VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering (SENSE)

PROJECT BASED LEARNING (J Component) - REPORT

COURSE CODE / NAME	ECM2001– Data Communication Networks		
PROGRAM / YEAR	B. Tech (Electronics and Computer Engineering)		
LAST DATE FOR REPORT SUBMISSION			
DATE OF SUBMISSION	16-11-22		
TEAM MEMBERS DETAILS	**REGISTER NO.**	**NAME**	
	20BLC1087	Aryan Pandey	
	20BLC1016	Ananya B	
	-	-	
	-	-	
-	-	-	
J TITLE	**OFFICE AUTOMATION USING ESP32**		
COURSE HANDLER'S NAME	**Dr. T. Jayavignesh**	**REMARKS**	
COURSE HANDLER'S SIGN			

Table of Content

ABSTRACT:

This paper describes how to use an ESP32-based web server to monitor and/or gather statistics on the PV and battery current and voltage in a small photovoltaic (PV) power system. The developed system makes use of inexpensive sensors, an SD card reader, Wi-Fi, and a microcontroller called ESP32.

Data is gathered from sensors by the ESP32. The data are saved on an SD card that is attached to the ESP32 using SPI pins as a text file. The system keeps the text file on the SD card for a week or more before deleting it and moving on to saving new data.

The ESP32 is configured to view the web page using Wi-Fi from a laptop, smartphone, or tablet because the web page file is kept on an SD card as well. Additionally, visitors can download the data text file from the website by clicking on a link on the page.

This is also possible from a distance. The studies' findings show that the web server operates in real-time and is useful for keeping an eye on modest solar energy plants.

INTRODUCTION:

The ESP32 family of system on a chip microcontroller features integrated Wi-Fi and dual-mode Bluetooth and is inexpensive and low power. The Tensilica Xtensa LX6 dual-core or single-core microprocessor, Tensilica Xtensa LX7 dual-core, or a single-core RISC-V microprocessor is used in the ESP32 series, which also has integrated antenna switches, RF baluns, power amplifiers, low-noise receive amplifiers, filters, and power-management modules. Chinese business Espressif Systems, with headquarters in Shanghai, invented and constructed the ESP32, which is produced by TSMC using their 40 nm technology. It is the ESP8266 microcontroller's replacement.

Features of the ESP32 include the following:

- Processors:
  - CPU: Xtensa dual-core 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
  - Ultra-low power co-processor
- Memory: 320 KiB RAM, 448 KiB ROM
- Wireless connectivity:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR and BLE
- Peripheral interfaces:
  - 34 × programmable GPIOs
  - 12-bit SAR ADC up to 18 channels
  - 2 × 8-bit DACs
  - 10 × touch sensors (capacitive sensing GPIOs)
  - 4 × SPI
  - 2 × I²S interfaces
  - 2 × I²C interfaces

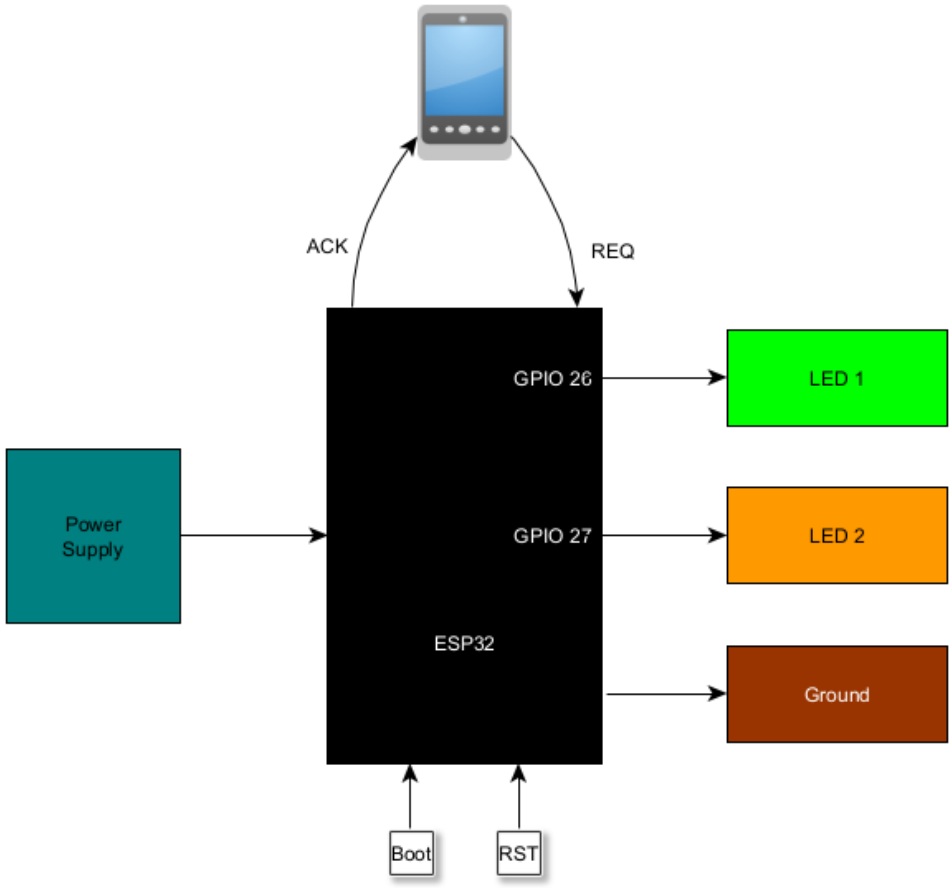
    - 3 × UART
    - SD/SDIO/CE-ATA/MMC/eMMC host controller
    - SDIO/SPI slave controller
    - Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time Protocol support
    - CAN bus 2.0
    - Infrared remote controller (TX/RX, up to 8 channels)
    - Pulse counter (capable of full quadrature decoding)
    - Motor PWM
    - LED PWM (up to 16 channels)
    - Hall effect sensor
    - Ultra-low power analog pre-amplifier
- Security:
    - IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 and WLAN Authentication and Privacy Infrastructure
    - Secure boot
    - Flash encryption
    - 1024-bit OTP, up to 768-bit for customers
    - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
- Power management:
    - Internal low-dropout regulator
    - Individual power domain for RTC
    - 5 µA deep sleep current
    - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

ALGORITHM:

```
START
Set SSID & Password for the ESP32 WiFi Module
Initialize Web server port number to 80
Initialize variales to store HTTP request and current output state
Initialize output variables to GPIO pins
Start the WiFi client to listen for incoming clients
If new client connects
     loop while client is connected
          if two newline inputs in a row exit
               close the connection
          else check for GPIOs on and off

          Display the web page to client
          Do changes according to the input
     close the connection
END
```

BLOCK DIAGRAM:

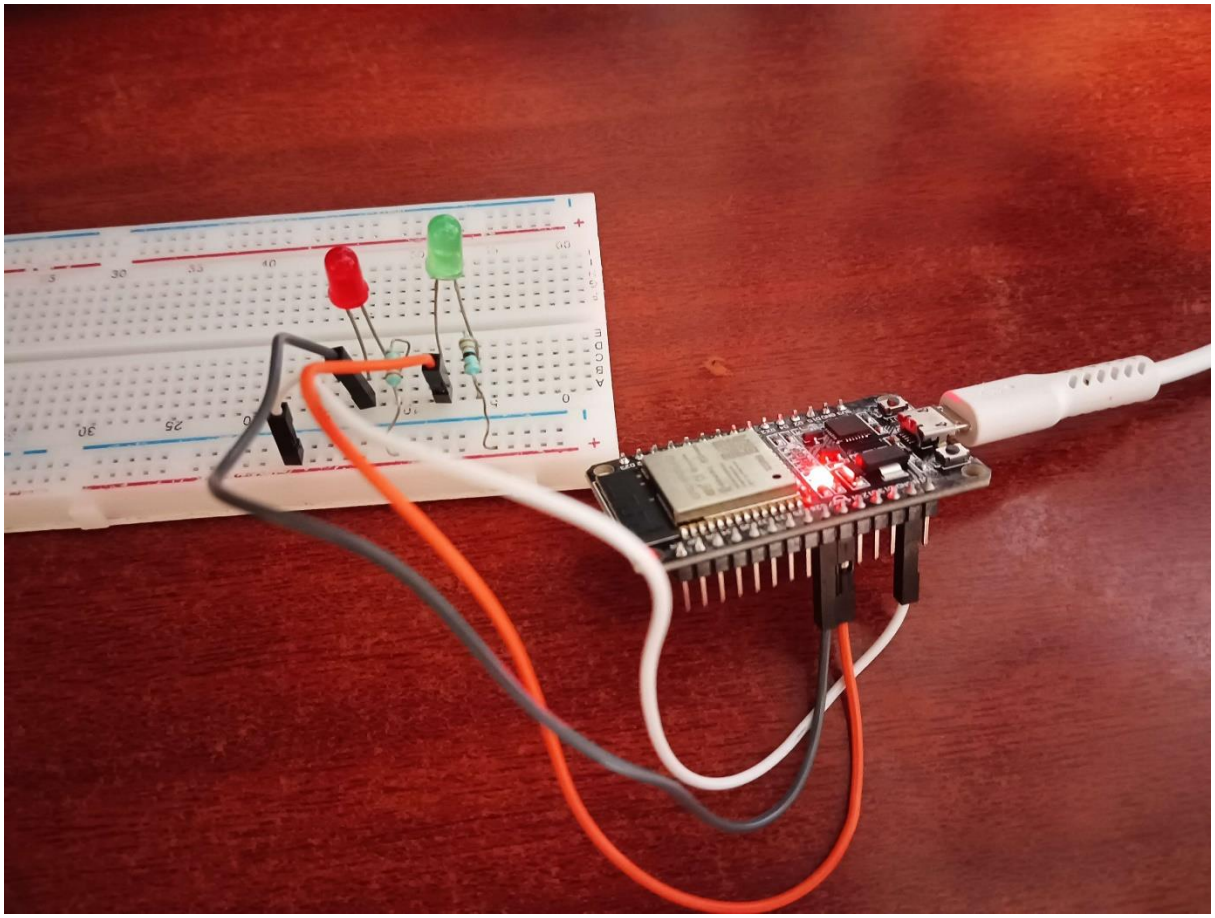


IMPLEMENTATION (REAL TIME):

It is important to outline what our web server will do: -

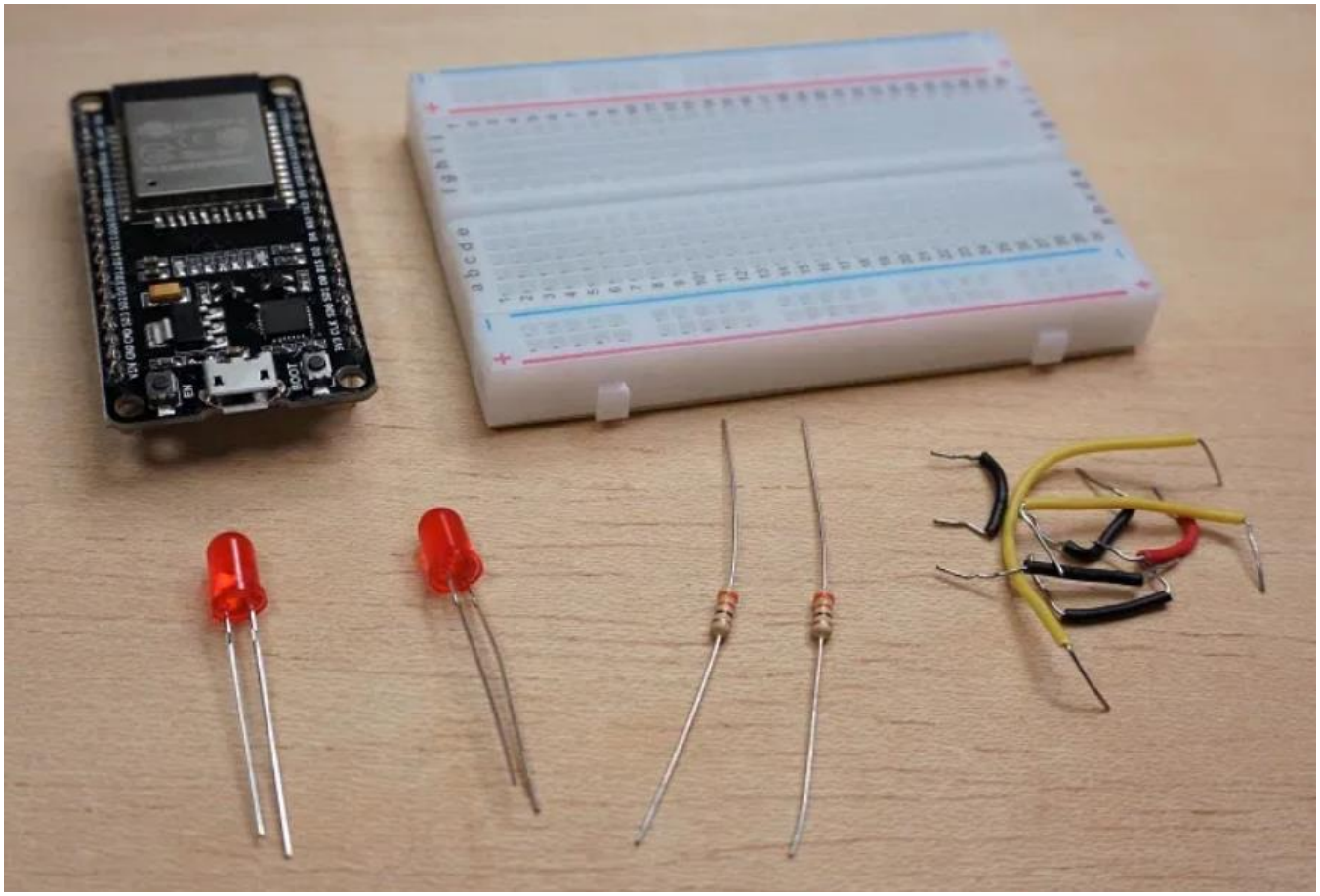
- The web server you'll build controls two LEDs connected to the ESP32 GPIO 26 and GPIO 27;
- You can access the ESP32 web server by typing the ESP32 IP address on a browser in the local network;
- By clicking the buttons on your web server, you can instantly change the state of each LED.

