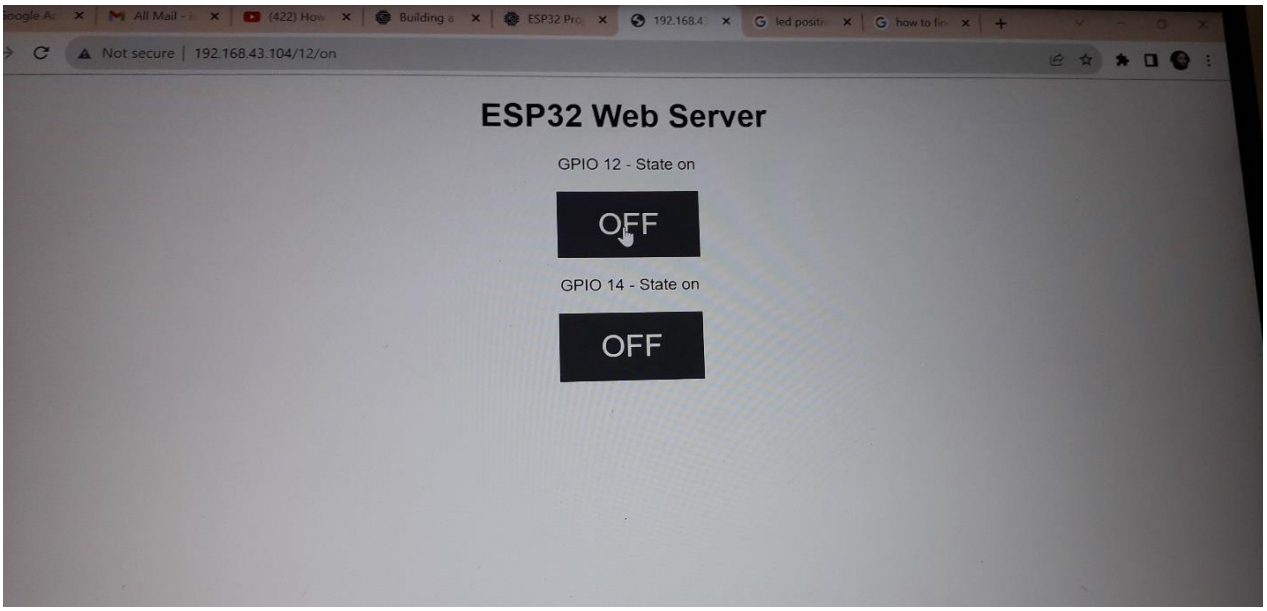
```
// Load Wi-Fi library
#include <WiFi.h>
// Replace with your network credentials
const char* ssid ="Shilpas";
const char* password = "shilpa19";
// Set web server port number to 80
WiFiServer server(80);
// Variable to store the HTTP request
String header;
// Auxiliary variables to store the current output state
String output12State = "off";
String output14State = "off";
// Assign output variables to GPIO pins
const int output12 = 12;
const int output14 = 14;
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output12, OUTPUT);
  pinMode(output14, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output12, LOW);
  digitalWrite(output14, 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();
```

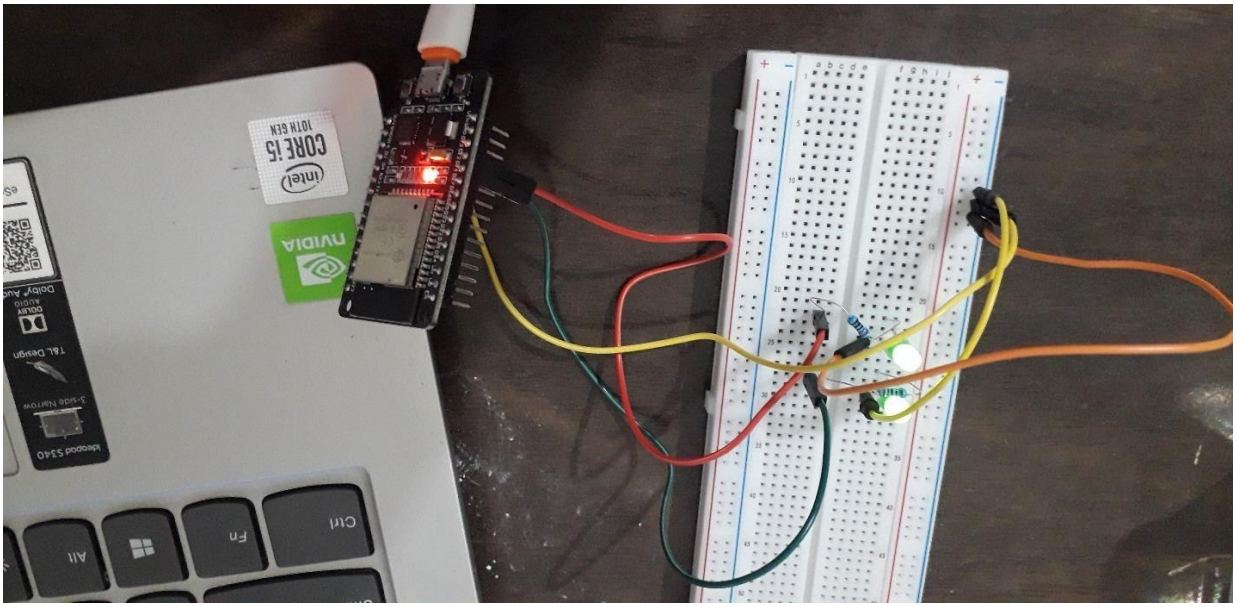
```
}
void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients
  if (client) {                             // If a new client connects,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");          // print a message out in the serial
port
    String currentLine = "";                // make a String to hold incoming
data from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) { //
loop while the client's connected
      currentTime = millis();
      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 /12/on") >= 0) {
              Serial.println("GPIO 12 on");
              output12State = "on";
              digitalWrite(output12, HIGH);
            } else if (header.indexOf("GET /12/off") >= 0) {
              Serial.println("GPIO 12 off");
              output12State = "off";
              digitalWrite(output12, LOW);
            } else if (header.indexOf("GET /14/on") >= 0) {
              Serial.println("GPIO 14 on");
              output14State = "on";
              digitalWrite(output14, HIGH);
            } else if (header.indexOf("GET /14/off") >= 0) {
```

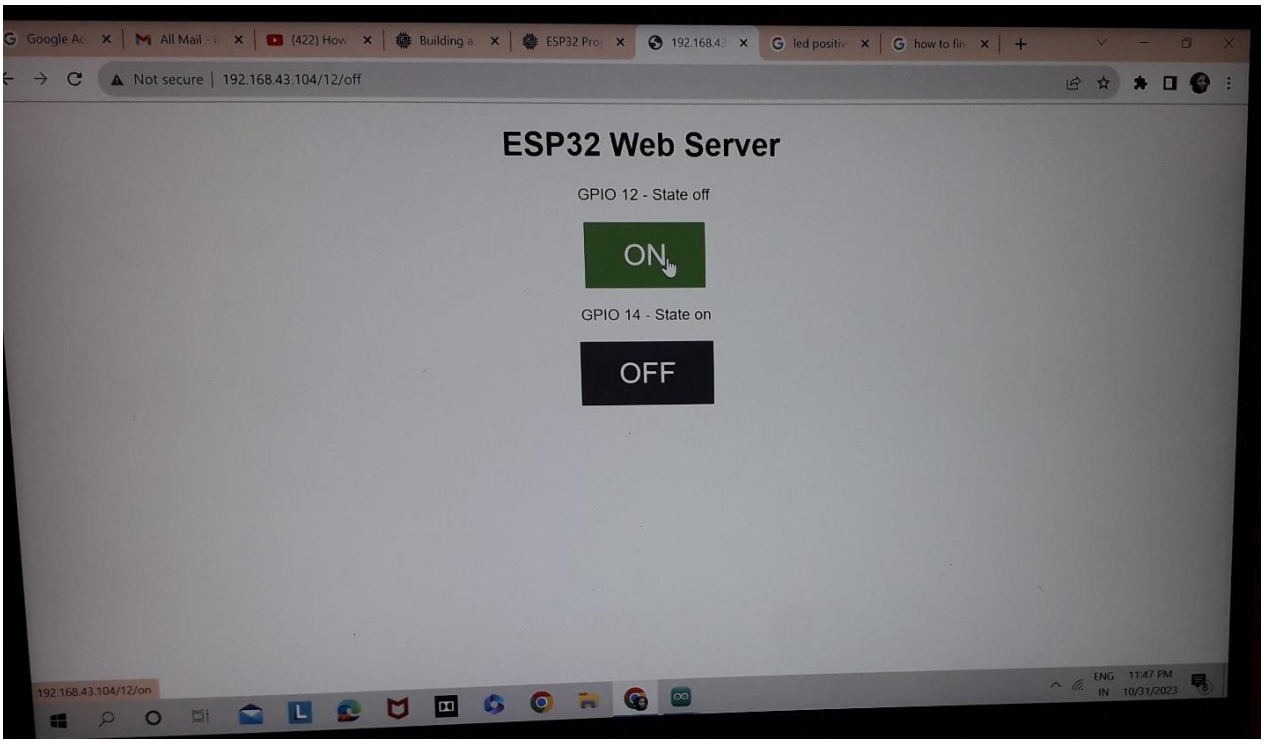
```
              Serial.println("GPIO 14 off");
              output14State = "off";
              digitalWrite(output14, LOW);
            }
            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-
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 12
            client.println("<p>GPIO 12 - State " + output12State + "</p>");
            // If the output12State is off, it displays the ON button
            if (output12State=="off") {
              client.println("<p><a href=\"/12/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/12/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
            // Display current state, and ON/OFF buttons for GPIO 14
            client.println("<p>GPIO 14 - State " + output14State + "</p>");
            // If the output14State is off, it displays the ON button
            if (output14State=="off") {