This is just a simple example to illustrate how to build a web server that controls outputs, the idea is to replace those LEDs with a relay, or any other electronic components you want in your office/home automation.

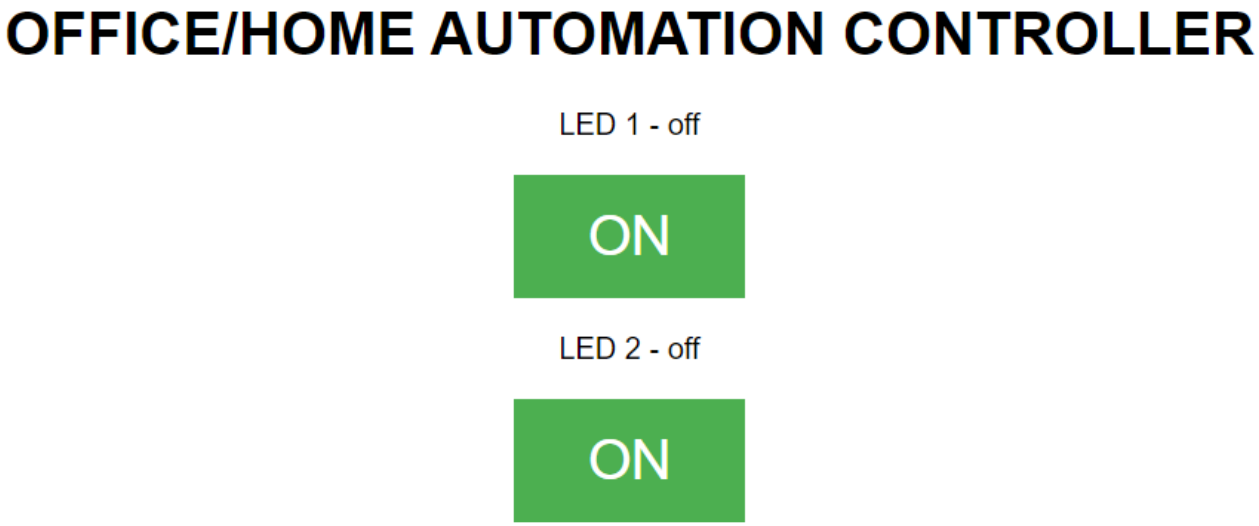


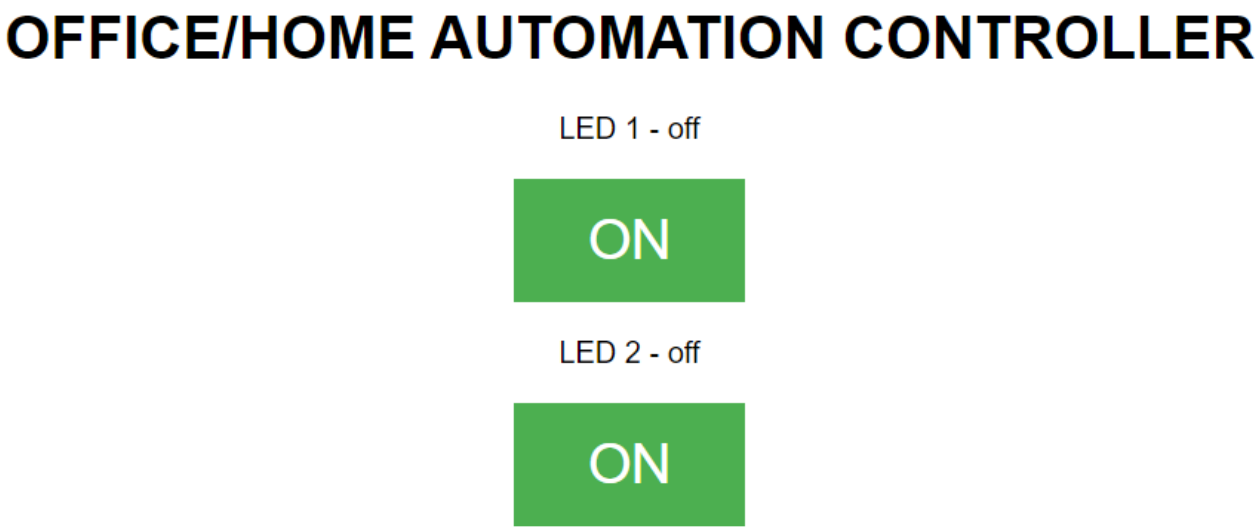
This is how the connections look like.

Components Required: -

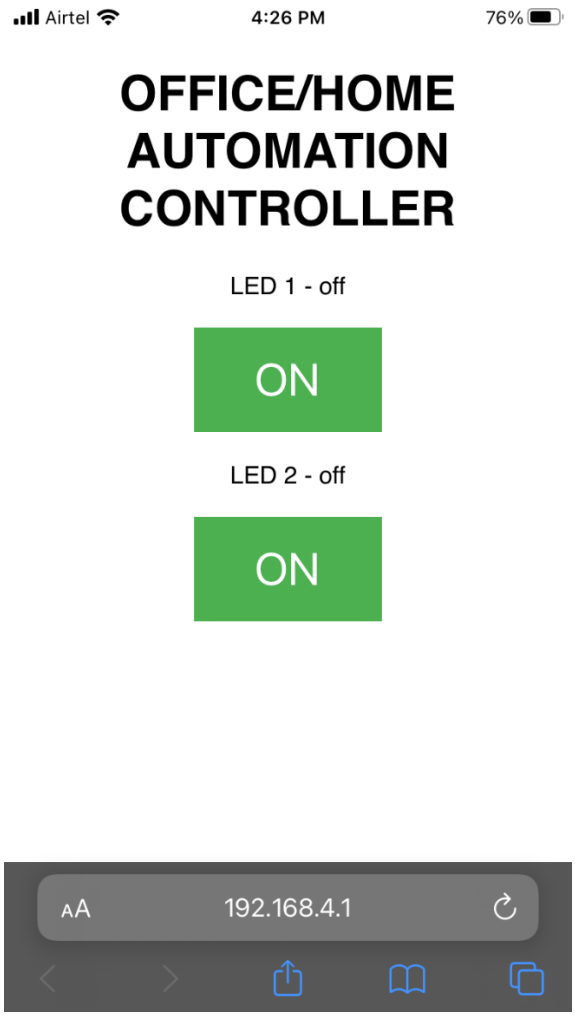


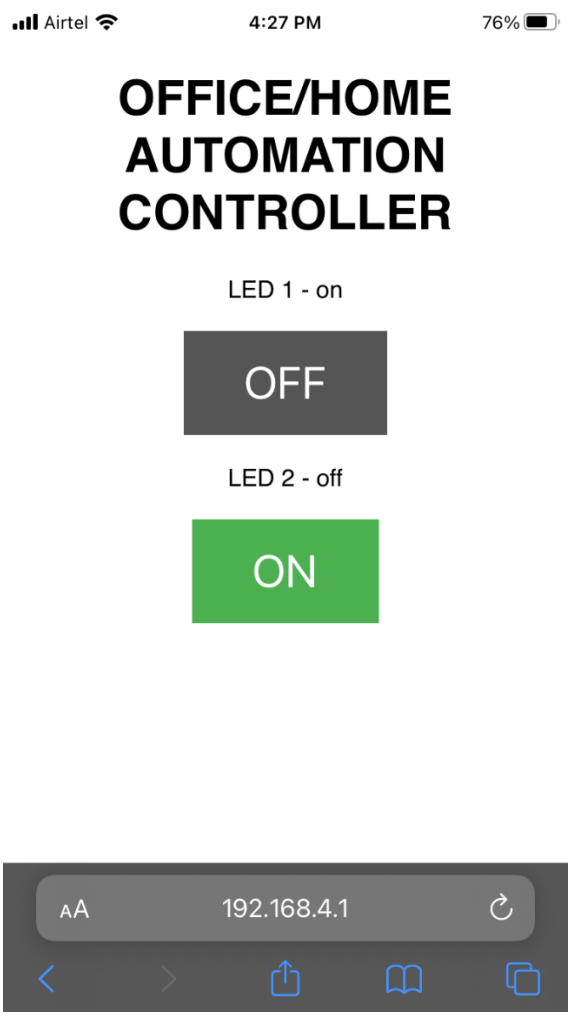
Web Page: - (We accessed this web page using the IP Address generated by the ESP32 in serial monitor)