              client.println("<p><a href=\"/14/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/14/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
            client.println("</body></html>");
            // The HTTP response ends with another blank line
            client.println();
```

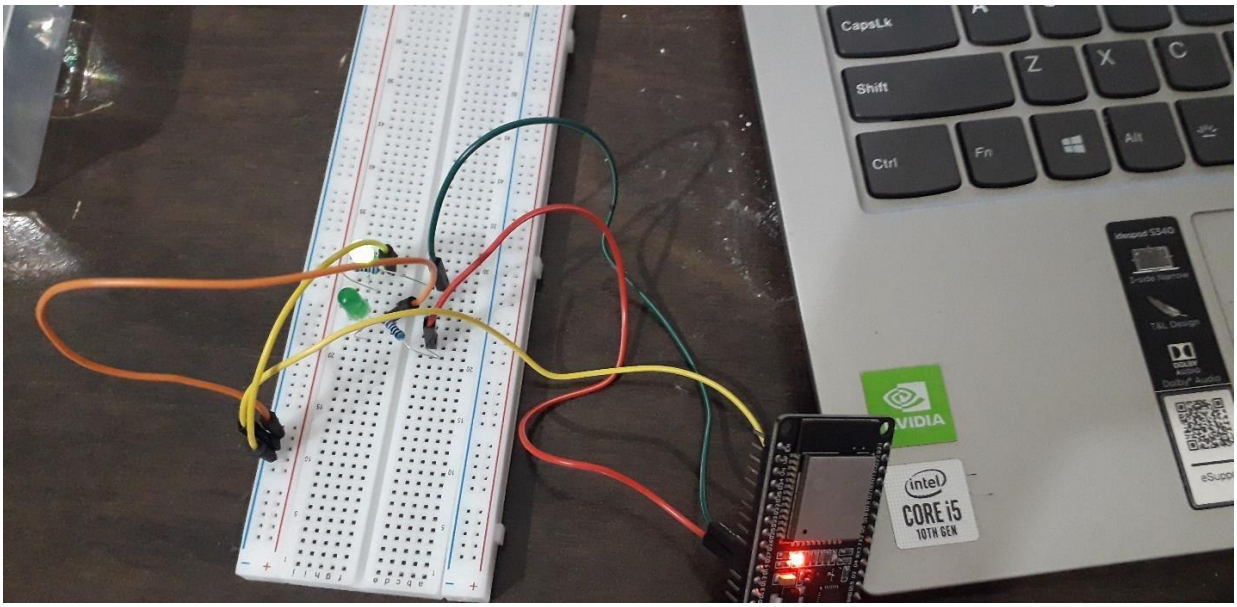
```
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage
return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

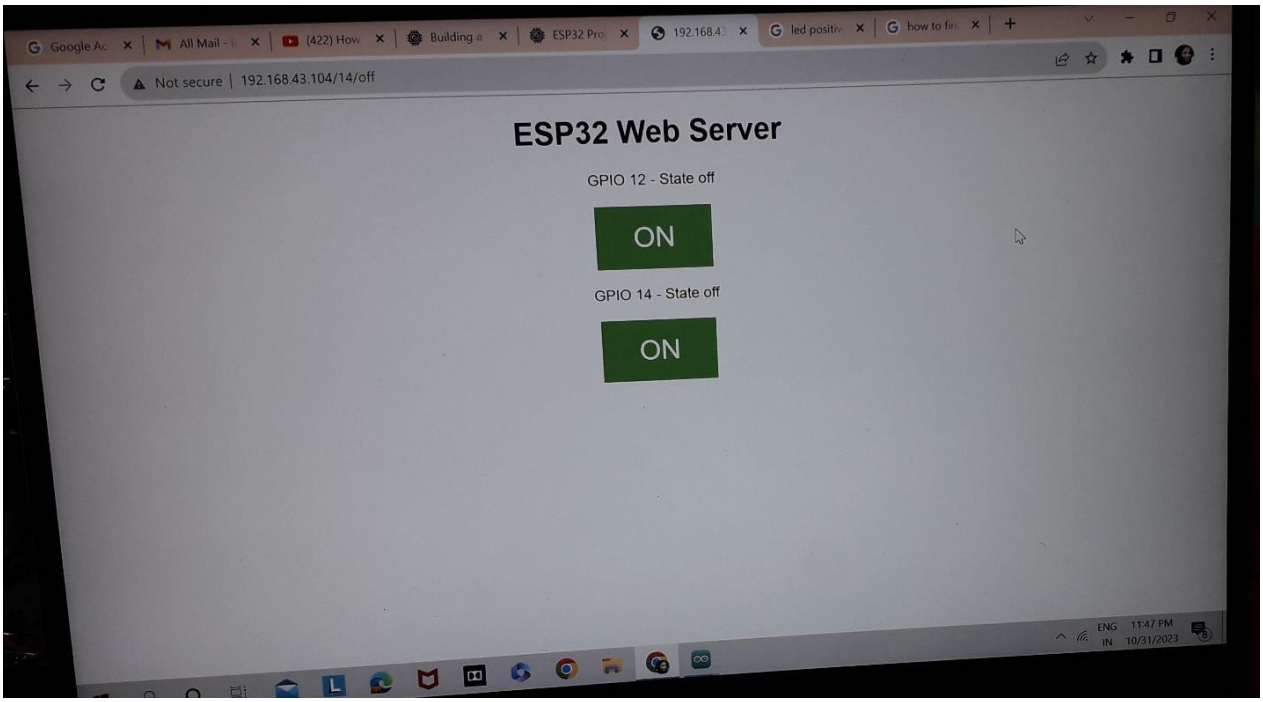
OUTPUT

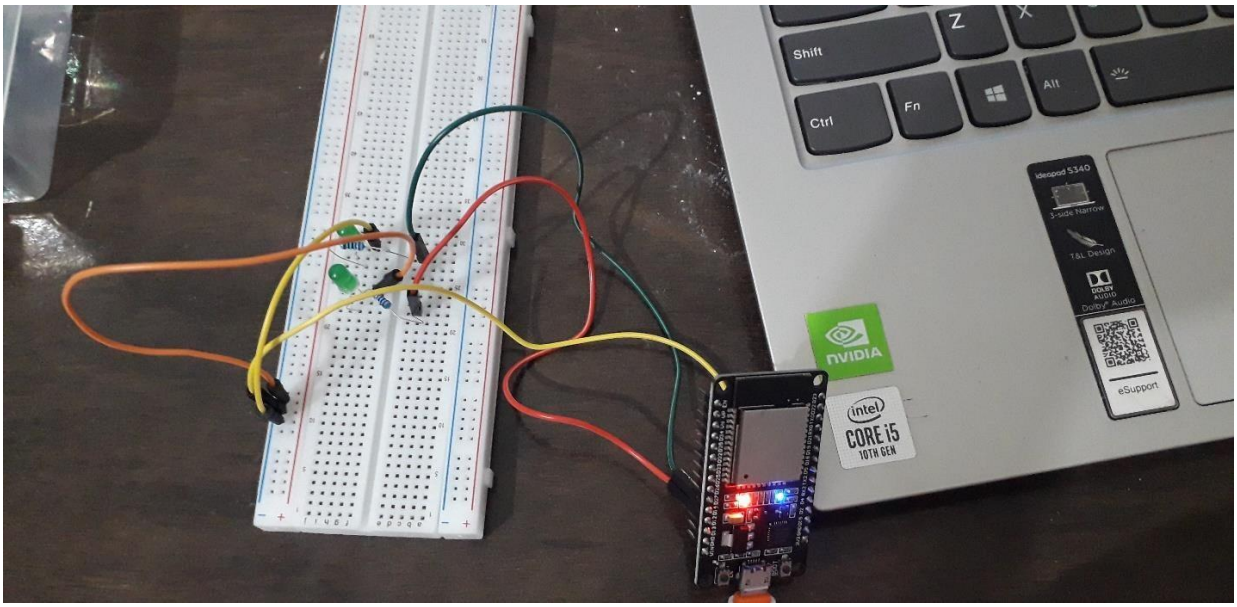

Not secure | 192.168.43.104/12/on
ESP32 Web Server
GPIO 12 - State on
OFF
GPIO 14 - State on
OFF




Not secure | 192.168.43.104/12/off
ESP32 Web Server
GPIO 12 - State off
ON
GPIO 14 - State on
OFF
192.168.43.104/12/on


CapsLk
Shift
Ctrl
Fn
Alt
Z
X
C
intel
CORE i5
10TH GEN


Not secure | 192.168.43.104/14/off
ESP32 Web Server
GPIO 12 - State off
ON
GPIO 14 - State off
ON


Shift
Ctrl
Fn
Alt
NVIDIA
intel
CORE i5
10TH GEN