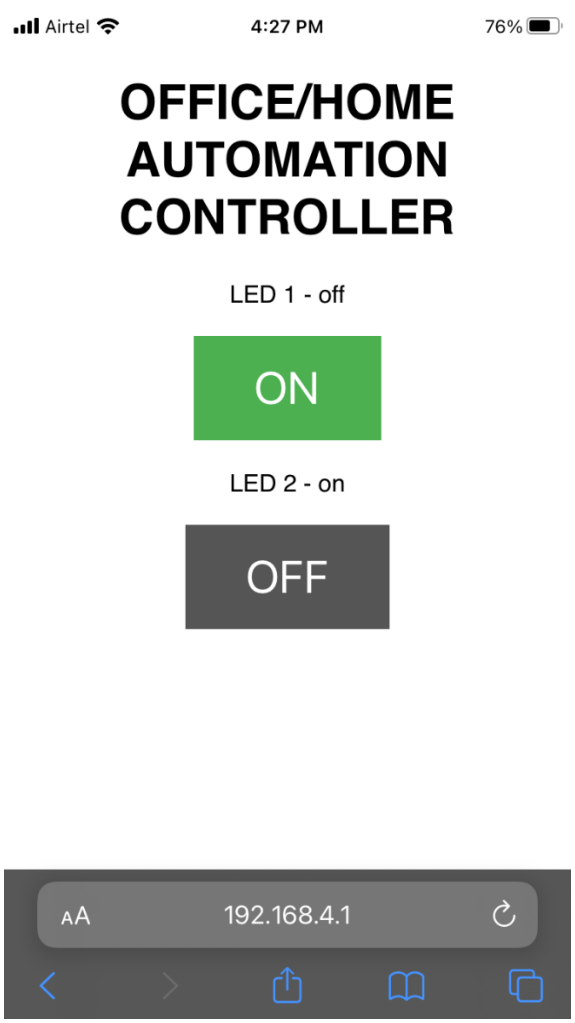


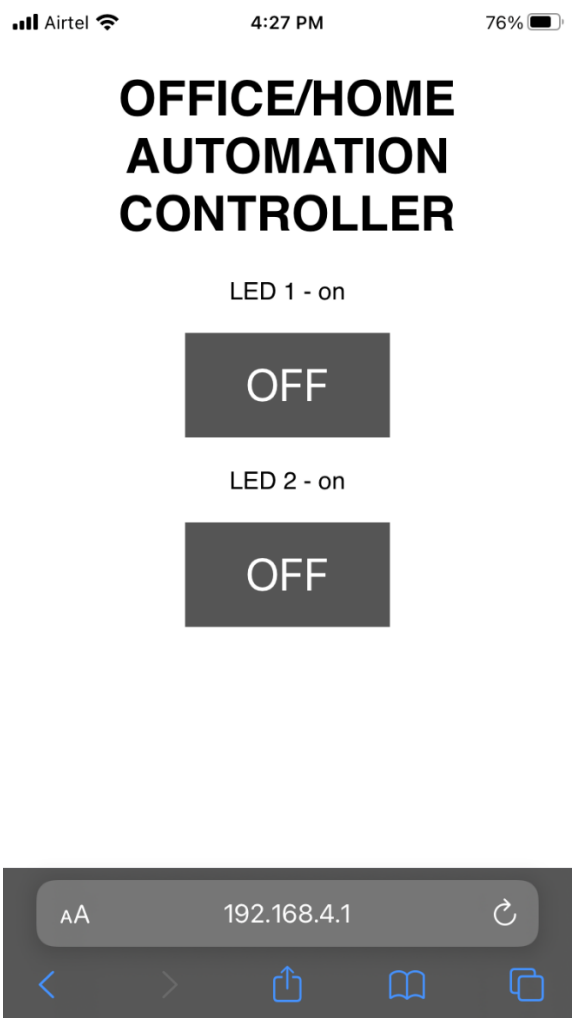


Phone GUI: -


Airtel
4:26 PM
76%
OFFICE/HOME AUTOMATION CONTROLLER
LED 1 - off
ON
LED 2 - off
ON
AA
192.168.4.1


Airtel
4:27 PM
76%
OFFICE/HOME AUTOMATION CONTROLLER
LED 1 - on
OFF
LED 2 - off
ON
AA
192.168.4.1

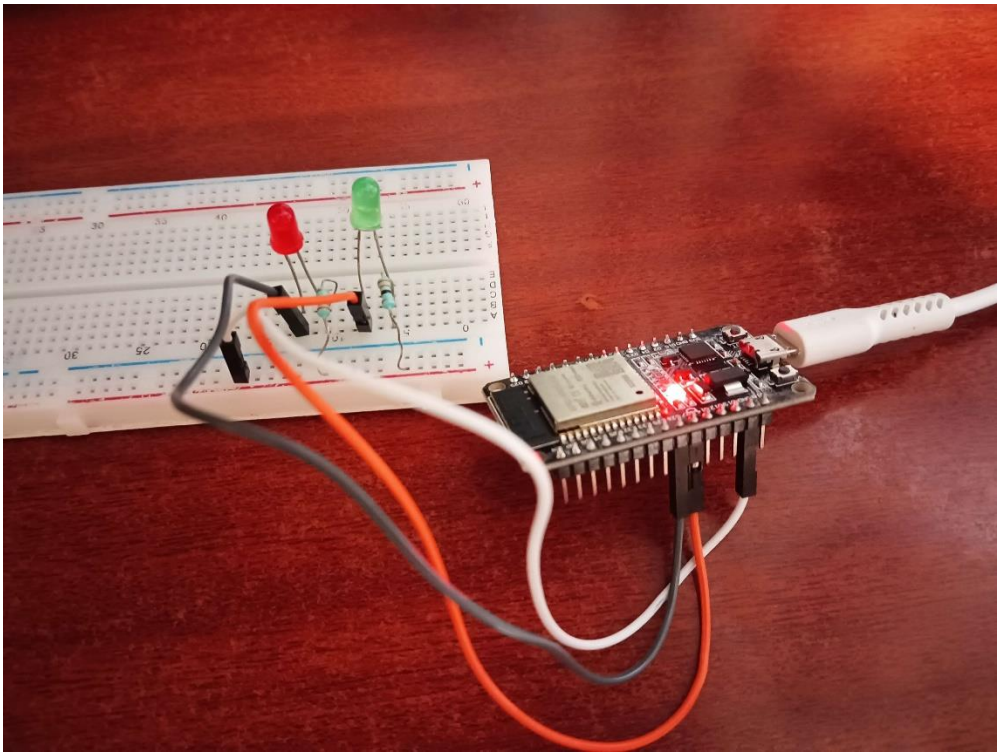




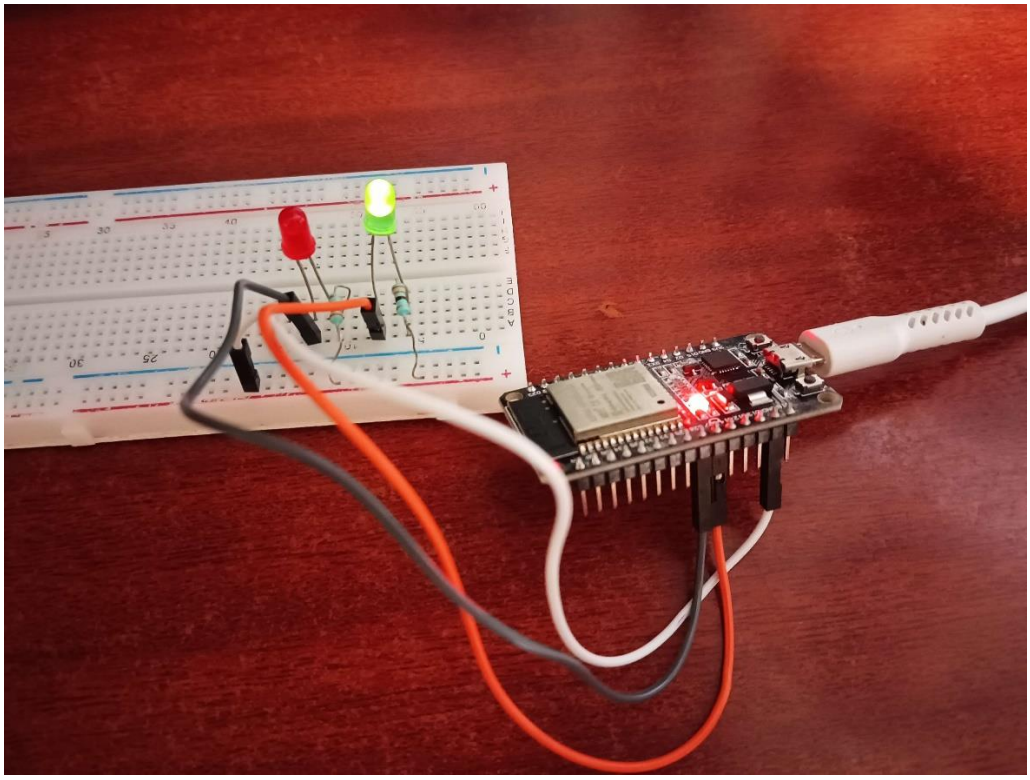
LED 1 is connected to the GPIO PIN no. 26
LED 2 is connected to the GPIO PIN no. 27

OUTPUTS: -

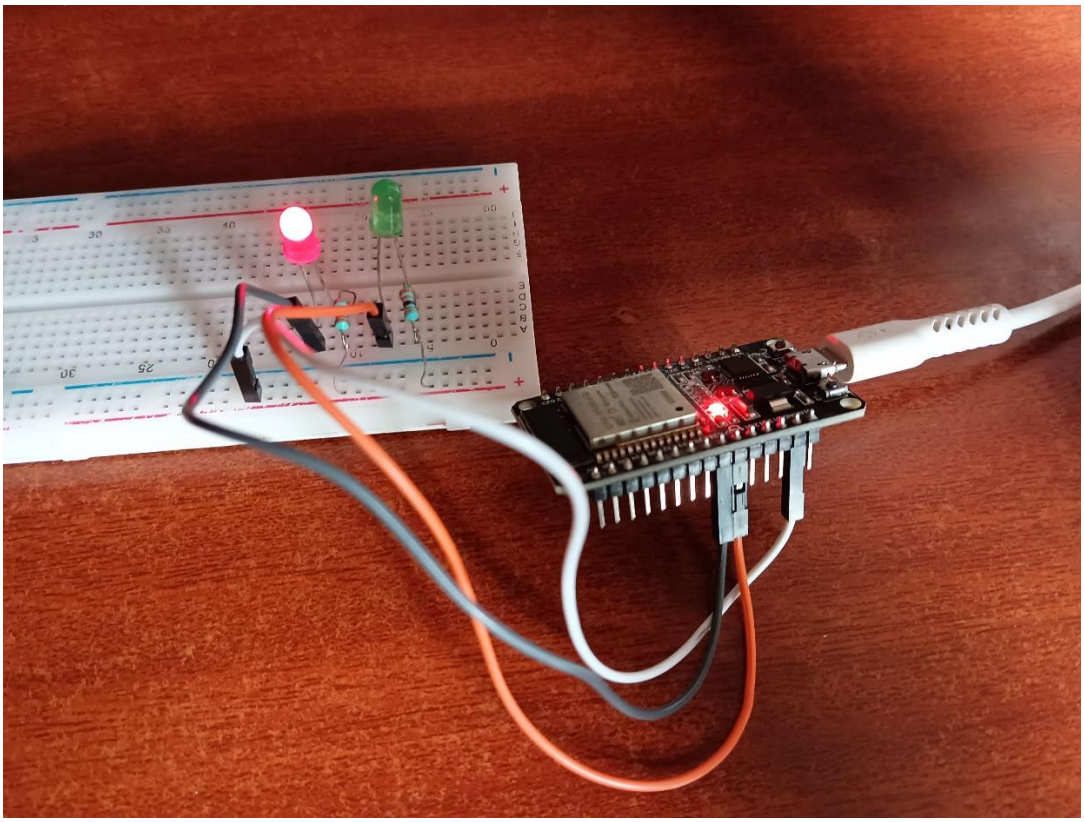
BOTH OFF: -



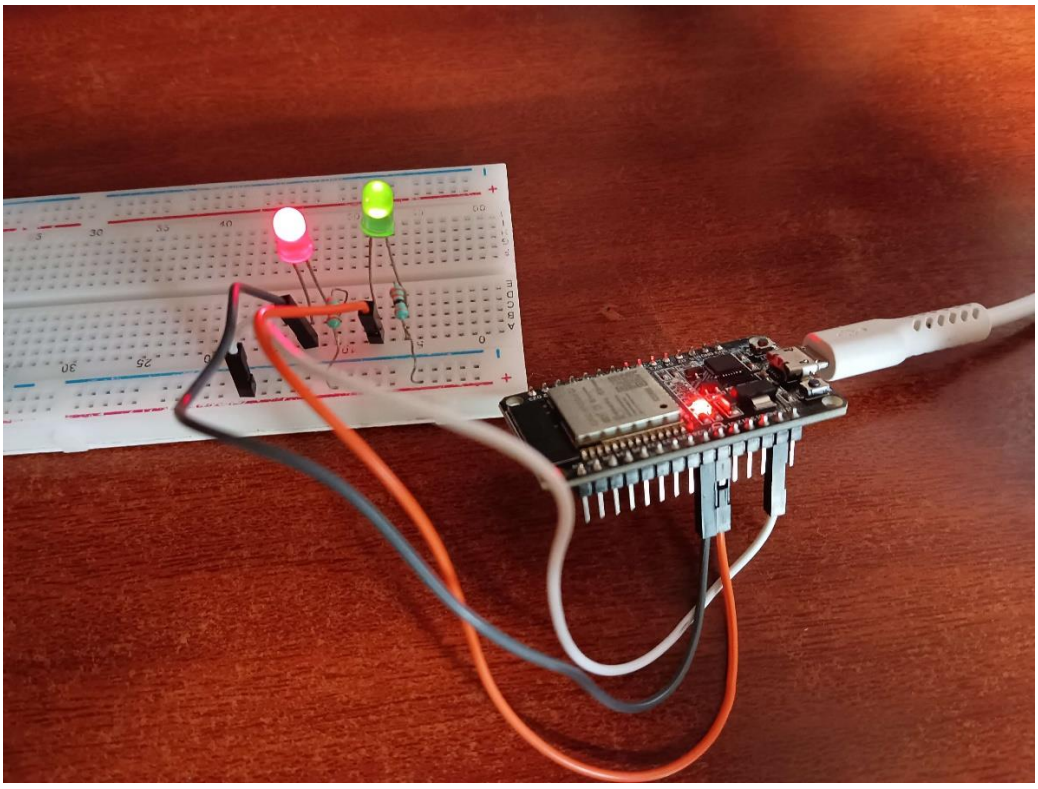
GPIO 27 (STATE ON): -



GPIO 26 (STATE ON): -



BOTH ON: -



Product testing was done in the Arduino Uno IDE and the output was also verified in the serial monitor: -

```
New Client.
GET / HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.(
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

Client disconnected.
```

```
GET /26/on HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
DNT: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.(
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
Referer: http://192.168.4.1/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

GPIO 26 on
Client disconnected.
```

```
GET /27/on HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
DNT: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.(
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
Referer: http://192.168.4.1/26/on
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

GPIO 27 on
Client disconnected.
```

```
GPIO 26 off
Client disconnected.

New Client.
GET /27/off HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
DNT: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.4.1/26/off
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

GPIO 27 off
Client disconnected.
```

CODE: -

GITHUB LINK: - https://github.com/aryanpandey3333/WEB_SERVER_USING_ESP_32.git

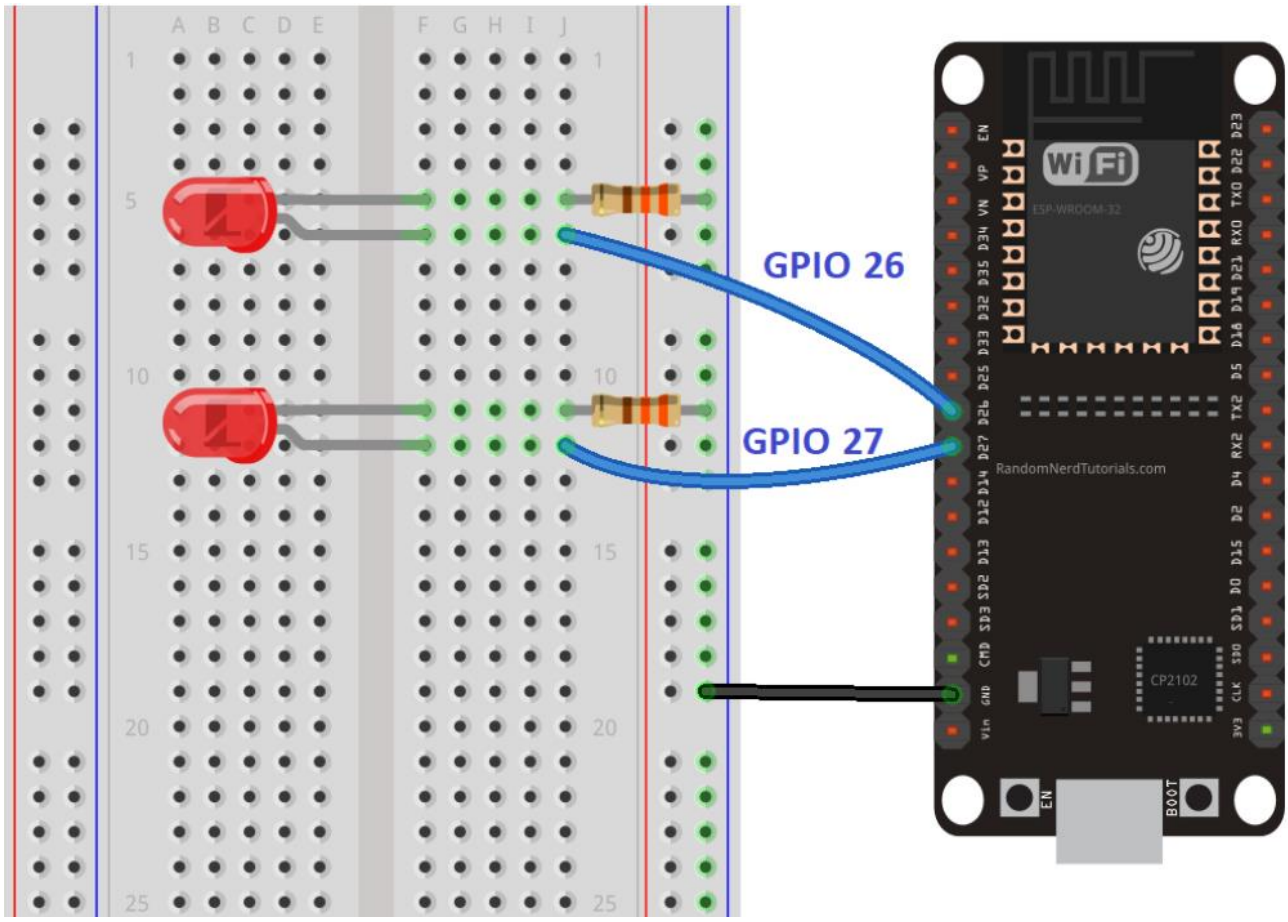
```
#include <WiFi.h>
const char* ssid     = "ESP32-Access-Point";
const char* password = "123456789";
WiFiServer server(80);
String header;
String output26State = "off";
String output27State = "off";
const int output26 = 26;
const int output27 = 27;
void setup() {
  Serial.begin(115200);
  pinMode(output26, OUTPUT);
  pinMode(output27, OUTPUT);
  digitalWrite(output26, LOW);
  digitalWrite(output27, LOW);
  Serial.print("Setting AP (Access Point");
  WiFi.softAP(ssid, password)
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);
  server.begin();
}
void loop(){
  WiFiClient client = server.available();
  if (client) {
    Serial.println("New Client.");
    String currentLine = "";
    while (client.connected())
     {
      if (client.available())
        char c = client.read();
        Serial.write(c);
        header += c;
        if (c == '\n')
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            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);
            }
            client.println("<!DOCTYPE html><html>");
            client.println("<head><metaname=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            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>OFFICE/HOME AUTOMATION
CONTROLLER</h1>");
            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p>LED 1 - " + 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>");
            }
            client.println("<p>LED 2 - " + 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>");
            client.println();
            break;
          } else {
            currentLine = "";
```

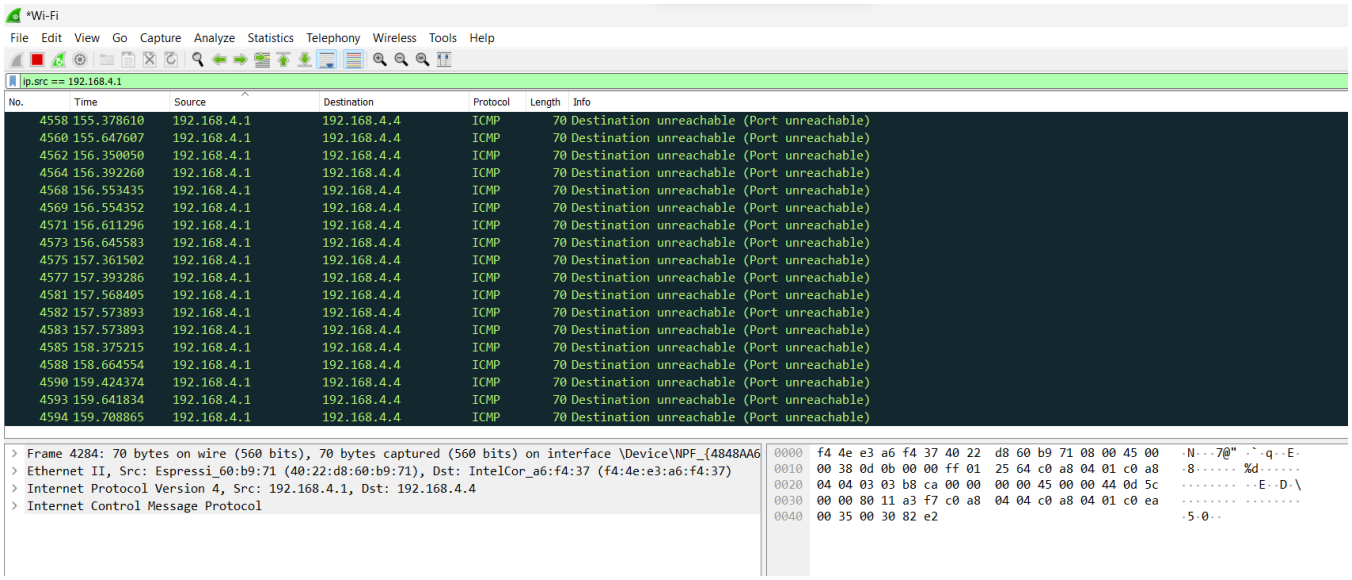
```
          }
        } else if (c != '\r') {
          currentLine += c;
        }
      }
    }
    header = "";
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

RESULTS & INFERENCES:

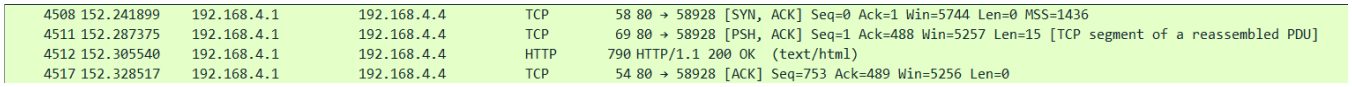
Schematic: -

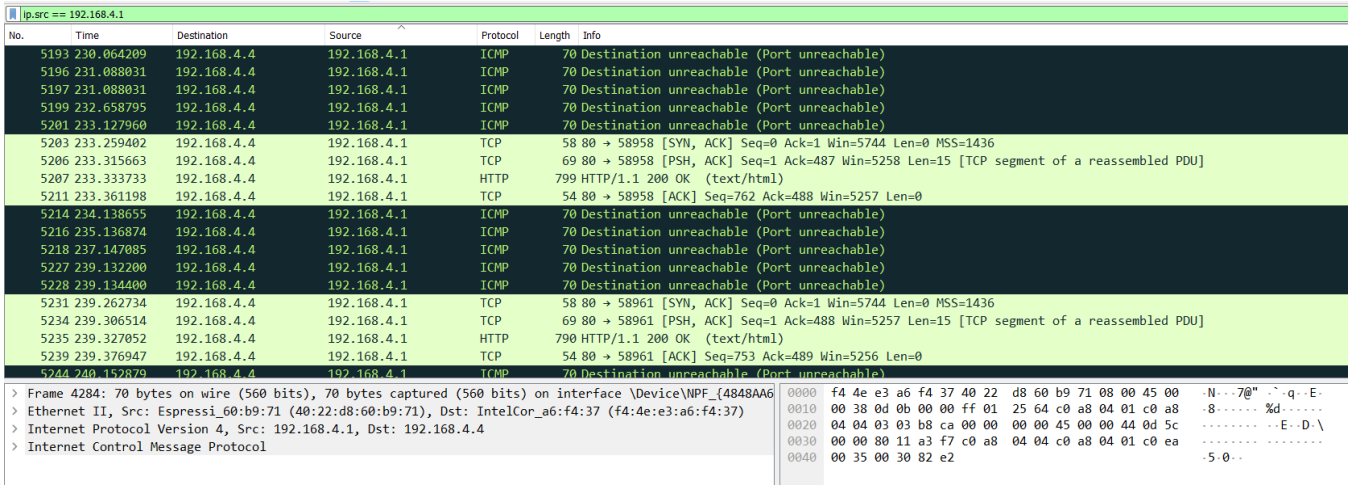


It keeps on pinging the port to see whether the destination port is reachable or not.



When the on and off buttons are pressed the protocols to send the data changes from ICMP to TCP and HTTP.





As shown above, the type of an ICMP packet contains the overall message that the message is intended to convey. For example, a type value of 3 means that the intended destination is unreachable. For some types, there are multiple code values intended to provide additional information.

Unlike the Transport Control Protocol (TCP) and User Datagram Protocol (UDP), the Internet Control Message Protocol (ICMP) is not designed for carrying data.

While ICMP packets do have a data section, their purpose is not to wrap and carry protocols like HTTP and DNS. Instead, ICMP is designed as a low-level management protocol for the internet. It carries error messages and implements simple management functions.

Type	Code	Description
0 – Echo Reply	0	Echo reply
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation needed and DF flag set
	5	Source route failed
5 – Redirect Message	0	Redirect datagram for the Network
	1	Redirect datagram for the host
	2	Redirect datagram for the Type of Service and Network
	3	Redirect datagram for the Service and Host
8 – Echo Request	0	Echo request
9 – Router Advertisement	0	Use to discover the addresses of operational routers
10 – Router Solicitation	0	
11 – Time Exceeded	0	Time to live exceeded in transit
	1	Fragment reassembly time exceeded
12 – Parameter Problem	0	Pointer indicates error
	1	Missing required option
	2	Bad length
13 – Timestamp	0	Used for time synchronization
14 – Timestamp Reply	0	Reply to Timestamp message

In HTTP GET response, the requested data (e.g., an HTML page) is returned. Wireshark will show the hex dump of the data in a new tab “Uncompressed entity body” in the “Packet Bytes” pane.

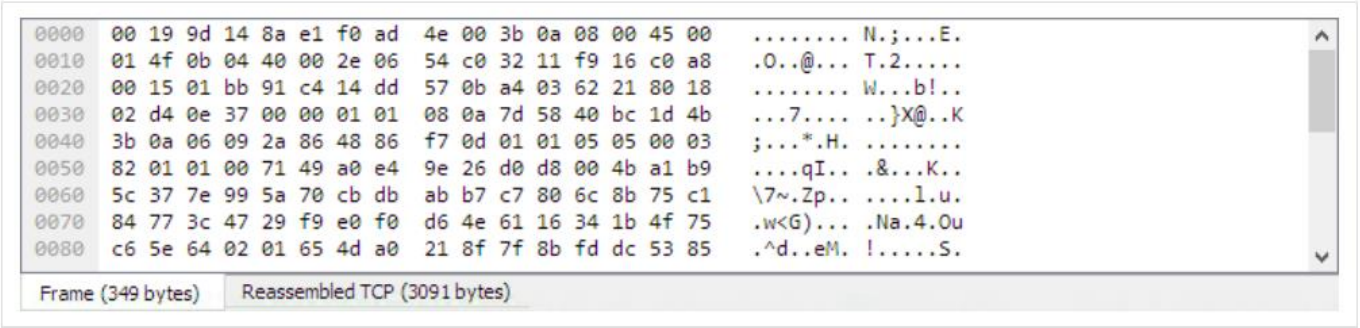
Reassembly is enabled in the preferences by default but can be disabled in the preferences for the protocol in question.

Enabling or disabling reassembly settings for a protocol typically requires two things:

- The lower-level protocol (e.g., TCP) must support reassembly. Often this reassembly can be enabled or disabled via the protocol preferences.
- The higher-level protocol (e.g., HTTP) must use the reassembly mechanism to reassemble fragmented protocol data. This too can often be enabled or disabled via the protocol preferences.

The tooltip of the higher-level protocol setting will notify you if and which lower-level protocol setting also has to be considered.

Figure 7.8. The "Packet Bytes" pane with a reassembled tab



Reassembly might take place at several protocol layers, so it's possible that multiple tabs in the "Packet Bytes" pane appear.

APPLICATION ORIENTED LEARNING:

Real-Time Application: -

- Smart industrial devices, including Programmable Logic Controllers (PLCs)
- Smart medical devices, including wearable health monitors.
- Smart energy devices, including HVAC and thermostats.
- Smart security devices, including surveillance cameras and smart locks.

Computer Communication Concepts Learned: -

- Measuring capacity of communication media
- Bandwidth
- Data transfer rate

Protocols we got more familiar with: -

- TCP

- ICMP
- HTTP

Cost of our project: -

If we just consider the basic prototyping it costs us around Rs.850

CONCLUSION:

One of the most useful features of the ESP32 is its ability to not only connect to an existing Wi-Fi network and act as a Web Server, but also to create its own network, allowing other devices to connect directly to it and access web pages. This project helps us understand many concepts.

This is possible because the ESP32 can operate in three modes: Station (STA) mode, Soft Access Point (AP) mode, and both simultaneously.

REFERENCES:

- https://www.youtube.com/watch?v=6zbEVAXVBjI
- https://www.youtube.com/watch?v=f6ovK57qPRg
- https://create.arduino.cc/projecthub/imjeffparedes/add-wifi-to-arduino-uno-663b9e
- https://github.com/stepansnigirev/ArduinoSerialToEthernet
- https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/
- https://www.youtube.com/watch?v=4moPYxNYjCY
- https://www.youtube.com/watch?v=ApGwxX6VVzk
- https://www.youtube.com/watch?v=G7Q3-XNkF3I
- https://www.youtube.com/watch?v=Hl0IpoS503A
- https://www.youtube.com/watch?v=3Ac6X6ZBQ0g
- https://create.arduino.cc/projecthub/425297/webservers-on-esp32-edffef
- https://www.youtube.com/watch?v=xOLG-88Ld3A
- https://microcontrollerslab.com/esp32-web-server-arduino-led/
- https://randomnerdtutorials.com/esp32-web-server-arduino-ide/
- GITHUB LINK: - https://github.com/aryanpandey3333/WEB_SERVER_USING_ESP_32.git